МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

В. В. АВРАМЕНКО, А. М. СКАКОВСЬКА

ПРОГРАМУВАННЯ НА VISUAL C++ ІЗ ЗАСТОСУВАННЯМ БІБЛІОТЕКИ MFC

Навчальний посібник

Рекомендовано Міністерством освіти і науки України

Суми
Сумський державний університет
2015

УДК 004.434(075.8)
ББК 32.973.2я73
А21

Рецензенти:
Є. П. Путятін – доктор технічних наук, професор, заслужений діяч науки і техніки України, завідувач кафедри інформатики Харківського національного університету радіоелектроніки;
М. Д. Кошовий – доктор технічних наук, професор, завідувач кафедри авіаційних приладів та вимірювань Національного аерокосмічного університету ім. М. Є. Жуковського «ХАІ»

Рекомендовано Міністерством освіти і науки України

як навчальний посібник для студентів

вищих навчальних закладів
(лист № 1/11-10429 від 08.07.2014)

Авраменко В. В.

А21 Програмування на Visual C++ із застосуванням бібліотеки MFC : навчальний посібник / В. В. Авраменко, А. М. Скаковська. – Суми : Сумський державний університет, 2015. – 215 с.
ISBN 978-966-657-558-9

Навчальний посібник містить стислий виклад теоретичного матеріалу до кожної теми, практичне завдання та зразок його вирішення. Проте, щоб дати початкове уявлення про програмування на C++/CLR, у посібнику наведена одна тема зі створення консольного застосування для версії C++.

Призначений для студентів вищих навчальних закладів, які навчаються за напрямом підготовки «Інформатика» і вивчають сучасні інформаційні технології у рамках дисциплін «Алгоритмічні мови» та «Системне програмування», а також для викладачів кафедр інформатики ВНЗ.

УДК 004.434(075.8)
ББК 32.973.2я73

ISBN 978-966-657-558-9

Зміст С.

Вступ

Метою навчального посібника є формування практичних навичок програмування у *Visual Studio C++* 2010. Проте перш ніж звернутися до специфіки *Windows* - програмування, необхідно добре вивчити засоби програмування *C++,* зокрема з його об'єктно-орієнтованими аспектами. Техніка об'єктно-орієнтованого програмування є ключем до ефективності всіх інструментів середовища *Visual C++*, що використовуються при створенні програм для *Windows.*

Версія системи розробки від корпорації *Microsoft Visual Studio C++ 2010* підтримує два різних, але тісно пов'язаних представлень мови *C++.* У ній реалізований вихідний стандарт *C++ ISO/ANSI.* Крім того, є нова версія *C++*, так звана *C++/CLI*, яка була розроблена *Microsoft* і тепер затверджена у стандарті *ECMA*. Ці версії взаємно доповнюють одна одну. *C++ ISO/ANSI* призначений для розроблення високопродуктивних програм, що виконуються вашим комп'ютером як «рідні». При цьому використовується бібліотека *Microsoft Foundation Classes (MFC). C++/CLI* розроблений спеціально для *.Net Framework* із використанням *Windows Forms*.

У посібнику розповідається про те, що являє собою процес програмування на *C++ ISO/ANSI* із застосуванням бібліотеки *MFC*.

При недостатньому об'сязі годин, відведених на практичні заняття, недоцільно їх розпорошувати на вивчення обох технологій. Проте, щоб дати початкове уявлення про програмування на *C++/CLI*, у посібнику

також наведені рекомендації щодо створення консольного застосування для цієї версії *C++.*

Навчання відбувається на прикладах програм, що працюють. При цьому *Microsoft Visual C++ 2010* розглядається як набір інструментів, за допомогою якого можна скласти програму для вирішення конкретного завдання. Матеріал розміщений у порядку зростання складності з урахуванням отриманого досвіду. Тому рекомендується всі роботи виконувати в тій послідовності, в якій вони викладені в навчальному посібнику. При виконанні кожної роботи необхідно в процесі ознайомлення із загальними вказівками ввести в комп'ютер як приклад супровідну програму, запустити її і відлагодити. Потім за контрольними запитаннями перевірити засвоєння матеріалу і повторити написання програми уже без допомоги посібника. У разі успішної перевірки для закріплення матеріалу рекомендується запропонувати власну постановку завдання і скласти програму для її вирішення.

У посібнику можна знайти означення всіх базових понять, що використовуються. Для цього передбачено спеціальний індексний покажчик.

Інтегроване середовище розроблення. Створення проекту консольного додатка Win32

Постановка завдання

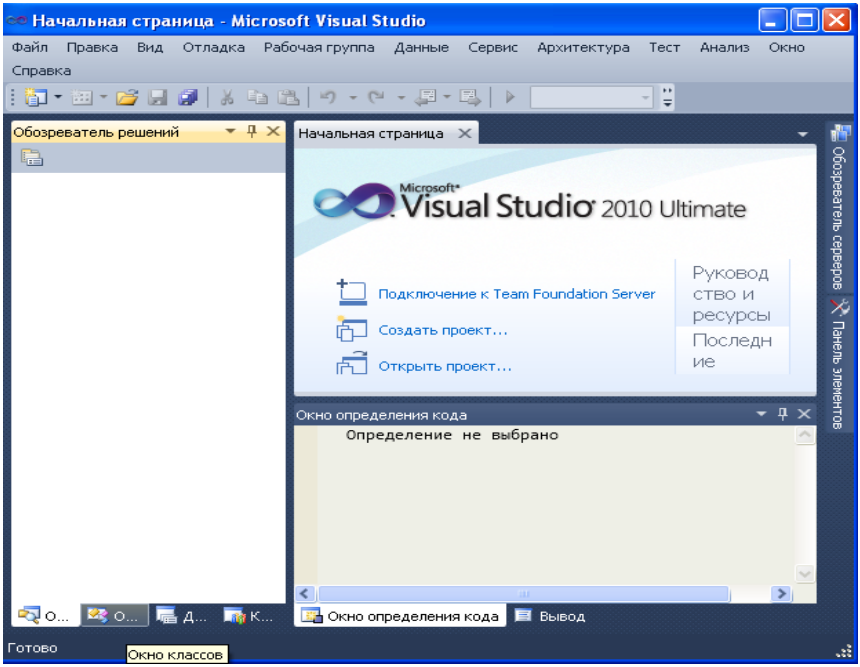
Ознайомитися з інтегрованим середовищем розроблення. Створити проект консольного додатка *Win32*.

Загальні вказівки

Інтегроване середовище розроблення *(Integrated Development Environment (IDE))* – середовище для створення, компіляції, компонування і налагодження програм на *C++.* Компоненти системи:

1. Редактор.
2. Компілятор.
3. Компонувальник.
4. Бібліотеки.

При запуску *VISUAL C++ 2010* відкривається вікно. Воно містить: вікно провідника рішень (*Обозреватель решений*), стартову сторінку (*Начальная страница*), в правій верхній частині вікна знаходиться вікно редактора, у нижній частині вікна – вікно виведення (*Вывод*). Вікно провідника рішень дозволяє вибирати потрібні файли та відображати їх у вікні редактора, а також додавати нові файли до існуючої програми. У цьому вікні чотири вкладки: *Обозреватель решений*, *Окно классов*, *Диспетчер свойств*, *Командный обозреватель*.



Вікно редактора – це місце, де відбуваються введення і модифікація коду та інших компонентів програми. Вікно виведення відображає повідомлення, отримані при компіляції та компонуванні програм.

Опції панелі інструментів

Передбачені для вибору панелей інструменти, які повинні відображатися. Клацнувши правою кнопкою мишки на область панелі інструментів, викликаємо динамічне меню. Встановлюємо потрібні команди.

Панелі інструментів, які стискуються (*dockable*)

Це такі панелі, які за допомогою мишки можна перетягувати в зручне місце всередині вікна. Всі панелі на панелі інструментів можуть стискуватися.

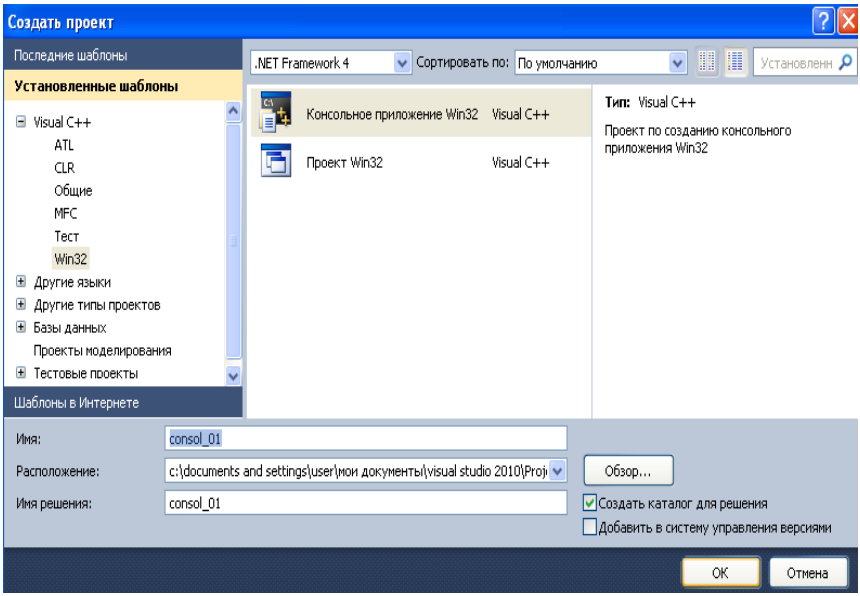
Документація

Бібліотека розробника *Microsoft Development Network (MSDN*) містить довідкову інформацію про всі можливості *Visual C++ 2010*. Виклик – кнопка *F1.*

Створення проекту консольного додатка *Win32*

Створити *Windows*-програму можна декількома способами: вибрати пункт меню *Файл/Создать/Проект*, або натиснути *Ctrl+Shift+N.* І, нарешті, у вікні *Начальная страница* можна клацнути на *Создать проект…*

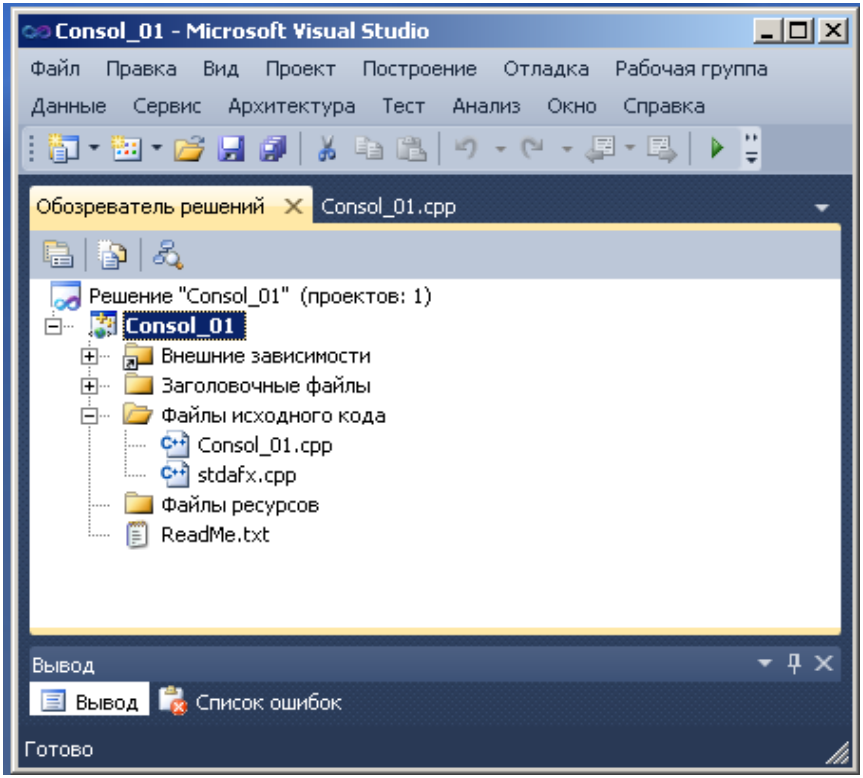
З'явиться вікно *Создать проект*. У лівій частині цього вікна вибрати тип проекту *Win32*. Це також ідентифікує майстер створення додатків, який наповнить проект початковим змістом. У правій частині вікна знаходиться список шаблонів, доступних для вибраного зліва типу проекту. Вибраний шаблон використовується майстром додатка при створенні файлів, з яких складається проект. Вибрати шаблон *Консольное приложение Win32*.



У полі *<Имя>* вводиться ім'я проекту, наприклад *Consol_01*. Натиснути *Enter* або клацнути на кнопці *ОК*. Викликається діалогове вікно майстра *Мастер приложений Win32*.

Клацнути на вкладку *Параметры приложения* в лівій частині, щоб відобразити сторінку настроєнь додатка. Тут можна вибрати опції, які потрібно застосувати до проекту. При створенні порожнього проекту вибирається опція *Пустой проект*. Як правило, вибирається *Консольное приложение.* Клацнути на кнопці *Готово.*

Якщо було вибрано *Консольное приложение*, то за допомогою провідника *Обозреватель решений* можна побачити папку рішення і папку проекту.

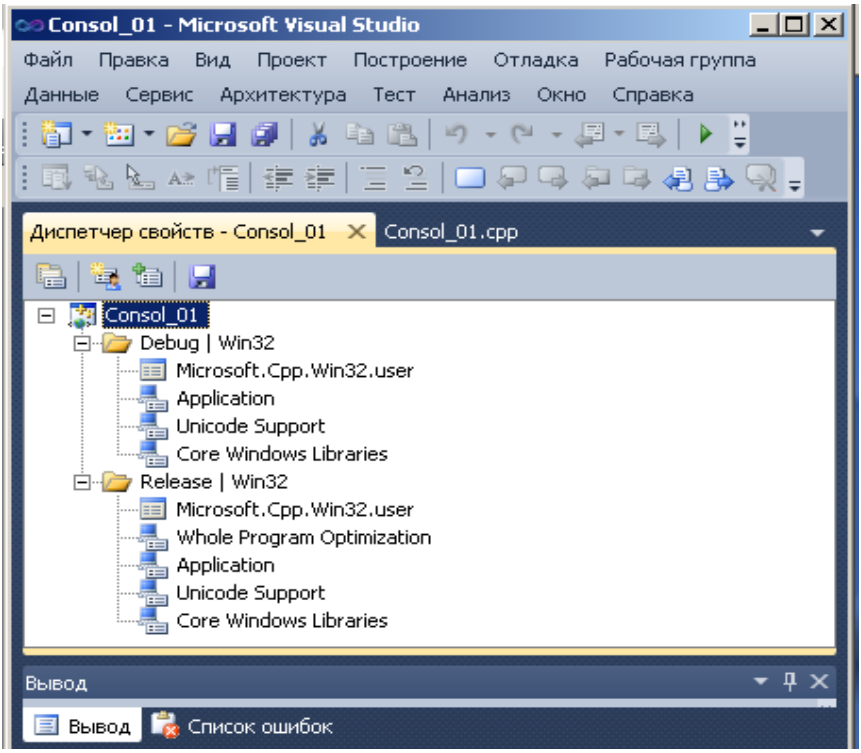


Створений проект буде автоматично відкритий у *Visual C++ 2010*. За допомогою вкладки *Обозреватель решений* відкрити файл *Readme.txt*. Там описаний вміст папки проекту. Щоб звернутися до проекту, необхідно двічі клацнути лівою кнопкою на його імені в полі *Обозреватель решений*. У полі

редактора можна перемикатися між декількома завантаженими файлами, просто клацаючи на відповідну вкладку.

Вкладка *Окно классов* відображає класи, визначені в проекті, а також вміст кожного з класів. Поки що в даному додатку немає жодного класу.

Вкладка *Диспетчер свойств* показує властивості, встановлені для налагоджувальної (*Debug*) і робочої (*Release*) версій проекту.



Вкладка *Ресурсы* відображає діалогові вікна, піктограми, панелі меню та інші ресурси, які використовує програма. Оскільки у даному випадку створюється консольна програма, в ній не використовуються жодні ресурси.

Подібно до більшості елементів *IDE*-середовища *Visual C++ 2010*, *Обозреватель решений* та інші вкладки являють собою контекстно-залежне меню. Воно викликається клацанням правої кнопки мишки на елементах, що відображаються на вкладках, а інколи і на порожньому місці цих вкладок.

Модифікація вихідного коду

Мастер приложений Win32 генерує повний консольний додаток *Win 32*, який можна скомпілювати і запустити. Програма, що спочатку була згенерована, нічого не робить. Внесемо до неї зміни, необхідні для виведення заданого тексту на екран. Якщо файл *Console_01.c*pp не відображається в панелі редактора, потрібно двічі клацнути на його імені в панелі *Обозреватель решений.* Цей файл – головний вихідний файл програми, який був згенерований майстром *Мастер приложений Win32*. Саме у цьому файлі буде розміщена головна функція *_tmain().* Якщо у файлі не відображаються номери рядків, виберіть в головному меню *Сервис/Параметры/Текстовый редактор/C/C++*. У правій панелі виберіть *Показывать/Номера строк*.

Необхідно звернути увагу, що головна функція має символічне ім'я *_tmain( )* і має два формальні параметри. Щоб мати можливість введення з клавіатури і виведення на екран, потрібно додати директиву препроцесора *#include <iostream>.* Тепер у текст головної функції можна вставити оператор для введення рядка символів, наприклад *std::cout<<"I am glad to see you!!!".* Щоб уникати префікса *std* при зверненні до імен стандартних функцій введення/виведення *(cin/cout),* можна застосувати інструкцію u*sing namespace std.* Її слід розмістити до написання заголовка функції *_tmain( ).*

```
// Consol_01.cpp

#include "stdafx.h"
#include <iostream>
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
 cout<<"I am glad to see you!!!";
 return 0;
}
```

Складання рішення

Щоб побудувати рішення, натисніть клавішу *F7* або виберіть пункт меню *Построение/Построить решение*. Ураховуючи, що в даному прикладі до рішення входить лише один проект, можна також

викликати меню *Построение/Построить Console_01.* Результати будуть одні й ті самі.

При внесенні до програми будь-яких змін, поправок необхідно викликати *Построение/ Перестроить решение або Ctrl+Alt+F7*. Можна також перебудувати лише проект *Построение/ Перестроить Console_01.*

У разі, коли необхідно запустити послідовність операцій компіляції, компонування і виконання програми, можна просто одночасно натиснути *Ctrl+F5*.

Налагоджувальна і робоча версії програми

Можна встановити широкий діапазон будь-яких опцій проекту, вибравши пункт меню *Проект/ Свойства: Console_01*. Ці опції визначають, як обробляється код на стадіях компіляції і компонування. Набір опцій, який породжує конкретну версію програми, називається конфігурацією. *Visual C++ 2010* автоматично створює конфігурації для побудови двох версій додатка. Одна з них – налагоджувальна (*Debug*), містить інформацію, що допомагає при налагодженні програми. Інша версія називається робочою (*Release*). У ній немає жодної налагоджувальної інформації, що дозволяє забезпечити максимальну ефективність виконуваного модуля.

Для вибору конфігурації використовується пункт меню *Построение/Диспетчер конфигураций…* З'явиться вікно, в якому в колонці *Активная конфигурация решения* можна вибрати *Debug* або *Release*, після чого клацнути по кнопці *Закрыть*. Як правило, конфігурацію *Release* встановлюють після ретельного налагодження програми в конфігурації *Debug*.

Виконання програм

Після успішної компіляції і компонування програму можна запустити, натиснувши *Ctrl+F5*. Це відповідає виклику за допомогою меню *Отладка/Запуск без отладки.*

Запуск налагоджувального режиму можна зробити з меню *Отладка/Начать отладку* або натиснувши *F5*. При налагоджуванні можна використовувати переривання, а також реалізувати покрокові режими *Шаг с заходом* (клавіша *F11*) і *Шаг с обходом* (клавіша *F10*).

Контрольні запитання

1. Які вікна входять до вікна додатка і для чого вони призначені?
2. Як установлюються опції панелі інструментів і які з них, як правило, вибираються?
3. Як звернутися в бібліотеку розробника *MSDN*?
4. Як створити проект консольного додатка *Win32*?

5. Що таке рішення і як його скласти?

Налагодження проекту консольного додатка Win32

Як правило, середовище розроблення *IDE* містить спеціальні засоби для налагодження програм – *налагоджувачі.* Налагоджувач має інтерфейс користувача для покрокового виконання програми оператор за оператором або функція за функцією із зупинками в *контрольних точках* програми *(breakpoint)* чи при досягненні якоїсь умови та для динамічного відображення значень змінних (*трасування* змінних).

Налагоджувач може використовуватися безпосередньо або запускатись автоматично, якщо під час виконання програми щось відбувається не так, як потрібно. Є можливість виявити місце аварійного переривання програми. Можна розглянути послідовність викликів функцій, що виконувалися в момент переривання (це називається переглядом *стека викликів*), відобразити значення локальних і глобальних змінних. Цієї інформації зазвичай буває достатньо для виявлення помилки. В іншому випадку можна повторно запустити програму в покроковому режимі, щоб виявити, де саме розпочинається неправильна поведінка. Проте навіть найкращий налагоджувач може лише допомогти локалізувати

помилку, а виправити її повинен програміст. Більш детально про це можна прочитати в [1, 2].

Отже, *налагоджувач (debugging)* – це програма, яка контролює виконання вашої програми таким чином, що ви можете покроково виконувати код, рядок за рядком або ж запустити його до певної точки в програмі. У кожній точці програми, де налагоджувач зупиняється, можна переглянути (або змінити) значення змінних, перш ніж продовжити виконання. Можна змінити вихідний код, перекомпілювати, а потім перезапустити програму спочатку. Можна змінити вихідний код під час покрокового виконання програми. При переході до наступного оператора після модифікації коду налагоджувач автоматично перекомпілює програму перед виконанням наступного оператора.

Щоб зрозуміти базові засоби налагодження середовища розроблення *Visual C++ 2010*, використовуйте *налагоджувач* у програмі, яка, як ви впевнені, працює правильно. Переконайтеся, що конфігурація складання прикладу встановлена в *Win32 Debug*, а не у *Win32 Release*. Поточна активна конфігурація складання показана в парі спадних списків панелі інструментів *Стандартная.* Потім клацніть правою кнопкою миші на панелі інструментів. Переконайтеся в наявності прапорця перед *Отладка,* щоб відображалася панель інструментів для налагодження програми. Вона

з'являється автоматично, коли запущений налагоджувач.

Конструкція *Debug* включає додаткову інформацію у виконувану програму під час компіляції, необхідну для налагодження. Вона зберігається у файлі **.pdb* в папці *Debug.* Конфігурація *Relase* виключає таку інформацію.

Запускати налагоджувач можна вибором пункту меню *Отладка/Начать отладку* або клавішею *F5*. Налагоджувач має два головних режими виконання – покрокове виконання коду і виконання до певної точки, зазначеної у вихідному коді. Визначається вона або місцем розташування курсора, або спеціально призначеною точкою зупину, яка називається *точкою переривання (breakpoint)*. У цій точці налагоджувач перериває виконання програми. Це дозволяє переглянути змінні програми і за необхідності змінити їх значення. Щоб установити точку переривання на початку рядка коду, необхідно клацнути кнопкою мишки на сірій вертикальній колонці зліва від номера рядка оператора, на якому потрібно призупинити виконання програми. З'явиться червоний круглий символ, який називається *гліфом (glyph).* Видалити точку переривання можна подвійним клацанням на *гліфі*. Існує можливість специфікувати безліч точок переривання з тим, щоб виконання програми зупинялося в необхідних місцях.

Опція *Начать отладку* виконує програму до першої точки переривання, якщо її встановлено. Далі программу можно виконувати послідовно крок за кроком. Для цього в опції *Отладка* існують команди *Шаг с заходом (F11)* і *Шаг с обходом (F10)*. В обох випадках виконується по одному оператору за один раз. У першому випадку здійснюється захід у кожен вкладений блок коду та в кожну функцію, що викликається. Вибір пункту *Шаг с обходом* виконує оператори по одному за крок. При цьому виконується весь код, використовуваний функціями, які можуть бути викликані всередині оператора, як суцільний потік операцій без зупинок.

Під час налагодження програми необхідно контролювати поточні значення параметрів. Для цього досить підвести курсор до потрібного ідентифікатора. Але зручніше, коли поточні значення постійно відображаються у вікні. З цією метою необхідно ввімкнути *Начать отладку*. Потім вибрати пункт меню *Отладка/Окна/Локальные* або *Отладка/Окна/Видимые*. У першому випадку у вікні з'являться поточні значення усіх локальних змінних. Це не завжди зручно, бо цей список може бути великим. Зручніше вибрати *Окна/Видимые*. У цьому випадку у вікні з'являються значення тільки видимих в даний момент змінних. За необхідності їх значення можна змінити. Для цього потрібно викликати

контекстне меню, вибрати пункт *Изменить значение* і встановити нове значення.

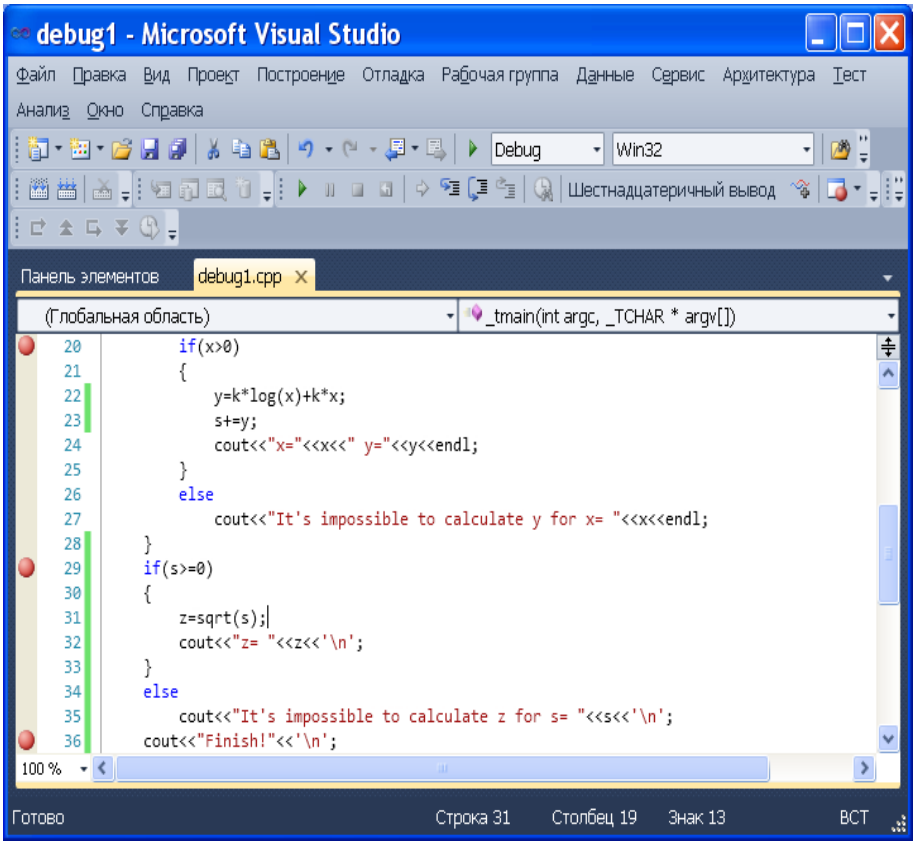
Необхідно звернути увагу на файл **.exe* , який на чорному фоні з'являється при введенні даних. Якщо це вікно залишається відкритим, налагодження програми зупиняється. Вікно **.exe* потрібно згорнути. Якщо необхідно переглянути проміжні результати, це вікно потрібно розгорнути. За необхідності переходу від однієї точки переривання до іншої, потрібно скористатися клавішею <*F5*>.

Розглянемо приклад налагодження програми *debug1.cpp*, в якій виконується табулювання функції $y = k \cdot \ln x + k \cdot x$ із одночасним накопиченням суми результатів, а потім обчислюється корінь квадратний із цієї суми.

```
#include "stdafx.h"
#include<iostream>
#include<math.h>
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
 double x,xn,dx,xk,y,k=2.1,s=0,z;
 cout<<"Enter xn="<<endl;
 cin>>xn;
 cout<<"Enter dx="<<endl;
 cin>>dx;
 cout<<"Enter xk="<<endl;
 cin>>xk;
 for(x=xn;x<=xk;x+=dx)
```

```
 {
  if(x>0)
  {
   y=k*log(x)+k*x;
   s+=y;
   cout<<"x="<<x<<" y="<<y<<endl;
  }
  else
   cout<<"It's impossible to calculate y for
x= "<<x<<endl;
 }
 if(s>=0)
 {
  z=sqrt(s);
  cout<<"z= "<<z<<'\n';
 }
 else
  cout<<"It's impossible to calculate z for
s= "<<s<<'\n';
 cout<<"Finish!"<<'\n';

 return 0;
}
```

На скріншоті наведено фрагмент програмного коду із трьома точками переривання.



Щоб запустити налагоджував, потрібно виконати такі дії:

1. Вибрати пункт меню *Отладка/Начать отладку*.

2. З’явиться вікно *debug1.exe*. Ввести значення *xn, dx, xk*.

3. Програма зупинилася на першій точці переривання. Згорнути вікно *debug1.exe*.

4. Виконати цикл за допомогою кнопки <*F10*>, якщо хочемо перевіряти роботу кожного оператора, або кнопки <*F5*>. При цьому у вікні *Видимые* можна контролювати поточні значення змінних, які в даний момент потрапляють в область видимості.

5. Після завершення циклу програма зупиниться на наступній точці переривання. Можна проконтролювати значення отриманої суми.

6. Натиснувши на <*F5*>, можна відразу перейти до останньої точки переривання. Подальше продовження налагодження приведе до кінця програми.

7. Вибрати пункт меню *Отладка/Остановить отладку*.

Мають місце й інші можливості для налагодження програми. Зокрема, це використання розширених точок переривання, точок трасування, стека викликів функцій. Крім того, існує механізм обробки виключень. Із усіма цими можливостями можна познайомитися самостійно в [1, 2].

Контрольні запитання

1. Звідки беруться помилки в програмах і як із ними боротися?
2. Прийоми налагодження проекту консольного додатка Win32.
3. Базові операції налагодження проекту консольного додатка Win32.

Додавання класів до консольного додатка Win 32

Постановка завдання

Створити проект консольної програми *Win 32*. До нього додати клас «Букет» з елементами даних: назва, кількість квіток у букеті, вартість однієї квітки, вартість букета. Передбачити функції-члени: для введення значень елементів-даних із клавіатури, для розрахунку вартості букета і для виведення значень усіх елементів-даних на екран. У головній функції описати об'єкт класу і реалізувати виклик усіх функцій-членів.

Визначення класу

Клас – це визначення типу даних, що задається користувачем. Він містить елементи даних, які можуть бути як змінними базових типів *C++*, так і інших, визначених користувачем типів. Елементи даних класу можуть бути окремими елементами даних, масивами, покажчиками, масивами покажчиків, об'єктами інших класів. Клас також може містити функції, які оперують об'єктами класу, звертаючись до елементів даних. Таким чином, клас об'єднує визначення елементарних даних, з яких складається об'єкт, і засоби маніпулювання цими даними.

Дані та функції в класі називаються *членами класу*. Члени класу, які є елементами даних, називаються *даними-членами* (або *полями*), а функції, що

належать класу, – *функціями-членами* (або *методами класу*).

Розглянемо приклад класу – клас *man. Визначити* тип даних *man, використовуючи ключове слово class, можно таким чином:*

```
class man
{
 private:
 char* name;
 int day;
 float ras;
 float zar;
 public:
 void getinfo(void);
 void calcul(void);
 void putinfo(char* imja);
};
```

Тут у фігурних дужках визначено чотири *дані-члени* класу і три *функції-члени*. Визначення класу завершується крапкою з комою. Імена всіх членів класу локальні стосовно нього, тому їх можна використовувати ще десь у програмі.

Ключове слово *public* (відкритий) визначає атрибут доступу до членів класу, які йдуть за ним. Визначення даних-членів як *public* означає, що ці члени доступні в будь-якій точці в області видимості об'єкта класу, до якого вони належать. Можна

вказати члени класу як *private* (закритий) або *protected* (захищений).

Оголошують об'єкти класу так само, як і об'єкти базових типів.

`man slesar; //`Оголошення об'єкта *slesar* типу *man*

Загальні вказівки

Нехай створений консольний додаток *Win32* з ім'ям *Consol_01* (див. попередню тему). Ставиться завдання додати до нього клас *man* з елементами даних:

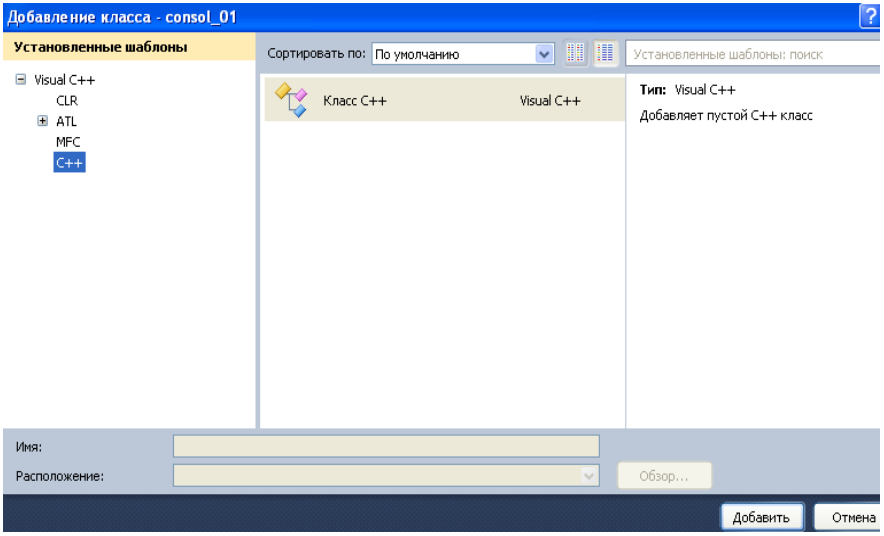
*char*name*; //ім'я
int days; //кількість відпрацьованих днів
float rascenka; //скільки гривень за один день
float zarplata;

Передбачити функції-члени (методи):

void getinfo(); // Введення елементів даних
void calculation(); // Розрахунок зарплати
void putinfo(char);* /* Виведення на екран усіх елементів даних. */

У головній функції описати об'єкт класу *man* і реалізувати виклик методів класу. Для цього необхідно виконати ряд дій:

1. Вибрати пункт меню *Проект/ Добавить класс…*
2. У вікні, що з'явилося, вибрати категорію *C++.* Шаблон (*templates*) автоматично встановиться на *Класс C++.* Натиснути кнопку *Добавить*.



3. З'явиться вікно *Мастер универсальных классов C++ – Consol_01.* У поле *Имя класса* ввести ім'я *man*. Одночасно отримають таке саме ім'я *.hfile* і *.cppfile*. У полі *Базовый класс* нічого не зазначати, оскільки клас *man* не є похідним. У полі *Доступ* залишити *public*. Натиснути кнопку *Готово*.

4. Переконаємося, що у файлі *man.h* з'явився опис класу *man*. Для цього вибрати вкладку *Обозреватель решений*, клацнути на знак «+» навпроти зображення папки *Заголовочные файлы.* Розкриється вміст цієї папки. Двічі клацнути на файлі *man.h.*

У вікні редактора з'явиться такий текст:

```
#pragma once
class man
```

```
{
public:
man(void);
~man(void);
};
```

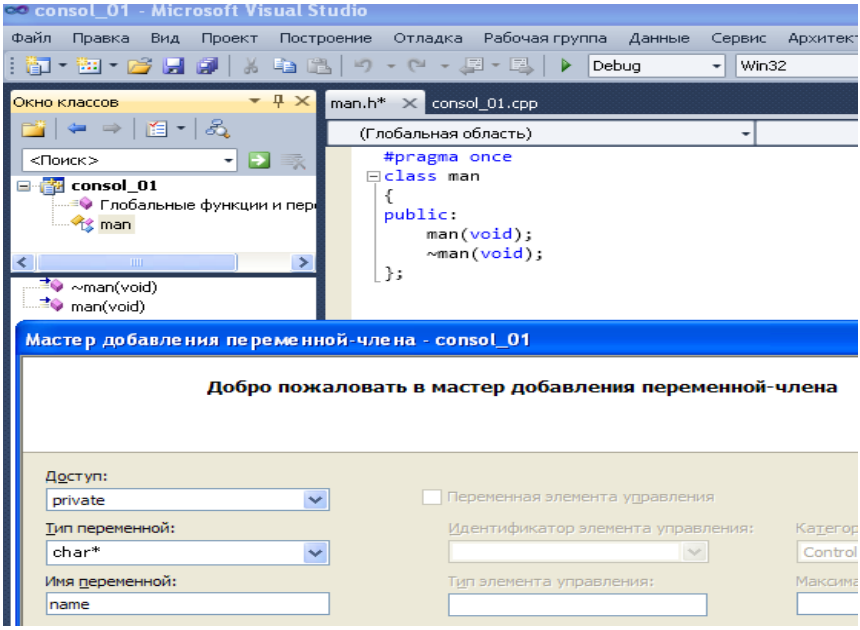
Двічі клацнути на файлі *man.cpp.* У ньому з'явиться текст:

```
#include "stdAfx.h"
#include "man.h"
 man::man(void)   //Пустий конструктор.
{   }
man::~man(void)   //Пустий деструктор.
{   }
```

5. До класу *man* додамо елементи даних. Це можна було б зробити самостійно. Але краще використовувати меню.

Для цього потрібно:

- Виділити вкладку *Окно классов.*
- Клацнути на «+» навпроти *Consol_01*. Розкриється вміст.
- Виділити клас *man* і натиснути праву кнопку мишки. У меню, що розкрилося, вибрати *Добавить/Добавить переменную…* З'явиться вікно *Мастер добавления переменной-члена – Consol_01.*

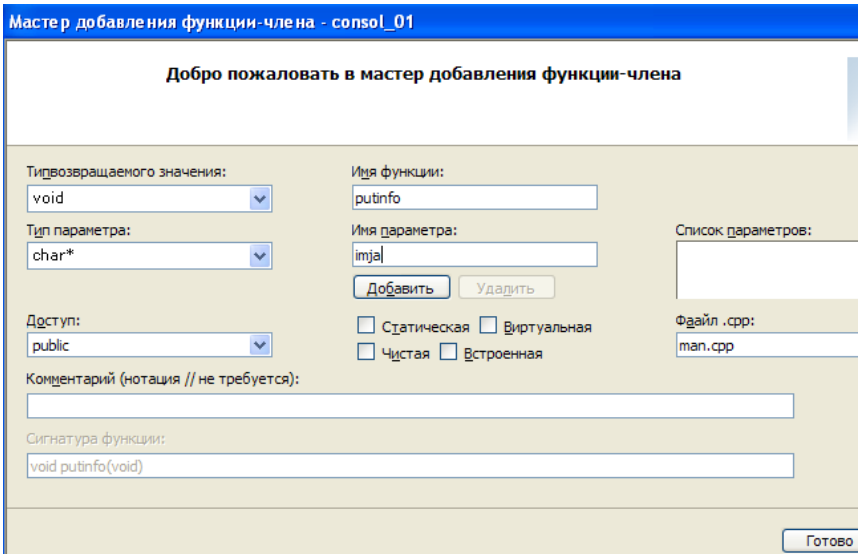


- У вікні *Доступ* вибрати доступ *private*.
- У вікні *Тип переменной* ввести *char**.
- Ім'я *name* ввести у вікні *Имя переменной*.
- Натиснути кнопку *Готово*.

Так само ввести й інші елементи даних.

6. До класу *man* додати функції-члени (методи). Для цього виділити клас man і правою кнопкою миші викликати контекстне меню. Вибрати *Добавить/Добавить функцию*. З'явиться вікно *Мастер добавления функции-члена – Consol_01.* Заповнити вікна: *Тип возвращаемого значения, Имя функции, Тип параметра, Имя параметра.* Якщо

функція без параметрів, у вікні *Тип параметра* вводити *void*.



При цьому вікно *Имя параметра* залишається порожнім. Якщо ж у функції є один і більше параметрів, то потрібно вводити тип першого параметра, його ім'я, далі – натиснути *Добавить*. Після цього вводять тип та ім'я другого параметра і знову натискають *Добавить*. У вікні *Доступ* встановлюють мітку доступу. За замовчуванням це *public*. Натиснути *Готово.*

Так, наприклад, для *void putinfo (char*imja)* у вікно *Тип параметра* вводимо *char**, а в *Имя параметра* – *imja*. Натиснути *Добавить*, а потім – *Готово.*

7. Переходимо на вкладку *Обозреватель решений* і розкриваємо файл *man.h.* Якщо він є у списку вверху вікна редагування, то необхідно просто клацнути на нього. Переконаємося, що у файлі *man.h* є заголовки членів-функцій та елементи даних класу *man* разом із мітками доступу.
8. Розкриваємо файл *man.cpp* та виявляємо, що всі елементи даних у конструкторі ініціалізовані нулями:

```
man::man(void):name(NULL),
 days(0),
 rascenka(0),
 zarplata(0)
{
}
```

9. Тепер необхідно самостійно визначити вміст членів-функцій класу *man*, заповнюючи їх у файлі *man.cpp*. За необхідності можна описати і використовувати у них локальні змінні.
Заповнюємо член-функцію *getinfo().*

```
void man::getinfo()
{
name=new char[12];
cout<<"Enter name"<<endl;
cin>>name;
cout<<"Enter days"<<endl;
cin>>days;
count<<"Enter rascenka"<<endl;
cin>>rascenka;
```

```
}
```

Далі заповнюємо інші методи.

```
void man::calculation()
{
zarplata=days*rascenka;
}
void man::putinfo(char*imja)
{
cout<<”For object”<<imja<<”name=”<<name<<
”days=”<<days<<”rascenka=”<<rascenca<<
”zarplata=”<<zarplata<<endl;
}
```

Для того щоб працювали *cin* і *cout*, необхідно до файла *man.cpp* додати

```
#include<iostream>
using namespace std;
```
// Опис простору імен.

10. Тепер необхідно заповнити головну функцію *_tmain().* Для цього на вкладці *Обозреватель решений* вибрати файл *Consol_01.cpp* і додати до нього

```
#include “man.h”
```

Інакше з'явиться помилка: *«man undeclared identifier»* (*man* – неоголошений ідентифікатор).

У головній функції визначимо об'єкт *slesar*, який працюватиме з функціями-членами:

```
{
 man slesar;
slesar.getinfo();
slesar.calculation();
slesar.putinfo("slesar");
}
```

11. Будуємо і запускаємо програму. Це можна зробити, натиснувши *Ctrl+F5* або за допомогою меню *Построение/Построить решение*, а потім (якщо немає помилок) *Отладка/Запуск без отладки.*

Контрольні запитання

1. Як створити проект консольного додатка Win 32?
2. Як додати до проекту клас?
3. Як ввести в клас елементи даних та члени-функції?
4. Яке розширення у файла, в якому описані клас, елементи даних і члени-функції, і як його розкрити?
5. Яке розширення у файла, в якому визначені члени-функції класу, і як його розкрити?
6. У якому файлі знаходиться головна функція і як його розкрити? Навіщо в цьому файлі директива #include "*.h"?

Введення/виведення в консольному додатку *CLR*. Табуляція функції

Постановка завдання

Створити проект консольного додатка для *C++/CLI*, що здійснює табуляцію функції *y=f(x)* при зміні *x* від *xn* до *xk* із кроком *dx*. Вид функції та числове значення задати самостійно.

Загальні вказівки

Загальномовне середовище виконання *(Common Language Runtime – CLR)* вимагає розширеної версії *C++,* яка має назву *C++/CLI.*

Тут *CLI (Common Language Infrastructure*) – інфраструктура спільної мови. Як і для «рідного» *C++,* що визначається стандартом *ISO/ANSI*, для *C++/ CLI* теж існує можливість створювати консольні проекти.

Для цього необхідно виконати ряд дій:

1. На *Начальная страница* клацнути на *Создать проект*.
2. У вікні *Создать проект* шаблон *CLR*/*Консольное приложение CLR.* У вікні *Имя* ввести ім'я *Inputoutput.* Клацнути на *OK* або на *Enter.*
3. У вікні для редагування з'явиться текст файла *Inputoutput.cpp*. У головній функції *main( )* уже є функція *WriteLine().* Це функція *C++/CLI*, яка визначена в класі *Console* простору імен *System.* Тому в тексті програми після *#include "stdafx.h"* розміщується *using*

namespace System; // використання простору імен *System.*

Клас *Console* являє собою стандартні потоки введення/виведення, що відповідають клавіатурі і командному рядку консольного вікна.

Функція *WriteLine( )* виводить на екран усе те, що знаходиться в дужках і дає перехід на новий рядок. Буква *L* попереду рядка символів вказує на те, що рядок складається із «широких» символів, де кожен займає 2 байти замість одного.

Тут запрограмовано виведення на екран рядка *"*Будьте здорові!*".*

4. Натиснувши *Ctrl+F5*, запускаємо компіляцію і компонування програми. На екрані з'явиться запрограмований рядок.

5. Внесемо зміни до головної функції. Запрограмуємо розрахунок відстані, яку може проїхати автомобіль. Нехай об'єм паливного бака автомобіля *V*=40 л, питома витрата палива *r* =0,095 л/км. Потрібно обчислити відстань *Rasst*, яку проїде машина.

```
int main (array<System::String^> args)
{
double v=40, r =0.095, Rasst;
Rasst=v/r;
Console::Write(L"Rasst=");  // Після L пустий
                            // символ  ставити не можна!
Consol::Writeline(Rasst);
```

```
return 0;
}
```

Зверніть увагу на те, що спочатку виведений рядок символів *"Rasst=",* а потім – саме значення *Rasst*. Можна виведення зробити простішим:

```
Consol::Writeline(L"Rasst={0}", Rasst);
```

або у разі виведення трьох параметрів:

```
Consol::Writeline(L"Rasst={0},v={1},r={2}",
                  Rasst,v,r);
```

У фігурних дужках зазначені номери аргументів у списку: під нульовим номером – *Rasst*, під першим – *v*, під другим – *r.*

Дані, що виводяться, можна форматувати. Припустимо, що вам потрібно вивести значення *Rasst* з плаваючою точкою з двома десятковими розрядами після точки. Це можна зробити таким чином:

```
Consol::Writeline(L"Rasst={0:F2}", Rasst);
```

Буква *F* у специфікаторі формату свідчить про те, що виведення повинне бути у форматі *±ddd.dd*, (де *d* – десяткова цифра), а 2 – кількість розрядів після точки.

У загальному вигляді специфікація формату має такий вигляд*: {n,w:axx}*, де

n – номер аргументу у списку;

w – необов'язкова специфікація ширини поля;

A – односимвольна специфікація формату значення;

xx – необов'язкове одно- або двозначне число, задаючи точність виведення значення.

Таблиця специфікаторів формату

Специфікатор формату	Опис
C або *c*	Виводить значення в грошовому форматі
D або *d*	Виводить ціле в десятковій системі
E або *e*	Виводить значення з плаваючою точкою
F або *f*	Виводить число з фіксованою точкою
G або *g*	Виводить число в найбільш компактній формі як *E* або як *F*
X або *x*	Виводить ціле число у шістнадцятковій системі на верхньому регістрі, якщо *X*, і на нижньому – якщо *x*

Специфікація *w* ширини поля – ціле зі знаком. Якщо $w > 0$, то значення буде вирівняне вправо у полі *w*, а якщо $w < 0$, то – вліво.

Якщо значення займає менше знаків, ніж зазначено у *w*, то введення доповнюється пропусками, а якщо

значення не поміщається в полі завширшки *w*, то специфікація ширини поля ігнорується.

Клавіатурне введення у *C++/CLI*

Можливості введення з клавіатури для консольних програм *.NET Framework* дещо обмежені. Можна прочитати рядок символів як текстовий рядок, використовуючи для цього функцію *Console::Readline().* Якщо потрібно прочитати певний символ, необхідно застосувати функцію *Console::Read().* Можна також визначити натиснуту клавішу за допомогою функції *Console::ReadKey().*

Прочитані символи повинні кудись заноситися. Для цього, як правило, використовуються покажчики. У *C++/CLI* є так звані відстежувані дескриптори, подібні до «рідних» покажчиків *C++*. Але на відміну від покажчика дескриптор зберігає адресу, яка автоматично оновлюється так званим збирачем «сміття», якщо об'єкт, на який він посилається, переміщується під час стискання купи. Друга відмінність: до таких покажчиків не можна застосовувати арифметику адрес. Також не дозволяється зведення дескрипторів.

Відстежувані дескриптори описуються таким чином:

```
тип^ім’я_дескриптора;
```

Нагадаємо, покажчик у *C++ ISO/ANSI* має такий вигляд:

```
тип *ім’я_покажчика;
```

Дуже важливо пам'ятати, що у *C++/CLI* пам'ять, відстежувана дескриптором, виділяється, оновлюється і звільняється автоматично. Для занесення інформації при зчитуванні рядків або окремих символів зручно застосовувати відстежуваний дескриптор типу *String*.

```
String^ ім’я _дескриптора;
```

Можна поєднати опис дескриптора з читанням рядка або символа. Наприклад:

```
String ^ line= Console::ReadLine();
```

Якщо потрібно використовувати дескриптор *line* послідовно кілька разів, необхідно окремо його описати, а потім послідовно в нього заносити результати роботи функцій *Console::ReadLine()* або *Console::Read()*. Наприклад:

```
String^str;
str=Console::ReadLine();
…….
str= Console::Read();
```

……. . . . .

Саме так використовується відстежуваний дескриптор у нижче наведеній програмі табуляції функції.

За необхідності введення числових значень спочатку зчитується рядок символів, а потім він перетворюється в число необхідного типу. Для цього можна використовувати функцію *Parse*. Наприклад, для змінної *x* типу *double* потрібно ввести з клавіатури числове значення. Нижче наводиться фрагмент програми:

```
String^str= Console::ReadLine();
x=Double::Parse(str);
Console::WriteLine(L"x={0}",x);
```

Нехай із клавіатури введено послідовність символів *12.35*. Вона заноситься в *str*, а потім перетворюється в число *12.35*, яке буде виведене на екран.

Приклад програми табуляції функції. Потрібно обчислити $y = sin(x)*e^{-x}$ для випадку, коли *x* змінюється від *xn* до *xk* з кроком *dx*. Необхідно створити консольну програму для *C++/CLI* з ім'ям *tabul*.

```
#include <stdafx.h>
#include <math.h>
using namespase System;
```

```
int main(array<System::String^>^args)
{
   double x,y,xn,xk,dx;
   String^str;
   Console::Write(L"Enter xn=");
   str= Console::ReadLine();
    xn=Double::Parse(str);
   Console::Write(L"Enter dx=");
   str= Console::ReadLine();
    dx=Double::Parse(str);
   Console::Write(L"Enter xk=");
   str= Console::ReadLine();
    xk=Double::Parse(str);
for(x=xn; x<=xk; x+=dx)
{
  y=sin(x)*exp(-x);
  Console::WriteLine(L"x={0},y={1}",x,y);
  Console::WriteLine(L"By-By!";
  return 0;
}
}
```

Потім запустити програму, натиснувши *Ctrl+F5*.

При введенні чисел може виникнути випадок, коли дробова частина відділяється від цілої не крапкою, а комою. Щоб це змінити, необхідно виконати такі дії: *Пуск/Настроєння/Панель керування/Дата, час, мова, і регіональні стандарти/Зміна формату відображення чисел, дати і часу.* Вибрати у вікні *Настроєння/Розділювач цілої і дробової частин.* А можна просто під час уведення відокремлювати дробову частину комою.

Контрольні запитання

1. Що таке *CLR і CLI* ?
2. Що являє собою клас *Console?*
3. Яка система імен використовується для консольних функцій введення/виведення символьних рядків і як вона зазначається?
4. Як здійснити введення числових значень змінних?
5. Як здійснити виведення числових значень змінних?
6. Що таке відстежуваний дескриптор, чим він відрізняється від покажчика і як описується?

Створення і виконання *Windows*-програм із застосуванням бібліотеки *Microsoft Foundation Classes*

Постановка завдання

Створити одно- і багатодокументні *Windows*-програми на *C++ ISO/ANSI* з використанням бібліотеки *Microsoft Foundation Classes (MFC).* Обидві програми повинні бути текстовими редакторами.

Бібліотека *Microsoft Foundation Classes*

Бібліотека *MFC* – це набір визначених класів, на яких побудоване програмування для Windows у середовищі розроблення *Visual C++ 2010*. Ці класи

являють собою об'єктно-орієнтований підхід до програмування *Windows*, що інкапсулює інтерфейс *API Windows.* Бібліотека *MFC* не слідує строго об'єктно-орієнтованим принципам у частині інкапсуляції та приховування даних головним чином тому, що більша частина коду бібліотеки була написана до того, як ці принципи були встановлені.

Процес написання програми *Windows* включає створення і використання об'єктів бібліотеки *MFC* або об'єктів класів, успадкованих від них. В основному ви наслідуєте власні класи від класів бібліотеки *MFC* з істотною допомогою з боку спеціалізованих інструментів середовища розроблення *Visual C++ 2010*, що спрощує завдання. Об'єкти цих заснованих на класах бібліотеки *MFC* типів класів включають функції-члени для взаємодії з *Windows*, обробки повідомлень *Windows*, а також відправлення повідомлень один одному. Ці похідні класи успадковують усі члени своїх базових класів. Успадковані функції виконують практично всю чорнову роботу, що забезпечує роботу програми *Windows*. Усе, що необхідно зробити, – додати дані і функції-члени для настроювання поведінки класів і забезпечення специфічної для додатка функціональності, яка вимагається від вашої програми.

Концепція "документ-представлення" в бібліотеці *MFC*

Структура програми *MFC* містить дві орієнтовані на додаток сутності: *документ (document*) і *представлення (view*). Документ – це ім'я, присвоєне колекції даних вашого додатка, з якими взаємодіє користувач. Це можуть бути дані гри, геометрична модель, текстовий файл і т. ін. Термін документ – це просто прийняте позначення даних програми, що розглядаються як єдине ціле. Документ у програмі визначається як об'єкт класу документа. Клас документа походить від класу бібліотеки *MFC* з ім’ям *CDocument*, в який ви додаєте власні змінні-члени для зберігання елементів програми, а також функції-члени, що підтримують їх обробку. Ваш додаток не обмежений єдиним типом документа; можна визначити декілька класів документів.

Користувач вирішує, з якою кількістю документів буде одночасно працювати програма – тільки з одним чи з кількома. Однодокументний інтерфейс *SDI (Single Document Interface)* підтримується бібліотекою *MFC* для програм, яким потрібно відкривати документи по одному за один раз. Програма, що використовує такий інтерфейс, називається *додатком SDI.*

Для програм, які повинні відкривати по декілька документів одночасно, можна застосовувати багатодокументний інтерфейс *MDI (Multiple Document*

Interface). В інтерфейсі *MDI* існує також можливість організувати одночасну обробку документів різного типу, причому кожен може відображатися у власному вікні.

Представлення завжди належить до певного об'єкта документа. Документ у програмі містить набір даних програми, а *представлення* – це об'єкт, що пропонує механізм відображення даних, які зберігаються в документі. Він визначає, як дані відображаються у вікні і як користувач може взаємодіяти з ними. Користувач визначає власний клас *представлення*, наслідуючи його від класу бібліотеки *MFC* з ім'ям *CView*. Вікно, в якому розміщене *представлення*, називається *обрамляючим вікном (frame window).*

Бібліотека *MFC* являє собою механізм інтеграції документа з його *представленнями* і кожного обрамляючого вікна з поточним активним *представленням*. Об'єкт документа автоматично підтримує список покажчиків на пов'язані з ним *представлення*. А об'єкт *представлення* має змінну-член, що містить покажчик на документ, з яким він пов'язаний. Координація між документом, *представленням* і обрамляючим вікном установлюється об'єктами іншого класу бібліотеки *MFC*, які називаються *шаблонами документа (document template)*.

Створення *SDI* (*однодокументних*)-програм

Як уже відомо з попередніх робіт, створити W*indows*-програму можна декількома способами. Можна вибрати пункт меню *Файл/Создать/Проект,* можна натиснути *Ctrl + Shift + N*. І, нарешті, у вікні *Начальная страница* можна клацнути на *Создать проект…* З'явиться вікно *Создать проект*. У лівій частині вибрати шаблон проекту *MFC*. У правій частині вікна вибрати *Приложение MFC*. У вікні *Имя* зазначити ім'я проекту, наприклад *Texteditor,* і натиснути *OK* або клавішу *Enter*. З'явиться вікно *Мастер приложений MFC – Texteditor.* Пропонується послідовність опцій. Їх вибір дозволяє вказати, які засоби необхідно внести в нову програму.

1. У правій частині *Тип приложения* необхідно вибрати *Один документ*. Потрібно зняти прапорець *Использовать библиотеки с поддержкой Юникода*, що встановлений за замовчуванням. Якщо залишити його установленим, програма під час роботи чекатиме введення в кодах, і у файлах зберігатимуться символи. Це зробить неможливим їх читання для програм, які чекають текст *ASCII.*

Вибрати *Стиль проекта/Стандарт MFC.* Клацнути на *Далее.*

2. У діалоговому вікні *Свойства шаблона документов* можна ввести розширення файлів, які створює ваша програма. Для даного прикладу підійде розширення *.txt*. У цьому вікні також можна ввести

Имя фильтра, що буде ім'ям фільтра, який з'являтиметься у вікнах *Открыть, Сохранить* для фільтрації файлів. Таким чином, будуть відображатися лише файли із зазначеним розширенням.

3. Якщо у лівій частині вікна вибрати *Свойства интерфейса пользователя*, ви отримаєте подальший набір опцій. Залишимо їх значення, встановлені за замовчуванням.

4. У діалоговому вікні *Созданные классы* ви побачите список класів, які *Мастер приложений MFC* створив в коді програми. Можна виділити будь-який клас у списку, клацнувши на його ім’я. У полях, що знаходяться нижче, будуть відображені ім'я класу, ім'я заголовного файла, в якому знаходиться його визначення, ім'я базового класу, а також ім'я файла, що містить реалізацію функцій-членів класу. Визначення класу завжди міститься у файлі **.h*, а вихідний код функцій-членів – у файлі **.cpp*.

У даному прикладі для класу *CTexteditorView* з'являється список можливих базових класів. За замовчуванням базовим є *CView.* Для того щоб можна було редагувати текст як базовий, вибираємо зі списку, що відкривається, клас *CEditView* і клацаємо на *Готово*. Вибір опцій закінчений.

5. Для запуску компіляції, компонування і виконання програми натискаємо *Ctrl + F5*. Відбуваються компіляція і складання проекту.

За відсутності помилок відразу ж відбувається запуск. З'являється вікно *Без названия – Texteditor*. У ньому можна ввести текст на будь-якій мові. Цей текст можна виділити і включивши правою кнопкою миші контекстне меню, можна копіювати, вирізати і так далі.

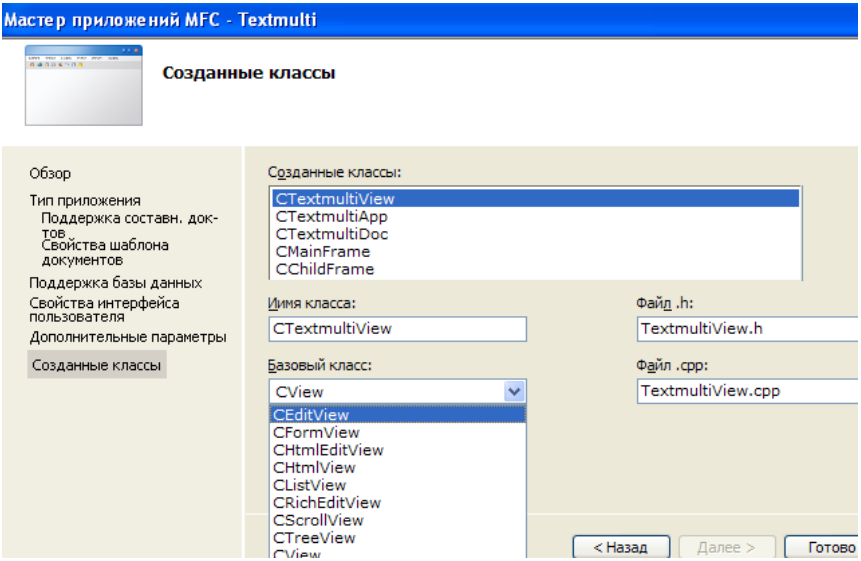
Вибравши меню *Файл*, можна набраний текст зберегти під новим ім'ям (*Сохранить как…*) або під тим, що вже є (*Сохранить*). Можна відкривати файл (*Открыть*), створювати новий (*Создать*), друкувати і так далі. І нарешті, можна припинити роботу програми, вибравши *Файл/Выход.*

Таким чином, створено готовий текстовий редактор без додавання власного коду.

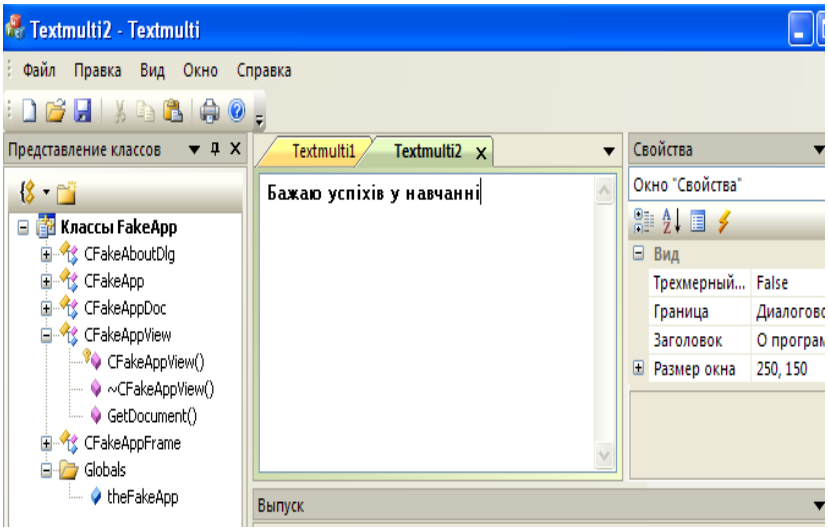
Створення *MDI* (багатодокументних)-програм

Створимо текстовий редактор, який може працювати з декількома вікнами. Дамо йому ім'я *Textmulti.*

Для створення багатодокументного текстового редактора в правій частині в *Тип приложения* потрібно залишити опцію *Несколько документов*. Усе інше – таке саме, як і при створенні однодокументного (*SDI*) текстового редактора. Пам'ятаємо, для того щоб можна було редагувати текст як базовий, у діалоговому вікні *Созданные классы* вибираємо зі списку, що відкривається, клас *CEditView* і клацаємо на *Готово*.



У результаті запуску програми з'являється вікно *Textmulti1 – Textmulti*, а в ньому – вікно *Textmulti1*. Це реалізований багатодокументний режим, коли всередині одного вікна можна створювати вкладені вікна. Головне вікно має меню і панель інструментів, що працюють. Наприклад, можна вибрати *Файл/Создать*, у результаті з'явиться вікно *Textmulti2.*



Примітки:

1. Якщо в діалоговому вікні *Свойства шаблона* не ввести розширення файлів, які створює ваша програма (наприклад, розширення **.txt),* то створювані текстовим редактором файли заноситимуться в папки без розширення. Немає жодних розширень за замовчуванням!
2. Лише текст латиною можна безпосередньо прочитати з файла, натиснувши на клавішу *F3.* Кирилицею текст відображається за допомогою псевдографіки. У програмі текстового редактора всі файли відкриваються і читаються нормально будь-якою мовою.

Контрольні запитання

1. Що являє собою бібліотека *Microsoft Foundation Classes* ?
2. Концепція «документ-представлення» в бібліотеці *MFC.*
3. Як створити однодокументний текстовий редактор?
4. Як створити багатодокументний текстовий редактор?
5. Як задаються розширення для файлів, що створюються текстовим редактором?
6. На якому етапі і як проект набуває функцій текстового редактора?
7. Якого типу проект і який шаблон потрібно вибрати при створенні *MFC*-програми?

Діалогове вікно як головне

Постановка завдання

Скласти програму, при запуску якої повинне з'явитися діалогове вікно. У ньому, крім кнопок *Ok* і *Cancel*, повинні бути стартова кнопка з написом «Пуск» і текстове вікно. При натисканні на стартову кнопку в текстовому вікні повинен з'явитися заданий рядок символів, наприклад «Будьте здорові!».

Поняття діалогових вікон

Діалогові вікна та елементи керування – це основні інструменти взаємодії користувача з операційною системою *Windows*. Для створення і відображення діалогового вікна на базі бібліотеки *MFC* вимагається: фізична поява діалогового вікна, яке визначено у файлі ресурсу, та об'єкт класу діалогового вікна, що використовується для управління операціями діалогового вікна і його елементами керування. Бібліотека *MFC* надає клас *CDialog* після того, як визначили ресурс діалогового вікна.

Що таке елементи керування

В операційній системі *Windows* доступна безліч різних елементів керування, і в більшості випадків система забезпечує високий ступінь гнучкості щодо їх вигляду і поведінки. Більшість цих елементів належить до однієї із шести категорій:

1. *Статичні елементи керування* – використовуються для відображення заголовків або описової інформації.
2. *Кнопкові елементи керування* – надають механізм уведення "одним клацанням". Існує три різновиди кнопкових елементів керування: *прості кнопки*, *кнопки-перемикачі*, з групи яких у кожен окремий момент часу може бути обрана лише одна, і *кнопки-прапорці*, які можуть знаходитися в обраному

стані незалежно від сусідніх прапорців у тій самій групі.

3. *Смуги прокрутки* – зазвичай використовуються для прокрутки тексту або графічних зображень, горизонтально або вертикально, в іншому елементі керування.

4. *Вікна списків* – надають списки, в яких одночасно можна виділити один або кілька варіантів.

5. *Поля введення* – для введення тексту або дозволяють редагувати відображуваний текст.

6. *Списки, що розкриваються,* надають можливість працювати зі списками, в яких можна або вибирати один варіант, або ввести текст самостійно.

Елемент керування може бути чи не бути пов'язаний з об'єктом класу. Оскільки елементи управління є вікнами, всі вони успадковані від класу *CWnd*.

Загальні вказівки

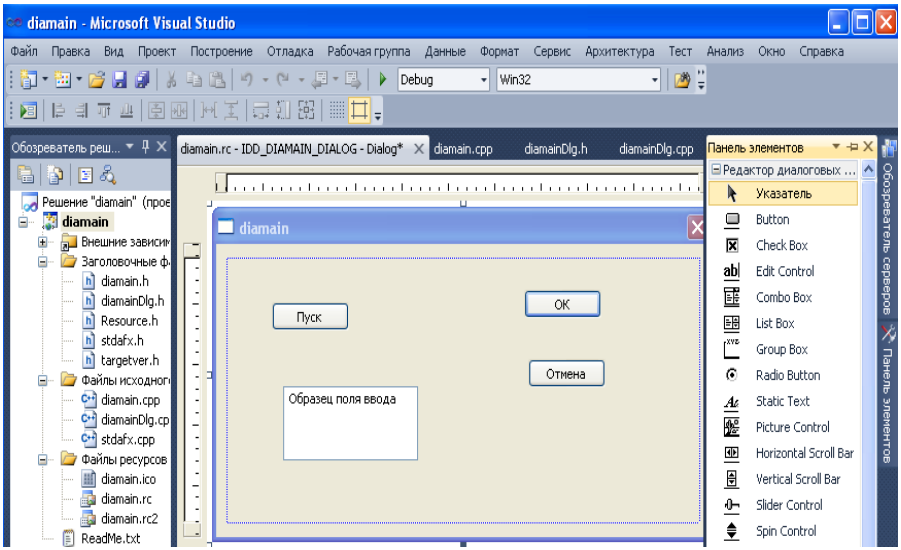
Інколи програма містить багато керувальних елементів, з якими користувач повинен працювати прямо з головного вікна. У цьому випадку зручно призначити головним вікном програми діалогове вікно. Для цього необхідно виконати ряд дій:

1. *Создать проект / MFC / Приложение MFC, Имя – diamain*.

2. У вікні *Тип приложения* вибрати *На основе диалоговых окон* і зняти прапорець *Использовать*

библиотеки с поддержкой Юникода. Клацнути на *Готово.* З'явиться діалогове вікно з кнопками *Ok* і *Отмена.*

3. Розмістити в діалоговому вікні кнопку B*utton1*. Змінити її напис на «Пуск». Ім'я кнопки за замовчуванням *IDC_BUTTON1* (див. *Свойства,* рядок *ID*).

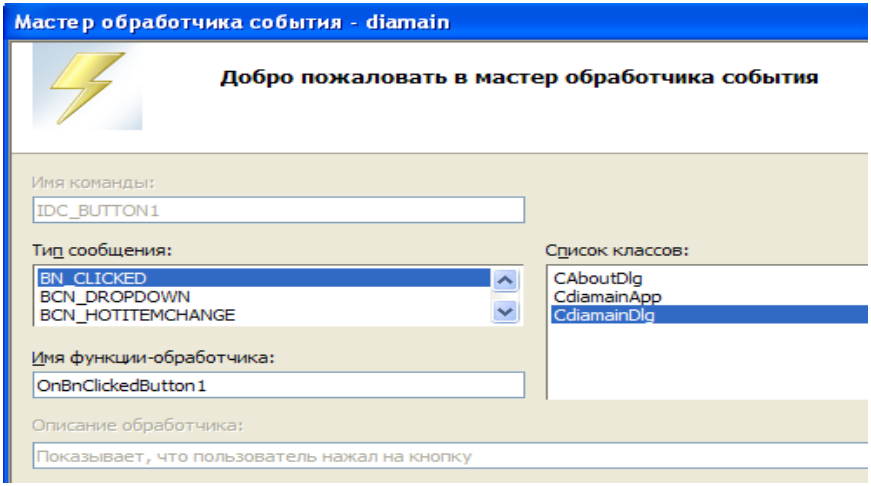


4. Розмістити в діалоговому вікні текстове вікно *(ab| Edit Control)*. Ім'я за замовчуванням *IDC_EDIT1.*

5. Для класу *CdiamainDlg* створити елемент даних, який буде пов’язаний з текстовим вікном *IDC_EDIT1*. Для цього потрібно відкрити *Окно ресурсов diamain / Dialog / IDD_DIAMAIN_DIALOG* виділити текстове вікно, викликати контекстне меню і вибрати в ньому

Добавить переменную… . У вікні, що з'явилося, ввести елемент даних *m_edit1* типу *CEdit* з доступом *private* .

6. Створити обробник (його заготовку) для кнопки «Пуск». Для цього її потрібно виділити, викликати контекстне меню і вибрати *Добавить обработчик событий*. У вікні, що з'явилося, тип повідомлення вибрати *BN_CLICKED*, а у списку класів – *CdiamainDlg*. Ім’я обробника встановлюється за замовчуванням OnBnClickedButton1. Клацніть на кнопку *Добавить/Править.*



Обробник можна створити більш простим способом. Для цього необхідно двічі клацнути на вибраній кнопці «Пуск». Але так можна вчинити, якщо немає необхідності вибирати клас зі списку, що

з'являється (у даному випадку клас один – *CdiamainDlg*), і не потрібно самому встановлювати ім’я обробника.

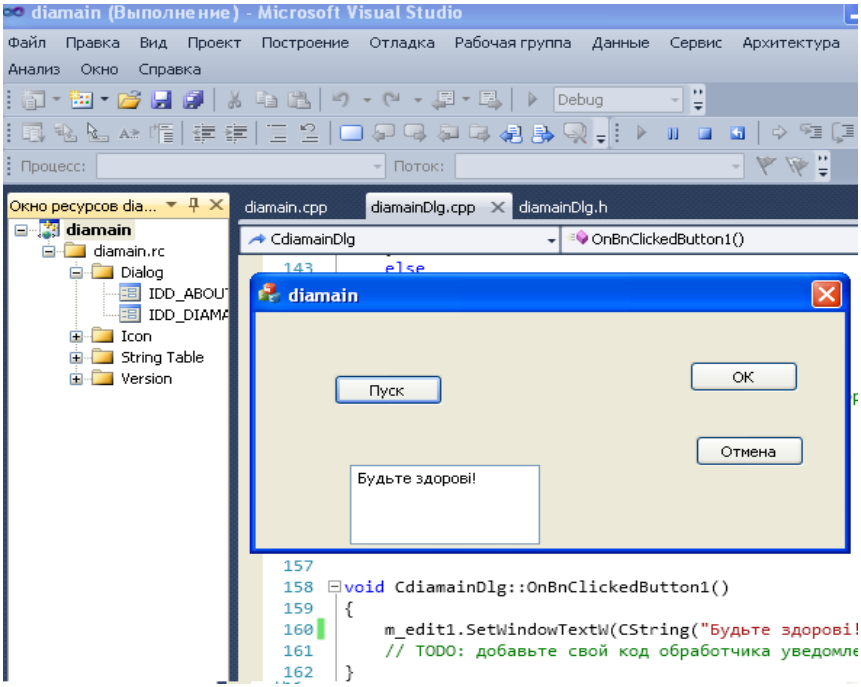
7. В отриману таким чином заготовку обробника необхідно додати оператор занесення рядка символів у текстове вікно, пов’язане з елементом даних m*_edit1 класу CdiamainDlg*. Прототип функції, що дозволяє це зробити, має вигляд:

```
void CWnd::SetWindowTextA(LPCTSTR lpszString);
```

Нижче наводиться текст обробника.

```
void CdiamainDlg::OnBnClickedButton1()
{
m_edit1.SetWindowTextA(Cstring("Будьте здорові"));
 /*TODO: добавьте свой код обработки
 уведомлений */
}
```

Програма готова. Після її запуску та натиснення кнопки «Пуск» у текстовому вікні з’явиться «Будьте здорові!».



Необхідно ще раз відмітити, що в режимі, коли діалогове вікно є головним *(На основе диалоговых окон),* немає класів «*Документ*» і «*Вид*». Звичайно, немає і функції *OnDraw(),* тому введення в діалоговому вікні запрограмоване безпосередньо в обробнику *OnBnClickedButton1().*

Контрольні запитання

1. Поняття діалогових вікон.
2. Що таке елементи керування?
3. Як зробити діалогове вікно головним?

4. Як створити діалогове вікно і встановити необхідні керувальні елементи?
5. Як створити обробник натиснення стартової кнопки?
6. Назвати ідентифікатори за замовчуванням для встановлених керувальних елементів.
7. Як додати до класу *CdialogtabulDlg* елементи даних, що відповідають керувальним елементам?
8. Як здійснюється виведення в обробнику *OnBnClickedButton1()?*

Табуляція функції, коли діалогове вікно є головним

Постановка завдання

Скласти програму табуляції функції *y = f(x)* при зміні *x* від *xmin* до *xmax* з кроком *dx.* При запуску програми повинне з'явитися діалогове вікно. У ньому, крім кнопок *Ok* і *Оттена*, мають бути стартова кнопка з написом «*Пуск*» і текстове вікно, в яке можна виводити результати у вигляді послідовності рядків (список). Крім того, мають бути три вікна відповідно для введення в них числових значень *xmin*, *dx*, *xmax*. При введенні цих даних і подальшому натисненні на кнопку «*Пуск*» у клієнтському вікні повинні з'явитися результати роботи програми. Функцію *y = f(x)* задати самостійно.

Загальні вказівки

Ця програма містить багато елементів, з якими користувач повинен працювати прямо з головного вікна. У цьому разі зручно призначити головним вікном програми діалогове вікно.

Для цього необхідно виконати ряд дій:

1. Создать проект / MFC / Приложение MFC, Имя – Dialogtabul.

2. У вікні *Тип приложения* вибрати *На основе диалоговых окон* і зняти прапорець *Использовать библиотеки с поддержкой Юникода.* Клацнути на *Готово.* З'явиться діалогове вікно з кнопками *Ok* і *Отмена.*

3. Стерти текст по центру *«//TODO…».* Для цього виділити його і натискувати кнопку *"Delete".*

4. Розмістити в діалоговому вікні кнопку *Button1*. Змінити її напис на «Пуск». Ім'я кнопки за замовчуванням *IDC_Button1* (див. *Свойства/ИД).*

Розмістити в діалоговому вікні керувальний елемент *ab| Edit Control*. Ім'я за замовчуванням *IDC_EDIT1*. Це вікно для занесення значення *xmin.* Також розмістити по вертикалі один під одним у діалоговому вікні керуючі елементи *ab| Edit Control* з іменами за замовчуванням *IDC_ EDIT2 і IDC_ EDIT3.* Це вікна для занесення значень *dx* і *xmax*. Щоб їх вирівняти по лівому краю, потрібно виділити верхнє вікно (для *xmin*), натиснути на *Ctrl* і, не відпускаючи її, клацнути

по останніх двох вікнах (для *dx* і *xmax*). Потім викликати контекстне меню, вибрати в ньому *Выровнять левые границы*.

5. Праворуч від кожного із цих вікон розмістити керувальні елементи *Aa Static Text*. Для кожного з них у рядку *Подпись* вікна *Свойства* внести відповідно *xmin*, *dx*, *xmax*, щоб позначити, в які текстові вікна вносити значення цих змінних.

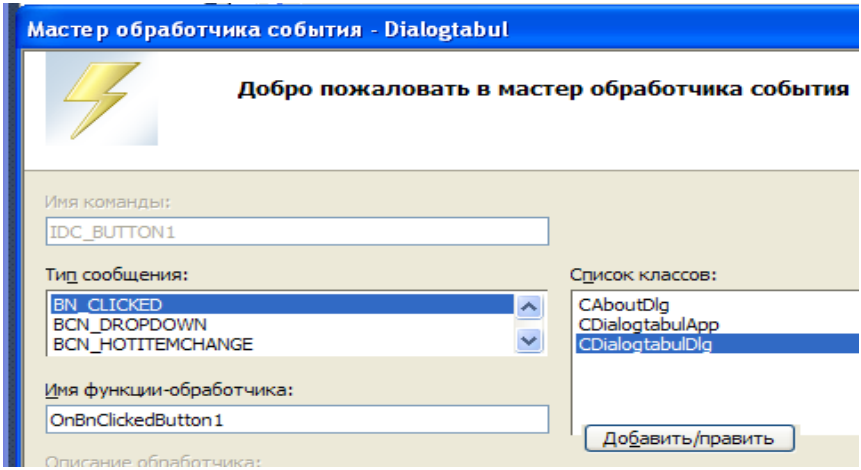
6. Розмістити в діалоговому вікні керувальний елемент *List Box*. Ім'я за замовчуванням – *IDC_LIST1*. Це вікно для занесення результатів.

7. У клас *Cdialogtabuldlg* додати елементи даних, відповідні кожному з керувальних елементів, крім кнопки *Button1 («Пуск»*). Це має вигляд як додавання змінних відповідних типів (об'єктів класів). Для цього виділити вікно, призначене для введення *xmin* (ідентифікатор *IDC_EDIT1*). Викликати контекстне меню, вибрати в ньому *Добавить переменную* і клацнути лівою кнопкою мишки. У вікні, що з'явилося, майстер автоматично встановлює тип змінної *CEdit*. Залишається вибрати доступ, наприклад, *public*, та ім’я змінної (елемента даних), наприклад, *m_edit1*.

8. Те саме виконати для вікон з ідентифікаторами *IDC_EDIT2* і *IDC_EDIT3*, призначеними для введення *dx* і *xmax*. Елементи даних відповідно назвати m_edit2 і m_edit3. Так само потрібно виконати і для керувального елемента *List Box* (ідентифікатор

IDC_LIST1). Для нього майстер установить тип *CListBox*. Нехай ім'я змінної буде *m_list*.

9. Для керувального елемента кнопки *Button1* («*Пуск*») потрібний не елемент даних, а обробник події (натиснення кнопки). Тобто потрібно додати метод класу *CdialogtabulDlg*. Для цього потрібно виділити кнопку, викликати контекстне меню, вибрати в ньому *Добавить обработчик событий*. У вікні, що з'явилося, автоматично майстром вибраний тип повідомлення – *BN_CLICKED* та у списку класів *Class list – CdialogtabulDlg.* Залишається натиснути *Добавить/Править* і заготовка обробника *OnBnClickedButton1()* з'являється в класі *CdialogtabulDlg.*



10. Додати в обробник опис локальних змінних і послідовність операторів. Нижче наводиться приклад

фрагмента файла *dialogtabulDlg.cpp,* у який входить обробник.

```
#include<math.h>
void CdialogtabulDlg::OnBnClickedButton1()
{
 double x,y,xn,dx,xk,b=2.1;
 CString s;
 m_edit1.GetWindowTextA(s);
 xn=atof(s);
 m_edit2.GetWindowTextA(s);
 dx=atof(s);
 m_edit3.GetWindowTextA(s);
 xk=atof(s);
 for(x=xn;x<=xk;x+=dx)
  if(x>=0)
 {
  y=sqrt(b*x);
  s.Format(_T("x=%lf y=%lf"),x,y);
  m_list.AddString(s);
 }
  else
  {
   s.Format(_T("No rezult for x=%lf"),x);
   m_list.AddString(s);
  }
 /*TODO:  добавьте  свой  код  обработки
уведомлений */
}
```

Спочатку в файл додана директива *#include<math.h>,* щоб можна було використовувати стандартні математичні функції.

Для введення значень *xn, dx, xk* використовується функція, прототип якої має вигляд

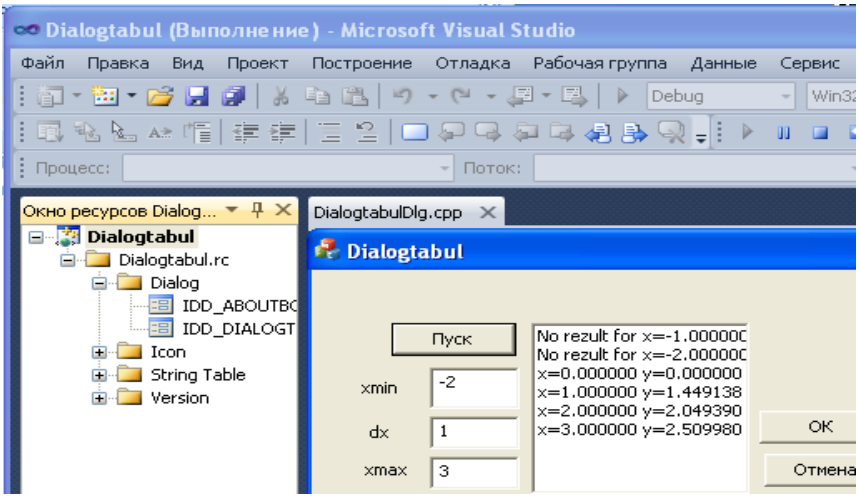
```
void CWnd::GetWindowTextW(CString&RString) const
```

Клас *CEdit* – похідний від *CWnd*. Тому об'єкти m_e*dit1, m_edit2 і m_edit3* можуть використовувати цей метод. Аргументом є посилання на *CString.* Конкретно використовується локальна змінна *CString s*. Їй привласнюється рядок символів, що представляють значення *xn, dx, xk.* Потім за допомогою функції *_wtof()* рядок символів перетвориться в число дійсного типу.

Для виведення рядка символів використовується функція *AddString()* класу *CListBox,* що викликається об'єктом *m_list.* Її аргументом є змінна *s*. Рядок символів, що виводяться, заздалегідь формується за допомогою функції *Format(*). Ця функція нагадує *printf().* Вона перетворить дані інших типів у текст. У даному прикладі рядок символів, що виводиться, заздалегідь обробляється функцією *_T().* Вона перетворить символ або рядок в їх аналог в унікоді *(Unicode).* Якщо при виборі опцій у вікні, де вибирався режим *На основе диалоговых окон*, відключити прапорець *“Использовать библиотеки с поддержкой Юникода”,* то *_T()* не потрібно, і можна тоді записати безпосередньо

```
s.Format(“x=%lf y=%lf”,x,y);
```

Також тоді замість *GetWindowTextW() буде GetWindowTextA, а замість _wtof() необхідно застосовувати функцію atof().* Програма готова. Після її запуску у відповідні вікна ввести значення *xn, dx, xk* і клацнути по кнопці «*Пуск*».



Контрольні запитання

1. Як зробити діалогове вікно головним?
2. Як створити діалогове вікно і встановити необхідні керувальні елементи?
3. Як створити обробник натиснення стартової кнопки ?
4. Назвати ідентифікатори за замовчуванням для встановлених керувальних елементів.

5. Як додати до класу *CdialogtabulDlg* елементи даних, що відповідають керувальним елементам?
6. Навіщо потрібно створювати локальну змінну типу Cstring в обробнику *OnBnClickedButton1*()?
7. Як здійснюються введення і виведення в обробнику *OnBnClickedButton1()?*
8. Для чого використовується функція *Format()?*
9. Яку функцію виконує *_T()* і як вимкнути використання унікоду?

Створення діалогового вікна, коли головним є документ

Постановка завдання

До пункту меню *Файл* додати підпункт *«Show Dialog…»,* при виборі якого на екрані з'являється діалогове вікно. У цьому вікні, крім *Ok* і *Отмена,* повинні бути стартова кнопка і текстове вікно. При натисканні стартової кнопки в текстовому вікні повинен з'явитися певний текст, який при натисканні кнопки *Ok* відтвориться в клієнтській області екрана.

Створення пункту меню та обробника події

1. Создать проект / MFC / Приложение MFC, Имя – dia.

2. У вікні *Тип приложения* вибрати *Один документ* і зняти прапорець *Использовать библиотеки с поддержкой Юникода.* Клацнути на *Готово.*

3. Додати в меню *Файл* команду *Show Dialog*. Для цього вибрати вкладку *Окно ресурсов dia*. Відкрити папку *dia* і потім *dia.rc* (ресурси). Відкрити папку *Menu* і в ній двічі клацнути по ідентифікатору головного вікна *IDR_MAINFRAME*. У вікні редактора з'являються пункти меню. Меню для *Файл* – розкрито. Виділити в ньому *Печать* і натиснути *Insert*.

4. Вище *Печать* з'являється порожній рядок. Занести в нього назву пункту – *Show Dialog.* Автоматично цей пункт меню отримає ідентифікатор *ID_FILE_ SHOWDIALOG*. Це ім'я можна побачити у властивостях (*Свойства*) для даного пункту в рядку *ИД*.

Створення діалогового вікна

1. Вибрати вкладку *Окно ресурсов dia.* Виділити *Dialog*. Викликати контекстне меню, а в ньому – *Вставить Dialog*. Діалогове вікно, що з'явилося, уже має кнопки *Ok* і *Отмена*. Заголовок вікна *Dialog*, а ім'я за замовчуванням – *IDD_DIALOG1* (див. *Свойства* рядок *ИД*).

Додавання керувальних елементів можна виконати двома способами. Можна просто брати потрібний елемент керування з палітри *Панель элементов* і перетягувати в позицію, де бажано його

розмістити. А можна клацнути на елементі керування зі списку в *Панель элементов*, щоб вибрати його, а потім клацнути на полі діалогового вікна, куди потрібно його розмістити. Коли елемент з'явиться, його можна переміщати і змінювати розміри.

У кожного елемента керування є ім'я за замовчуванням. Його можна побачити у рядку *ИД* вікна *Свойства*.

2.Додати в діалогове вікно *Dialog* кнопку *Button1*. Потім викликати вікно властивостей *Свойства* і у рядку *Подпись* ввести текст: «Натисни!».

3.Додати в діалогове вікно керувальний елемент – текстове вікно (у списку *Панель элементов* цей елемент називається *ab|Edit Control).* За замовчуванням його ідентифікатор *IDC_EDIT1*.

Створення класу діалогового вікна

Виділити діалогове вікно, вибрати в головному меню пункт *Проект / Добавить класс*, з'являється вікно *Мастер добавления классов - dia*. У рядок *Имя класса* ввести ім'я класу, наприклад *Dlg*. У вікні *Базовый класс* виберіть ім'я базового класу *CDialog.* Натиснути *Finish*. У списку класів з'явився клас *Dlg*. Якщо вибрати вкладку *Обозреватель решений*, то можна побачити, що з'явилися файли *Dlg.h* і *Dlg.cpp.*

Зв’язування методів діалогового класу з елементами діалогового вікна

Ставиться завдання пов'язати методи створеного класу *Dlg* з елементами керування діалогового вікна *Dialog*. Йдеться про створення обробників подій.

Обробник натискання стартової кнопки «Натисни!»

Виділити кнопку «Натисни!», викликати контекстне меню і в ньому клацнути по *Добавить обработчик событий*. У вікні, що з'явилося, в *Тип сообщения* вибрати *BN_CLICKED*, а у списку класів вибрати клас *Dlg*. Натиснути кнопку *Добавить/править*. У результаті буде створений обробник *OnBnClickedButton1().* Його прототип розміщується у файлі *Dlg*.h і має вигляд

```
public: afx_msg void OnBnClickedButton1();
```

Необхідно звернути увагу на рядок символів *afx_msg*. Символи *afx* є абревіатурою від словосполучення *Application FrameWorkX*. Префікс *afx_msg* автоматично ставиться перед обробником подій, щоб вони вирізнялися з-поміж інших членів-функцій. Потрібно пам’ятати, що члени-функції, які оголошені в базовому класі *CWnd*, завжди мають префікс *afx_msg.*

У файлі *Dlg.cpp* розміщується заготовка обробника події – натисненням стартової кнопки «Натисни!».

```
void Dlg::OnBnClickedButton1()
{
 /* TODO: Добавьте свой код обработки
 уведомлений */
}
```

У цьому обробнику потрібно запрограмувати виведення певного тексту в текстове вікно при натисненні стартової кнопки «Натисни!». Для цього у створеному класі *Dlg* має бути відповідний елемент даних типу рядок символів (*CString*). Для його створення необхідно в діалоговому вікні виділити текстове вікно, викликати контекстне меню і вибрати *Добавить переменную*. У вікні, що з'явиться, вибрати мітку доступу *public.* Тип змінної за замовчуванням стоїть *CEdit.* Щоб його змінити на *CString*, необхідно у рядку *Категория* замість *Control* вибрати *Value.* Це приведе до зміни типу змінної на *CString.* Ім'я змінної нехай буде *m_edit1*.

Обмін інформацією між *m_edit1* і *IDC_EDIT1* відбувається у методі *DoDataExchange()*. Можна переконатися, що в цьому методі з'явився відповідний оператор

```
void Dlg::DoDataExchange(CDataExchange*pDX)
{
   ……..
DDX_Text(pDX,IDC_EDIT1,m_edit1);
}
```

Функція *DDX_Text()* переміщає дані між *m_edit1* та елементом редагування *IDC_EDIT1.* Якби ми додали елемент-даних *CString m_edit1,* виділивши клас *Dlg* і вибравши в контекстному меню *Добавить переменную,* функцію *DDX_Text()* у метод *DoDataExchange()* довелося б включати самостійно.

До класу *Dlg* додайте змінну *public, Cstring, m_text.* Тепер можна визначити вміст обробника події натисненням стартової кнопки.

```
void Dlg::OnBnClickedButton1()
{
 m_edit1="Будьте щасливі!”;
 UpdateData(false);
}
```

Параметр *false* означає, що *IDC_EDIT1=m_edit1*. Тобто відбувається занесення у текстове вікно значення елемента даних *m_edit1.* А *true* означає, що вміст текстового вікна присвоюється елементу даних *m_edit1.*

Обробник натиснення кнопки *Ok*

При виборі кнопки *Ok* необхідно, навпаки, із текстового вікна присвоїти рядок символів елементу даних *m_edit1.* Тобто реалізувати *m_edit1=IDC_EDIT1.* У даній програмі цього можна не робити, оскільки зміст текстового вікна не змінювався. Але взагалі в іншій програмі це може спостерігатися. Тому є сенс

показати, як бути в цьому разі. Необхідно створити обробник

```
void Dlg::OnBnClickedOk()
{
 UpdateData(true);
 OnOK();
}
```

Відображення діалогового вікна

Створити обробник пункту меню *«Show Dialg».* Для цього необхідно вибрати вкладку *Окно ресурсов*, розкрити папку *Menu*, виділити пункт *IDR_MAINFRAME*, викликати контекстне меню, клацнути на *Открыть*. У вікні редактора меню вибрати пункт *«Show Dialog»,* викликати контекстне меню, клацнути на *Добавить обработчик событий*. У вікні, що з'явилося, вибрати *COMMAND* та *CdiaView.* Після натиснення кнопки *Добавить/править* з’явиться обробник *void CdiaView::OnFileShowdialog().* У ньому потрібно використати функцію *DoModal()*, яка викликає так зване модальне вікно.

Діалогові вікна можуть бути модальними і немодальними. Поки працює модальне вікно, всі останні вікна працювати не можуть. У той самий час можуть одночасно працювати декілька немодальних вікон.

Щоб викликати *DoModal()*, потрібно створити об'єкт класу *Dlg.* Назвемо його *dlg1*. Необхідно також

пам'ятати, що обробка даних відбувається в класі документа *CdiaDoc,* а результати виводяться в *CdiaView*. Тому в *CdiaDoc* потрібно визначити елемент даних *m_stroka* типу *CString* із міткою доступу *public*. Як уже було сказано вище, для цього потрібно у вкладці *Окно классов* виділити клас *CdiaDoc*, викликати контекстне меню, вибрати *Добавить / Добавить переменную* й у вікні *Мастер добавления переменной-члена - dia* ввести *public, CString, m_stroka.*

Нижче наводиться текст обробника:

```
void CdiaView::OnFileShowdialog()
{
 Dlg dlg1;                          //Створили об'єкт .
 Int rezult=(int)dlg1.DoModal();
 //Щоб відобразити діалог у модальному режимі, викликали
 // DoModal(). При натисненні на кнопку Ок повертається
 // значення IDOK

 if(rezult==IDOK)
 {
  CdiaDoc*pDoc=GetDocument();
  ASSERT_VALID(pDoc);
  pDoc->m_stroka=dlg1.m_text;
 // Занесли в елемент даних m_stroka класу Doc значення
 // m_text класу Dlg.
 Invalidate();
//Оголошуємо, що вміст вікна недійсний та може бути
// замінений.
 // TODO: Добавьте свой код программы обработки здесь
```

```
    }
}
```

Увага! Щоб клас *CdiaView* зміг працювати з класом *Dlg*, необхідно у файл *diaView.cpp* додати

```
#include "Dlg.h"
```

Відображення в клієнтському вікні

Відомо, що для відображення в клієнтському вікні використовується метод *OnDraw().*

```
void CdiaView::OnDraw(CDC* pDC/*pDC*/)
{
    CdiaDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
     return;
    pDC->TextOutA(0,0,pDoc->m_stroka);
    //Додано
    /* TODO: Добавьте свой код программы обработки
   здесь */
}
```

У ньому додано виклик за допомогою покажчика *pDC* на контекст *CDC* методу *TextOutA()* виведення рядка символів на екран в область із координатами (0,0). Текст, що виводиться, містить змінну *m_stroka* класу *CdiaDoc*. Тому для роботи з нею використовується покажчик *pDoc.*

Програма готова. Після запуску виберіть пункт меню *Show Dialog,* у текстовому вікні з'явиться заданий рядок символів. Після натиснення кнопки

“Ok” цей самий текст з’явиться в клієнтській області головного вікна.

Контрольні запитання

1. Як створити діалогове вікно?
2. Як створити клас діалогового вікна?
3. Як створити обробник натиснення стартової кнопки?
4. Навіщо потрібно створювати елементи даних типу *CString* для класів *Dlg* і *CdiaDoc*?
5. Навіщо потрібна функція *DoModal()* і що вона повертає?
6. Чим модальні вікна відрізняються від немодальних?
7. Для чого використовується функція *UpdateData(),* яких значень набувають її параметри і що вони означають?
8. Для чого використовується функція *Invalidate()*?
9. Яка функція використовується для виведення рядка символів на екран і які у неї параметри?

Створення та редагування ресурсів меню під час роботи із *C++* стандарту *ISO/ANSI*

Постановка завдання

У пункт меню *File* додати підпункт «*Privet*», при виборі якого на екрані з’явиться текст «Будьте

здорові!». Додати кнопку інструментів, натиснення на яку дублює роботу доданого підпункту меню.

Додавання нової команди меню

1. Создать проект / MFC / Приложение MFC, Имя – menu1.

2. У вікні *Тип приложения* вибрати *Один документ* і зняти прапорець *Использовать библиотеки с поддержкой Юникода.* Клацнути на *Готово.*

3. Вибрати вкладку *Окно ресурсов*. Клацнути по значку «+» біля *menu1*. Відкрити папку *menu1.rc* (ресурси). Відкрити папку *Menu* і в ній двічі клацнути по ідентифікатору головного вікна *IDR_MAINFRAME*. У вікні редактора з'являються пункти меню. Меню для *Файл* – розкрито. Виділити в ньому *Печать* і натиснути *Insert*.

4. Вище *Печать* з'являється порожній рядок. Занести в нього назву пункту меню *«PRIVET».* Автоматично цей пункт меню отримає ідентифікатор *ID_FILE_PRIVET*. Це ім'я можна побачити у властивостях (*Свойства*) для даного пункту в рядку *ИД*.

5. Для підключення нової команди меню до коду програми необхідно виділити пункт *«Privet»,* правою кнопкою миші викликати контекстне меню та вибрати в ньому *Добавить обработчик событий*.

6. З'явиться вікно *Мастер обработчика события-menu1.* У вікні *Тип сообщения* вибрати *COMMAND* (як

правило, воно буває встановлене автоматично), а у списку *Список классов* вибрати клас *Cmenu1View* та клацнути на кнопці *Добавить/править.*

7. У файлі *menu1View.cpp* з'явиться заготовка для функції – обробника подій. Пункт меню названий на латині. Тому ім'я обробника автоматично встановлюється *OnFileprivet().* Для пункту меню, названого на кирилиці, ім'я може бути таким: *OnFile32772().* Незалежно від імені обробника в ньому необхідно запрограмувати запис рядка символом «Будьте здорові!» та виведення його на екран. Як відомо, інформація про те, що виводити на екран, надходить від об'єкта класу "Документ". Тому попередньо в клас *Cmenu1Doc* необхідно додати елемент даних типу *CString*. Назвемо його *stroka*. Для цього виберемо вкладку *Окно классов*, виділимо клас *Cmenu1Doc,* викличемо контекстне меню та вибиремо в ньому *Добавить / Добавить переменную*. З'явиться вікно *Мастер добавления переменной-члена - menu1.* У вікні *Доступ* вибираємо мітку доступу *public*. У вікні *Тип переменной* пишемо *CString.* У вікні *Имя переменной* поміщаємо *stroka*. Клацнути на *Готово*.

8. В обробнику пишемо необхідний код:

```
void Cmenu1View::OnFilePrivet()
{
  Cmenu1Doc*pDoc=GetDocument();
```

```
  ASSERT_VALID(pDoc);
  pDoc->stroka="Будьте здорові!";
 Invalidate();        //Оголошується недійсним зміст
                || вікна. Отже, його можна замінити на нове.
}
```

Запрограмуємо виведення на екран змінної *stroka.*

```
void Cmenu1View::OnDraw(CDC*pDC)
{
  pDC->TextOutA(0,0, pDoc->stroka);
// Виведення на екран змінної stroka в точку з координатами (0,0).
}
```

Додавання нової кнопки інструментів меню

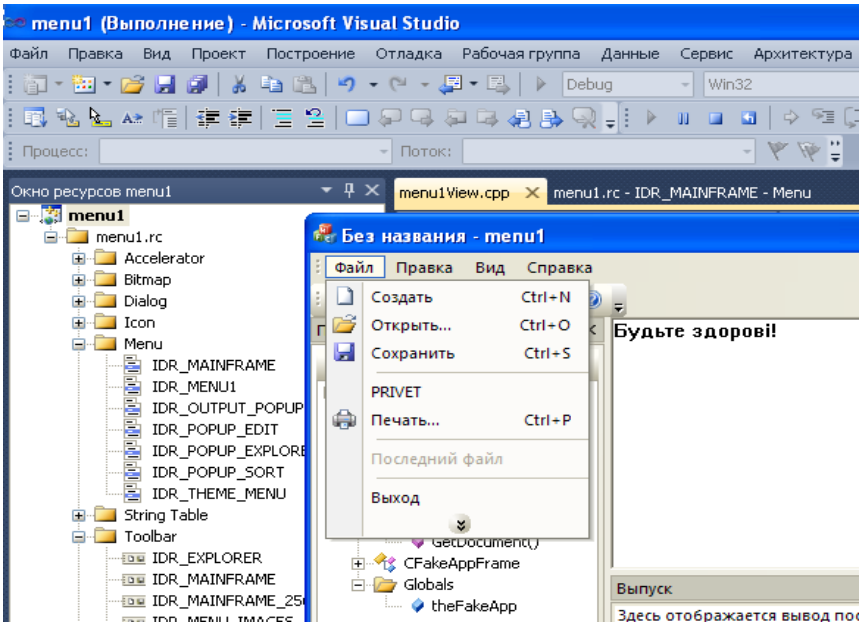
1.Створити кнопку інструментів, що дублює пункт меню *Файл / Privet.*

2.У вікні *Окно ресурсов menu1* вибрати папку *Toolbar* та розкрити її. Двічі клацнути по *IDR_MAINFRAME*.

З'явиться вікно графічного редактора. На панелі інструментів є порожня кнопка, а у графічному вікні – збільшене зображення кнопки. Якщо клацнути по порожній кнопці, у графічному вікні буде її зображення. Виберіть колір олівця та накресліть у збільшеній кнопці будь-який придуманий Вами символ для пункту меню *"Privet"*. Після закінчення наведіть курсор на нарисовану кнопку на панелі інструментів і викличте контекстне меню. У ньому виберіть *Свойства*. У рядку *ИД* клацніть по правій колонці та відкрийте список. У ньому виберіть

ID_FILE_PRIVET. Таким чином, установлюється зв'язок нової кнопки з пунктом меню *Файл / Privet*. У рядку *Приглашение* введіть *Privet\nPrivet* – це ім'я кнопки, та після *"\n"* – текст спливаючої підказки при наведенні курсора на кнопку.

3. Закрийте графічне вікно, клацнувши по хрестику праворуч. Програма готова.



Контрольні запитання

1. Як називається редактор меню?
2. Як вставити новий пункт меню?
3. Як створити обробник для нового пункту меню?
4. Як додати нову кнопку на панель інструментів?

5. Як зв’язати кнопку інструментів з необхідним пунктом меню?
6. Які дії необхідно виконати для пункту меню, щоб була спливаюча підказка?
7. Як запрограмувати виведений обробником текст та як його вивести на екран?

Робота з графікою за допомогою бібліотеки MFC

Постановка завдання

Створити W*indows*-програму для рисування типових графічних елементів: ліній, прямокутників, еліпсів, кіл, дуг із вибором пера і пензлика. Передбачити фарбування або штрихування деяких фігур та інших замкнених областей.

Основи рисування у вікні

Якщо хочете рисувати в клієнтській області вікна, то необхідно дотримуватися правила. Потрібно перерисовувати клієнтську область щоразу, коли додаток отримує повідомлення *WM_PAINT*. Існує безліч подій, що вимагають перерисовування вікна програми, наприклад зміна користувачем розміру вікна або перекриття частини вашого вікна вікном іншої програми. Операційна система *Windows* посилає поряд із повідомленням *WM_PAINT* інформацію, що дозволяє визначити, яка саме

частина клієнтської області має бути відновлена. Це означає, що не потрібно перерисовувати всю клієнтську область у відповідь на кожне повідомлення *WM_PAINT*, а тільки ту область, яка поновилася.

Виведення на клієнтську область дисплея є графічним незалежно від того, виводяться лінії, коло чи текст. Початок координат – у лівому верхньому кутку. Вісь *OY* спрямована вниз.

Windows вимагає, щоб ви визначали виведення, використовуючи інтерфейс графічних пристроїв *(Graphical Device Interface- GDI)*. Саме *GDI* дозволяє програмувати графічне виведення незалежно від устаткування, на якому він відображається.

Контекст пристрою

Коли потрібно щось нарисувати на пристрої графічного виводу, необхідно використовувати *контекст пристрою (device context)*. Це структура даних, яка визначена операційною системою *Windows.* Вона містить інформацію, що дозволяє *Windows* транслювати запити на виведення, які надходять у формі незалежних від пристрою викликів функцій *GDI*, в дії фізичного виведення на конкретний пристрій. За допомогою функцій *GDI* можна змінювати параметри, що впливають на виведення в контекст пристрою. Ці параметри називаються

атрибутами. До них належать колір рисування, колір фону, товщина лінії, шрифт та ін.

Контекст пристрою також дає можливість вибору координатних систем, що називаються *режимами відображення (mapping modes).* Кожен режим відображення в контексті пристрою має ідентифікатор *(ИД)* подібний до того, як це робиться з повідомленнями в *Windows*. Кожен має префікс *MM_*, що означає *mapping mode*. Зокрема, в режимі *MM_TEXT* логічною одиницею для задання координат є один піксель пристрою. Початок координат у лівому верхньому куті – точка (0,0). Вісь *OУ* прямує вниз, а *OX* – праворуч. Кількість пікселів залежить від роздільної спроможності дисплея і може бути, наприклад, 1024*x*768 або 1280*x*1024.

Клас *CDC*

Усі об'єкти цього класу і його похідних класів містять контекст пристрою і функції-члени, необхідні для того, щоб надсилати графіку і текст на дисплей і принтер. Таким чином, контекст є об'єктом класу *CDC*, що містить методи, необхідні для побудови зображення у вікні. Від класу *CDC* є похідний клас *CClientDC*, який має контекст клієнтської області вікна. Перевага класу *CClientDC* перед класом *CDC* полягає в тому, що він завжди містить контекст пристрою, що представляє тільки клієнтську область вікна.

Функція *OnDraw( )*

Майстер програм *Мастер приложений MFC* генерує клас для відображення інформації з документа в клієнтську зону вікна документа. Якщо ви дали ім'я додатку, наприклад, *My*, то цей клас автоматично отримає ім'я *CMyView* (вид). Нагадаємо, що майстер також створює класи *CMyDoc* (документ), *CMainFrame* (обрамлення головного вікна), *CMyApp* (застосування).

Визначення класу *CMyView* передбачає перевизначення деяких віртуальних функцій. Одна з них *OnDraw().* Вона викликається кожного разу, коли потрібно перерисовувати клієнтську область документа. Ця функція викликається каркасом додатка, коли програма отримує повідомлення *WM_PAINT.* Реалізація методу *OnDraw(),* створеного майстром *Мастер приложений MFC,* наводиться нижче.

```
void CMyView::OnDraw(CDC*/*pDC*/)
/* Не забути прибрати коментар або підставити власне ім'я покажчика. */
{
   CMyDoc*pDoc=GetDocument();
   /* Функція GetDocument() забезпечує доступ компілятора до членів класу CMyDoc. Інакше він матиме доступ лише до членів базового класу. */

   ASSERT_VALID(pDoc);
   //Тут перевіряється, що pDoc містить коректну адресу.
```

```
   if(!pDoc)
   return;
//TODO:тут необхідно додати код рисування для власних даних
}
```

Оператор *if(!pDoc)* перевіряє, що *pDoc* не дорівнює нулю. Параметр *pDC* для функції *OnDraw()* є покажчиком на контекст пристрою *(pointer to Device Context).*

Об'єкти *GDI*

У процесі рисування використовуються спеціальні об'єкти *GDI*, перелічені в таблиці.

Ідентифікатор	Клас	Назва
HPEN	*CPen*	Перо
HBRUSH	*CBrush*	Пензлик
HFONT	*CFont*	Шрифт
HBITMAP	*CBitmap*	Растрове зображення
HPALETTE	*CPalette*	Палітра
HRGN	*CRegion*	Область

Об'єкт «Перо»

Об'єкт «Перо» призначений для рисування. Щоб використовувати перо, необхідно вибрати його в контексті пристрою, в якому виконується рисування. Для цього служить функція-член класу *CDC SelectObject()*. Щоб вибрати перо, яке ви хочете

використовувати, згадана функція викликається з покажчиком на об'єкт пера як аргумент. Функція повертає покажчик на попередній використаний об'єкт пера, так що ви можете зберегти його і відновити старе перо по завершенні рисування.

Як і у разі з будь-якими іншими об'єктами в *Visual C++,* перед використанням потрібно створити об'єкт «Перо». При цьому можна задати його ширину, колір і тип лінії.

Синтаксис:

```
CPen ім’я_об'єкта_перо;
ім'я_об'єкта_перо.Createpen(стиль_пера,товщина,колір);
```

Ці самі дані можна задати за допомогою конструктора:

```
Cpen ім’я_об'єкта_перо(стиль_пера,товщина,колір);
```

Товщина лінії задається в пікселях. Для суцільної і порожньої ліній вона може бути будь-якою. Для всіх інших стилів товщина має дорівнювати лише одиниці.

Колір задається макросом *RGB* (червоний, зелений, блакитний). Кожен колір задається числом від 0 до 255. Наприклад, *RGB*(255,0,0) відповідає червоному кольору.

Стиль пера	Опис
PS_SOLID	Суцільна лінія
PS_DASH	Пунктирна лінія
PS_DOT	Точкова лінія
PS_DASHDOT	Штрихпунктирна лінія
PS_DASHDOTDOT	-*- з подвійними крапками
PS_NULL	Перо нічого не малює
PS_INSIDEFRAME	Суцільна лінія, але з крапками на межі пера, а не в його центрі

Приклад 1.

```
CPen pen1;              //Створений об'єкт pen1
pen1.CreatePen(PS_DOT,5,RGB(0,0,0);
// Перо буде рисувати пунктирну лінію завтовшки 5
// пікселів чорного кольору.
```

Такий спосіб рекомендується в тому разі, якщо одне і те саме перо може змінювати стиль, товщину або колір.

Приклад 2. Використовується конструктор

```
CPen pen2(PS_DOT,5,RGB(0,0,0));
```

Використання пера

Щоб використовувати перо, його потрібно вибрати в контексті пристрою, в якому виконується рисування. Для цього використовується метод класу *CDC*

SelectObject(). Його аргументом є адреса об'єкта «Перо». Він повертає адресу раніше встановленого пера. Цю адресу можна привласнити покажчику на клас *CPen* і відновити старе перо по закінченні рисування. Нижче наведено типовий приклад вибору пера.

```
CPen pen1;
pen1.CreatePen(PS_SOLID,1,RGB(0,0,0));
CPen*pOldPen=pDC->SelectObject(&pen1);
// Обрали перо pen1. Адресу старого пера занесли в
// покажчик pOldPen.
pDC->MoveTo(100,50);
// Перемістили курсор в задану точку (100,50).
pDC->LineTo(200,150);
// Нарисували лінію з точки (100,50) у точку (200,150).
……
pDC->SelectObject(pOldPen);
// Відновили старе перо.
```

Увага! Щоб стерти лінію або фігуру, потрібно повторно її відобразити кольором фону.

Об'єкт «Пензлик» (*Brush*)

Об'єкт «Пензлик» є растровим зображенням 8x8 пікселів. Його призначення – зафарбовування замкнених областей. Розрізняють два різновиди пензля: логічний і фізичний. Об'єкт *GDI* задає логічний пензлик з набором властивостей (стиль, колір, форма). Можливо, не всі ці властивості можуть бути

реалізовані обраним фізичним пристроєм (фізичним пензликом). Розрізняють чотири види логічних пензликів, при створенні яких використовуються різні методи.

- Суцільний пензлик одного кольору. Створюється за допомогою методу *CreateSolidBrush().*
- Стандартні пензлики. Використовують метод *CreateStockBrush().*
- Штрихові пензлики характеризуються кольором і штрихуванням. Використовують методи *CreateHatchBrush()* та *CreateSysColorBrush().*
- Шаблонні пензлі можуть мати довільну форму, що заливається растровим зображенням (*BMP*) або апаратно-незалежним растровим зображенням (*DIB*). Використовують методи *CreatePatternBrush()* та *CreateDibPatternBrush().*

Необхідно також пам'ятати, що стандартні пензлики можна створювати за допомогою конструкторів класу *CBrush*. Якщо параметри пензля з одним і тим самим ім'ям передбачають змінювати кілька разів, то зручно використовувати конструктор з одним параметром-кольором. Але тоді потрібно використовувати функцію *Selectobject()* для вибору цього пензлика. Наприклад,

```
CBrush brush1(RGB(255,0,0));
pDC → SelectObject(brush1);
// Вибрано пензлик brush1 із суцільним зафарбуванням
// червоним кольором.
```

Є також конструктор із двома параметрами: параметр зафарбовування та колір. Його використання виправдане, якщо пензлик із таким ідентифікатором свої параметри не змінює. У загальному вигляді визначення й одночасно вибір пензля за допомогою конструктора мають вигляд

Cbrush ім'я_пензлика(стиль_штрихування,колір);

Приклад

```
CBrush brush2(HS_CROSS, RGB(255,0,0));
// Перший аргумент задає стиль штрихування, другий –
// колір.
```

Аргумент стиля штрихування може набувати одне зі значень, показаних у таблиці:

Стиль штрихування	Опис
HS_HORIZONTAL	Горизонтальне штрихування
HS_VERTICAL	Вертикальне штрихування
HS_BDIAGONAL	Діагональне штрихування зліва направо зверху вниз під кутом 45
HS_FDIAGONAL	Діагональне штрихування зліва направо знизу вверх під кутом 45
HS_CROSS	Пересічне штрихування з горизонтальних і вертикальних ліній
HS_DIACROSS	Пересічне штрихування з діагональних ліній

Стандартні пензлики суцільного зафарбування також можна створювати за допомогою методу *SelectStockObject* (стандарт). Природно, що встановлений пензлик не має ідентифікатора. Такий пензлик називається *поточним*.

Стандарт суцільного зафарбування	Опис
GRAY_BRUSH	Сірий
BLACK_BRUSH	Чорний
HOLLOW_BRUSH	Порожній
LTGRAY_BRUSH	Світло-сірий
WHITE_BRUSH	Білий
NULL_BRUSH	Прозорий
DKGRAY_BRUSH	Темно-сірий

Приклад:

```
pDC->SelectStockObject(GRAY_BRUSH);
```

/*Установлений пензлик, який заливатиме фігури сірим кольором. Тут pDC, як і раніше, покажчик на об'єкт CDC. */

Використання пензлика

Подібно до використання пера при виборі нового пензлика рекомендується використовувати одну з функцій із префіксом *Select*, які повертають адресу старого пензлика. Це дозволить за необхідності відновити його.

```
 CBrush*pOldBrush=
(CBrush*)pDC->SelectStockObject(WHITE_BRUSH);
/* Вибрали пензлик і в покажчик pOldbrush занесли адресу
 старого.  */
pDC->SelectObject(pOldBrush);
/* Відновлення старого пензля за його адресою у покажчику
pOldBrush.  */
```

Функція *SelectObject* () повертає покажчик на старий пензлик або значення *NULL*, якщо операція завершена невдало. Повернений покажчик можна використовувати для відновлення старого пензлика в контексті пристрою.

Зафарбовування довільних замкнених областей

Прямокутники та еліпси зафарбовуються пензлем автоматично. Проте замкнені ділянки можуть також утворитися і внаслідок рисування серії ліній. За необхідності їх зафарбувати використовують метод *FloodFill().* Його прототип має вигляд

```
BOOL FloodFill(int x,int y,COLORREF crColor);
```

Тут *x* і *y* – координати точки всередині зафарбованої області; а *COLORREF crColor* – колір межі, яким нарисували цю область.

Приклад:

```
pDC->FloodFill(70,70,RGB(0,0,255));
/* Зафарбувати раніше встановленим пензлем замкнену
область.  */
```

Об'єкт «Растрове зображення»

Растрове зображення – об'єкт, що містить у собі прямокутну область із пікселів. У цій області можна задавати будь-яке зображення, а також зчитувати і записувати його у файл і робити з ним інші дії.

Об'єкт «Палітра»

У *Windows* розрізняють фізичні й логічні палітри. Логічна палітра призначена для роботи самої програми. Для створення палітри використовують методи *CreatePalette()* і *CreateHalftonePalette()*. Сформована палітра може бути змінена за допомогою методу *SetPaletteEntries().*

Є ще один спосіб керування кольором за допомогою макросу *RGB* (червоний, зелений, блакитний). Про нього уже було сказано вище.

Відображення графіки

Вибравши перо і пензлик, можна приступити до рисування ліній і різних фігур. Для цього використовують функції класу *CDC*, на які вказує покажчик *pDC,* що передається як параметр при виклику *OnDraw().* Вище уже були наведені приклади, коли викликалися функції *MoveTo()* і *LineTo()*. Розглянемо деякі функції:

1. *MoveTo().* Переміщення курсора. Клас *CDC* перевизначає цю функцію:

```
CPoint MoveTo (int x, int y);
  // Перейти в позицію x, y
CPoint MoveTo (POINT aPoint);
  // Перейти в позицію, визначену aPoint
```

Перша версія розглядає координату *x* як окремий аргумент. Друга бере один аргумент типу *POINT*, який є структурою:

```
typedef  struct tagPOINT
{
LONG x;
LONG y;
} POINT;
```

2. *LineTo().* Рисування лінії. Клас *CDC* перевизначає цю функцію.

```
BOOL LineTo(int x, int y);
  // Рисувати лінію до позиції x, y.
BOOL LineTo (POINT aPoint);
  // Рисувати лінію до позиції, визначеної aPoint.
```

Функція повертає TRUE, якщо лінія нарисована. Поточна позиція переміщається в точку, зазначену в кінці лінії. Це дозволяє рисувати серію сполучених ліній.

3. *Rectangle().* Рисування прямокутника. Нижче наводиться прототип одного з варіантів цієї функції:

```
BOOL Rectangle(int x1,int y1,int x1,int y2);
```

Тут *(x1, y1), (x2, y2)* – відповідно координати лівого верхнього і правого нижнього кутів прямокутника.

```
pDC->Rectangle(100,100,300,405);
```

Якщо було вибрано пензлик, то він зарисує прямокутник.

4. *Ellipse().* Рисування еліпса і, як окремий випадок, кола. Нижче наводиться прототип одного з варіантів цієї функції:

```
BOOL Ellipse(int x1,int y1, int x1, int y2);
```

Тут *(x1, y1), (x2, y2)* – відповідно координати лівого верхнього і правого нижнього кутів прямокутника, в який вписаний еліпс.

```
pDC->Ellipse (400,300,200,50);
```

Якщо був вибраний пензлик, то він зарисує еліпс.

5. *Arc().* Рисування дуги, еліпса і, як окремий випадок, кола. Можна нарисувати будь-які дуги та незамкнені фрагменти еліпсів і кіл. Якщо будуть нарисовані еліпси або кола, вони залишаться

незафарбованими. Нижче наводиться прототип одного з варіантів цієї функції:

```
BOOL Ellipse(int x1, int y1, int x2,int y2,
             int x3, int y3, int x4, int y4);
```

Тут *(x1, y1), (x2, y2)* – відповідно координати лівого верхнього і правого нижнього кутів прямокутника, в який вписана дуга;

(x3, y3), (x4, y4) – відповідно координати початкової і кінцевої точок нарисованого сегмента. Сегмент рисується проти годинникової стрілки. Якщо координати цих точок збігаються, то нарисована крива буде сегментом кола.

```
pDC ->Arc (50,50,150,150,100,50,150,100);
```

6. *SetPixel(x, y, колір);* Рисування по пікселях. Піксель з координатами *x, y* набуває заданого за допомогою *RGB()* атрибуту (кольору).

```
pDC->SetPixel (120,235,RGB(255,0,0));
```

У процесі рисування може знадобитися визначення поточного атрибуту пікселя із заданими координатами. Для цього використовують метод *GetPixel(x,y);*. Він повертає значення типу *COLORREF*. Щоб дізнатися значення кожної складової *Red, Green і*

Blue, використовують відповідно функції *GetRValue(), GetGValue(), GetBValue().*

```
COLORREF atribut;
atribut=pDC->GetPixel(200,200);
unsigned int colorRed,colorGreen,colorBlue;
colorRed=GetRValue(atribut);
colorGreen=GetGValue(atribut);
colorBlue=GetBValue(atribut);
```

Якщо є масив точок і потрібно з'єднати лінією кожну подальшу точку з попередньою, можна використовувати метод *PolyLine(),* у якому першим параметром є покажчик на масив об'єктів класу *CPoint* або елементів типу *POINT*, а другий – кількість точок.

```
CPoint a[200];
for(int i=0;i<200;i++)
{
  a[i].x=300+i;
  a[i].y=200+40*sin(0.1*(double)i);
pDC-
 >SetPixel(a[i].x+100,a[i].y,RGB(0,0,255));
}
```

Синусоїду можна було б побудувати іншим способом після виходу із циклу, застосувавши метод *pDC->Polyline(a,200).*

Приклад функції *OnDraw()* для рисування

```
void CMyView::OnDraw(CDC*pDC /*pDC*/)
{

 CMyDoc* pDoc = GetDocument();
 ASSERT_VALID(pDoc);
 if (!pDoc)
  return;

 CPen pen;          // Створений об'єкт pen - перо
 pen.CreatePen(PS_DOT,1,RGB(0,0,255));
/* Створено безпосередньо саме перо. Воно рисує  точкову
лінію завтовшки 1 піксель блакитного кольору.  */

CPen*pOldPen=pDC->SelectObject(&pen);
/* Тепер це перо вибрали для роботи. Одночасно записали
адресу старого пера в покажчик. Старе, за замовчуванням,
рисує суцільну чорну лінію завтовшки 1 піксель. */

Cbrush brush(HS_DIAGCROSS,RGB(0,0,255));
//Створили пензлик. Синє діагональне штрихування.

pDC->SelectObject(brush); // Вибрали цей пензлик
pDC->MoveTo(10,10);
/* Установили курсор у точку (10,10). Нарисували
 послідовність ліній.  */

pDC->LineTo(100,100);
pDC->LineTo(200,0);
pDC->LineTo(10,10);
pDC->Rectangle(600,10,680,100);
// Прямокутник, заштрихований пензлем.
```

```
CBrush*pOldBrush1=
(CBrush*)pDC->SelectStockObject(BLACK_BRUSH);
/* Вибрали новий (чорний) пензлик і запам'ятали адресу
старого пензлика. Використано явне перетворення типу*/

pDC->Rectangle(150,150,250,250);
/* Нарисувати прямокутник, який зафарбований новим
чорним пензлем.  */

pDC->SelectStockObject (LTGRAY_BRUSH);
// Вибрали стандартний світло-сірий поточний пензлик.
pDC->Ellipse(200,200,100,75);

/* Нарисований еліпс, зафарбований пензлем. Він частково
 перекрив чорний прямокутник.  */

pDC->SelectObject(pOldPen);
// Відновили старе перо.

pDC->SelectObject(pOldBrush1);
/* Відновили старий пензлик. Діагональна штриховка
 блакитного кольору.  */

pDC->Rectangle(600,200,680,250);
// Нарисували і зафарбували відновленими пером і пензлем.

pDC->SelectStockObject (HOLLOW_BRUSH);
// Змінили пензлик на порожній.

pDC->MoveTo(400,30);
pDC->LineTo(450,150);          // Накреслили еліпс.
pDC->Ellipse(400,30,450,150);
```

```
/*Лінію видно через еліпс, оскільки пензлик порожній. Якщо пензлик білий ( WHITE_BRUSH), то еліпс перекриває частину лінії. */

pDC->MoveTo(300,200);
pDC->LineTo(570,200);
pDC->LineTo(570,400);
pDC->LineTo(300,200);    // Нарисували трикутник.
pDC->SelectStockObject(GRAY_BRUSH);
// Вибрали сірий пензлик.

pDC->FloodFill(450,250,RGB(0,0,0));
// Зафарбували трикутник.

pDC->Arc (50,250,150,350,50,250,130,300);
// Нарисували дугу.
}
```

Послідовність дій для створення проекту

1. *Создать проект / MFC / Приложение MFC, Имя – My1*.
2. У вікні *Мастер приложений MFC - My1* для вкладки *Тип приложения* вибрати *Один документ* і зняти прапорець *Использовать библиотеки с поддержкой Юникода.* Клацнути на *Готово.*
3. У вікні *Окно классов* розкрити папку *My1,* вибрати клас *Cmy1View.* Відкриється список методів цього класу. Вибрати метод *OnDraw()* і двічі клацнути на ньому. У вікні редактора з'явиться заготовка методу.

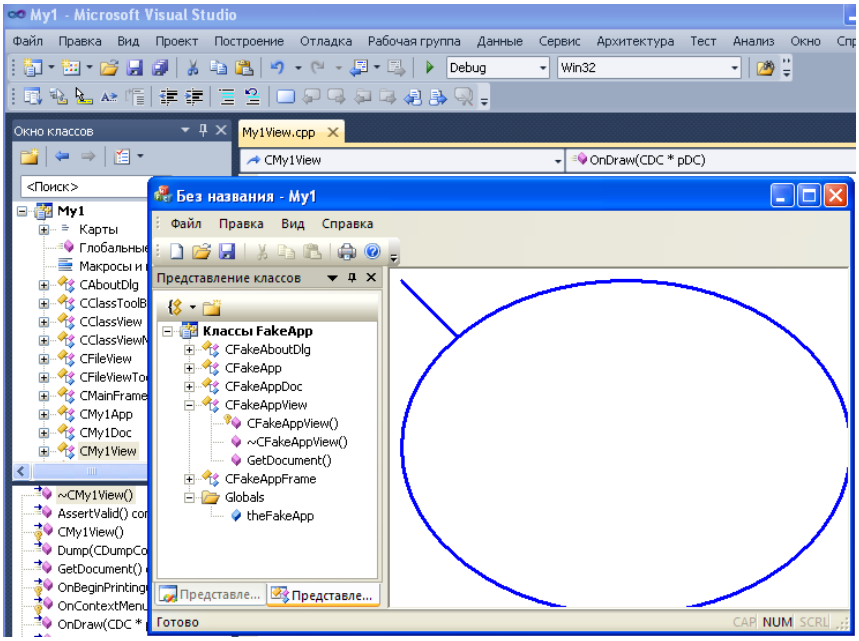
4. У круглих дужках ввести ім'я покажчика на контекст, наприклад, *OnDraw(CDC*pDC).* У заготовці функції *OnDraw()* в місці, позначеному коментарем

```
// TODO: добавьте здесь код отрисовки для собственных
// данных
```

ввести власний код для рисування, наприклад:

```
void CMy1View::OnDraw(CDC* pDC/*pDC*/)
{
 CMy1Doc* pDoc = GetDocument();
 ASSERT_VALID(pDoc);
 if (!pDoc)
  return;
CPen pen1(PS_SOLID,3,RGB(0,0,255));
pDC->SelectObject(&pen1);
pDC->MoveTo(200,200);
pDC->LineTo(10,10);
pDC->Ellipse(10,10,400,300);
 // TODO: добавьте здесь код отрисовки для
// собственных данных

}
```

7. Натисніть *Ctrl+F5* і запустіть програму для компіляції, компонування (за відсутності помилок) та виконання.



Контрольні запитання

1. Що таке контекст пристрою і яку функцію він виконує?
2. Яка функція здійснює графічне виведення, коли вона викликається і яке призначення функції *GetDocument()*?
3. Які є стандартні об'єкти *GDI* ?
4. Як можна встановлювати колір пера або пензлика?
5. Як можна встановити перо?
6. Як можна встановити пензлик?
7. Як нарисувати прямокутник?
8. Як нарисувати еліпс?
9. Як нарисувати дугу?

10. Навіщо потрібний покажчик на *CDC*?
11. Як зафарбувати довільну замкнену область?
12. Як залишити фігуру незафарбованою?
13. Які методи використовують, щоб нарисувати зображення по пікселях?
14. Як з'єднати лінією кожну наступну точку з попередньою, якщо є масив точок?
15. Як визначити складові кольори для заданого пікселя?

Створення та використання прапорців під час програмування на *C++* стандарту *ISO/ANSI*

Постановка завдання

Створити додаток, у якому діалогове вікно є головним. Розмістити в ньому три прапорці й текстове поле. Коли вибирають один із прапорців, він помічається «галочкою», а в текстовому полі повідомляють його номер.

Загальні вказівки

Прапорці дозволяють вибрати один або декілька варіантів із запропонованого списку. У *Windows* вони показані у вигляді маленьких квадратиків, порожніх або відмічених «галочкою». Якщо клацнути на

прапорець, його стан змінюється на протилежний: зі встановленого він стає знятим і навпаки.

Додавання прапорців

1. Создать проект / MFC / Приложение MFC, Имя – flags.

2. У вікні *Мастер приложений MFC - flags* для вкладки *Тип приложения* вибрати *На основе диалоговых окон* та зняти прапорець *Использовать библиотеки с поддержкой Юникода.* Клацнути на *Готово.* З'явиться діалогове вікно з кнопками *Ok* і *Оттмена.*

3. Стерти текст по центру *«//TODO…».* Для цього виділити його і натискувати кнопку *"Delete".*

4. З *Панель элементов* розмістити у вікні три прапорці *Check Box* і текстове вікно *ab| Edit Control*. Прапорці вирівняти по горизонталі.

5. Пов'язати прапорці з кодом програми, тобто з обробниками повідомлень. Необхідно створити обробник події для кожного прапорця. З цією метою потрібно виділити прапорець, наприклад *Check1*, викликати контекстне меню і вибрати в ньому *Добавить обработчик событий*. У вікні, що з'явилося, вибираємо *Тип сообщения BN_CLICKED*, а у списку класів – *CflagsDlg.* Як правило, вони вибрані за замовчуванням. Також за замовчуванням у рядку *Имя функции-обаботчика* встановлене ім'я функції - обробника *OnBnClickedCheck1.* Залишається натиснути

на кнопку *Добавить/править*. У файлі *flagsDlg.cpp* з'явиться заготовка для функції із зазначеним вище ім'ям. Так само потрібно створити обробники для *Check2* та *Check3.*

6. До класу *CFlagsDlg* необхідно додати елемент даних, відповідний текстовому вікну. Для цього потрібно виділити текстове вікно, викликати контекстне меню і вибрати в ньому *Добавить переменную*. Доступ зробити *private*. Тип *CEdit* уже встановлений за замовчуванням. Ім'я дамо *m_edit1.* Тобто елементом цього класу *CFlagsDlg* є об'єкт *m_edit1* класу *CEdit.* Разом із ним викликається в обробнику метод *SetWindowTextA(str)* класу *CEdit,* який розмістить у текстове вікно значення елемента даних *str* типу *CString.*

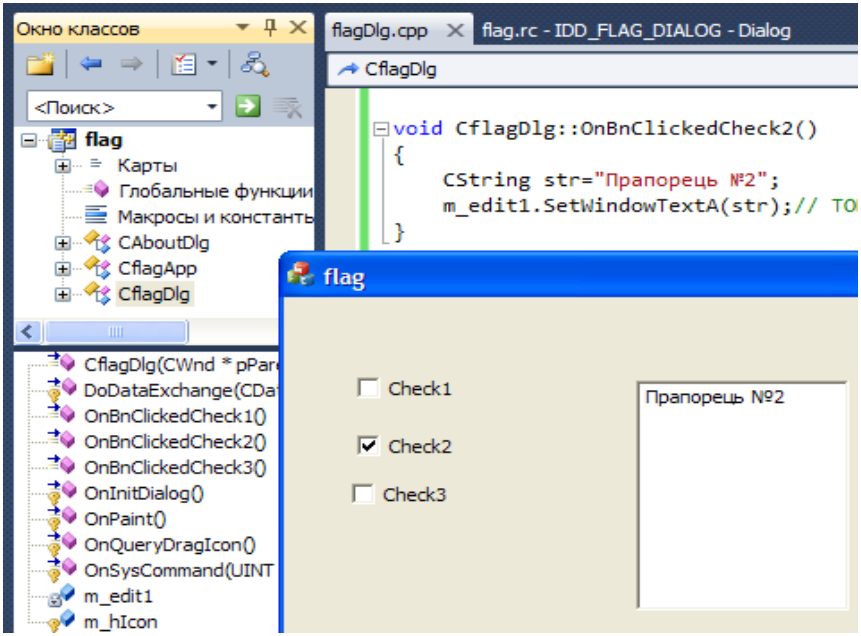
7. Заповнити заготовки обробників.

```
void CflagsDlg::OnBnClickedCheck1()
{
 CString str="Прапорець №1";
 m_edit1.SetWindowTextA(str);
 // TODO: добавьте свой код обработчика
 // уведомлений
}

void CflagsDlg::OnBnClickedCheck2()
{
 CString str="Прапорець №2";
 m_edit1.SetWindowTextA(str);
// TODO: добавьте свой код обработчика
 // уведомлений
```

```
}

void CflagsDlg::OnBnClickedCheck3()
{
   CString str="Прапорець №3";
 m_edit1.SetWindowTextA(str);
// TODO: добавьте свой код обработчика
 // уведомлений
}
```

У всіх трьох обробниках використовують локальну змінну *CString str,* яку ініціалізують певним текстом, а потім вона виводиться в текстове вікно.

8. Натиснувши *Ctrl+F5,* запускаємо програму.



Контрольні запитання

1. Чим відрізняється прапорець від звичайної кнопки з панелі інструментів?
2. Як створити прапорець?
3. Як створити обробник для прапорця?
4. Навіщо в цій програмі потрібний елемент *ab|Edit Control*?
5. Навіщо в цій програмі потрібний елемент даних *m_edit1*, якого він типу і як його створити?

Використання перемикачів під час програмування на *C++* стандарту *ISO/ANSI*

Постановка завдання

Створити додаток, у якому діалогове вікно є головним. Розмістити в ньому три перемикачі і текстове поле. Коли вибирається один із перемикачів, у текстовому полі повідомляється його номер.

Загальні вказівки

Перемикачі на відміну від прапорців дозволяють вибрати лише один варіант із запропонованого списку. У *Windows* вони подані у вигляді маленьких круглих кнопок, порожніх або помічених чорною точкою. При клацанні перемикач, як і прапорець, змінює свій стан на протилежний. На відміну від

прапорців перемикачі об'єднуються в групи і працюють спільно. У будь-який момент часу у групі може бути встановлений лише один перемикач. При його виборі всі інші перемикачі групи знімаються. Існують два способи групування перемикачів: або вони розташовуються всередині спеціального керуючого елемента (групового поля), або просто поміщаються в одне вікно. В останньому випадку всі перемикачі всередині вікна працюють разом навіть за відсутності групового поля.

Послідовність дій при створенні програми

1. Создать проект / MFC / Приложение MFC, Имя – switches.

2. У вікні *Мастер приложений MFC - switches* для вкладки *Тип приложения* вибрати *На основе диалоговых окон* та зняти прапорець *Использовать библиотеки с поддержкой Юникода.* Клацнути на *Готово.* З'явиться діалогове вікно з кнопками *Ok* і *Оттена.*

3. Стерти текст по центру *«//TODO…».* Для цього виділити його і натиснути кнопку *“Delete”.*

4. З *Панель элементов* розмістити у діалоговому вікні три перемикачі *Radio Button* і текстове вікно *ab| Edit Control*. Перемикачі вирівняти по горизонталі та пов'язати їх з кодом програми, тобто з обробниками повідомлень. Необхідно створити обробник події для кожного перемикача. З цією

метою необхідно виділити його, наприклад, *Radio1*, викликати контекстне меню і вибрати *Добавить обработчик событий*. У вікні, що з'явилося, вибираємо тип повідомлення – *BN_CLICKED*, а у списку класів – *CswitchesDlg.* Як правило, вони вибрані за замовчуванням. Також за замовчуванням у рядку *Имя функции-обработчика* встановлене ім'я функції-обробника *OnBnClickedRadio1.* Залишається натиснути на кнопку *Добавить/править*. Обробник можна створити і більш простим шляхом. Для цього досить двічі клацнути по виділеному перемикачу. У файлі *switchesDlg.cpp* з'явиться заготовка для функції із зазначеним вище ім'ям. Так само необхідно створити обробники для *Radio2* і *Radio3.*

5. У клас *CswitchesDlg* необхідно додати елемент даних, що відповідає текстовому вікну. Для цього необхідно виділити текстове вікно, викликати контекстне меню і вибрати в ньому *Добавить переменную*. Доступ зробити *private*. Тип *CEdit* уже встановлений за замовчуванням. Ім’я дамо *m_edit1*. Тобто елементом даного класу *CswitchesDlg* буде об’єкт *m_edit1* класу *CEdit*. З ним ми будемо викликати в обробнику метод *SetWindowTextA(str)* класу *CEdit,* який помістить у текстове вікно рядок *str* типу *CString.*

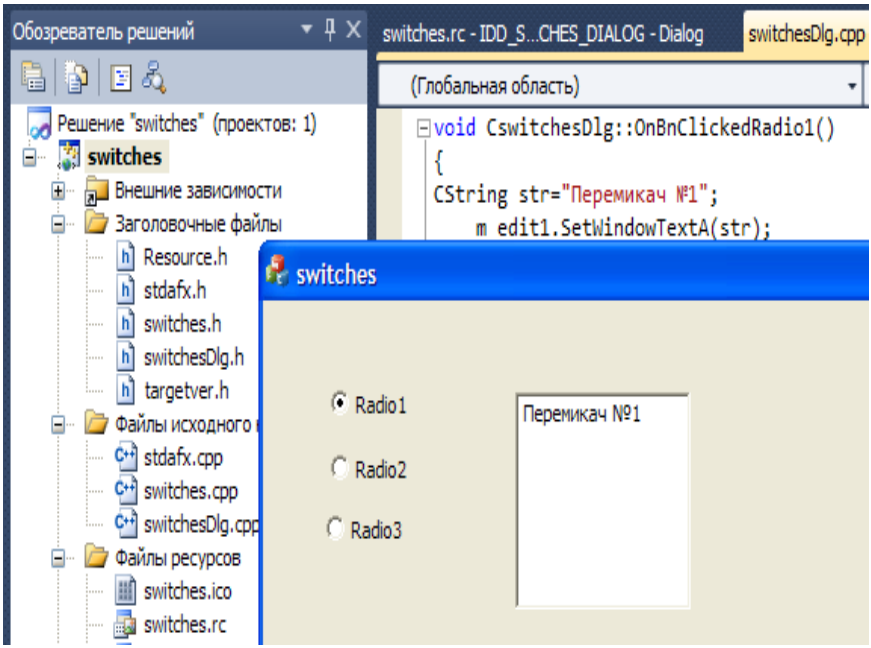
6. Заповнити заготовки обробників.

```
void CswitchesDlg:: OnBnClickedRadio1()
```

```
{
 CString str="Перемикач №1";
 m_edit1.SetWindowTextA(str);
// TODO: добавьте свой код обработчика …
 // уведомлений
}

void CswitchesDlg:: OnBnClickedRadio2()
{
 CString str="Перемикач №2";
 m_edit1.SetWindowTextA(str);
 // TODO: добавьте свой код обработчика
 // уведомлений
}

void CswitchesDlg::OnBnClickedRadio3()
{
 CString str="Перемикач №3";
 m_edit1.SetWindowTextA(str);
// TODO: добавьте свой код обработчика
 // уведомлений
}
```

У всіх трьох обробниках використовують локальну змінну *CString str,* яку ініціалізують певним текстом, потім її виводять у текстове вікно.

7. Натиснувши *Ctrl+F5,* запускаємо програму.



Контрольні запитання

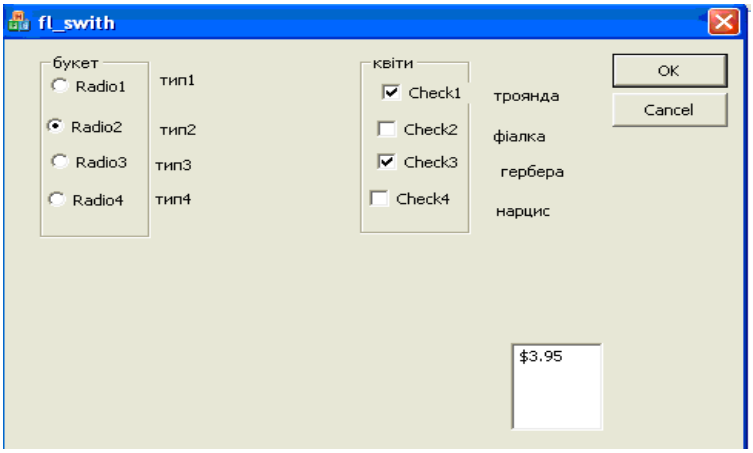
1. Чим відрізняється перемикач від прапорця?
2. Як створити перемикач?
3. Як створити обробник для перемикача?
4. Навіщо у цій програмі потрібний елемент *ab|Edit Control*?
5. Навіщо у цій програмі потрібний елемент даних *m_edit1*, якого він типу і як його створити?

Спільне використання прапорців і перемикачів під час програмування на *C++* стандарту *ISO/ANSI*

Постановка завдання

Написати програму для квіткового магазину з метою вибору типу композиції букета і квітів для нього. Коли користувач за допомогою перемикача вибирає певний тип букета, програма повинна визначити, які квіти входять у букет, установити відповідні прапорці і вивести ціну букета в текстовому полі.

Якщо користувач вибирає інший тип букета, програма повинна показати відповідні дані уже для нового типу.



Загальні вказівки

При створенні програми необхідно використовувати групові поля. Вони призначені для візуального і функціонального групування елементів. Усі перемикачі всередині групового поля працюють спільно, що дозволяє розмістити в діалоговому вікні декілька незалежних груп перемикачів. У додатку буде використано два групові поля.

Послідовність дій при створенні програми

1. Создать проект / MFC / Приложение MFC, Имя – flagswitches.

2. У вікні *Мастер приложений MFC - flagswitches* для вкладки *Тип приложения* вибрати *На основе диалоговых окон* та зняти прапорець *Использовать библиотеки с поддержкой Юникода.* Клацнути на *Готово.* З'явиться діалогове вікно з кнопками *Ok* і *Отмена.*

3. З *Панель элементов* розмістити у діалоговому вікні два групових поля *Group Box* і текстове вікно *ab| Edit Control*. У лівому груповому полі замінити напис *Static* на «Букет». Для цього виділити це поле, викликати контекстне меню, вибрати *Свойства.* У полі *Подпись* ввести новий напис і натиснути *Enter*. Відповідно праве групове поле підписати «Квіти».

4.Потрібно, щоб користувач вибирав тип букета за допомогою перемикача, а програма позначала квіти, що в нього входять, встановлюючи потрібні прапорці.

Тому слід додати в групу «Букет» чотири перемикачі, а в групу «Квіти» – чотири прапорці. Привласнити їм відповідні написи, наприклад, «Тип1»,…,«Тип 4», «Троянди», «Лілії» і т.п. Для цього з *Панель элементов* слід вибрати елемент *Aa Static Text* і розмістити його праворуч від перемикача або прапорця. У його властивостях у полі *Подпись* замість *Static* зробити необхідний напис. Вирівняти по горизонталі і вертикалі всі керуючі елементи.

5. До класу *Cflagsswitchesdlg* необхідно додати елемент даних, відповідний текстовому вікну. Для цього потрібно виділити текстове вікно, викликати контекстне меню і вибрати в ньому *Добавить переменную*. Доступ зробити *private*. Тип *CEdit* уже встановлений за замовчуванням. Ім'я дамо *m_edit1*. Тобто елементом даного класу *Cflagsswitchesdlg* є об'єкт *m_edit1* класу *CEdit*.

6. Для кожного перемикача створити обробник (від *OnBnClickedRadio1()* до *OnBnClickedRadio4().*

7. У клас *Cflagsswitchesdlg* необхідно додати елементи даних для кожного прапорця. Для цього потрібно виділити прапорець, викликати контекстне меню і вибрати в ньому *Добавить переменную*. Доступ зробити private. Тип *CButton* уже встановлений за замовчуванням. І’мя дамо відповідно від *m_check1* до *m_check4*. Тобто елементами даного класу *Cflagsswitchesdlg* є об'єкти *m_check1…m_check4* класу *CButton*.

8. Заповнити заготовки обробників. При цьому виникає необхідність примусової установки кожного із прапорців у заданий стан. Для цього використовують метод класу *CButton Setcheck* (параметр). Якщо параметр *true*, це означає, що прапорець вибирається і помічається «галочкою». Параметр *false* – прапорець знімається. Для кожного типу букета визначені квіти, які входять до його складу. Відповідні цим квітам прапорці будуть помічені. Якщо потрібно встановити поточний стан прапорця, використовується метод *GetCheck().* Далі наведені тексти обробників.

```
void CflagsswitchesDlg::OnBnClickedRadio1()
{
  m_check1.SetCheck(true);
  m_check2.SetCheck(true);
  m_check3.SetCheck(true);
  m_check4.SetCheck(true);
  m_edit1.SetWindowTextA("$6.86");
   UpdateData(false);
// TODO: добавьте свой код обработчика
 // уведомлений
}

void CflagsswitchesDlg::OnBnClickedRadio2()
{
  m_check1.SetCheck(true);
  m_check2.SetCheck(false);
  m_check3.SetCheck(true);
  m_check4.SetCheck(false);
  m_edit1.SetWindowTextA("$3.95");
```

```
    UpdateData(false);
// TODO: добавьте свой код обработчика
 // уведомлений
}

void CflagsswitchesDlg:: OnBnClickedRadio3()
{
  m_check1.SetCheck(false);
  m_check2.SetCheck(true);
  m_check3.SetCheck(false);
  m_check4.SetCheck(true);
  m_edit1.SetWindowTextA("$2.75");
    UpdateData(false);
// TODO: добавьте свой код обработчика
 // уведомлений
}

void CflagsswitchesDlg::OnBnClickedRadio4()
{
  m_check1.SetCheck(false);
  m_check2.SetCheck(false);
  m_check3.SetCheck(false);
  m_check4.SetCheck(false);
  m_edit1.SetWindowTextA("$0.00");
    UpdateData(false);
// TODO: добавьте свой код обработчика
 // уведомлений
}
```

Як бачимо з коду обробників, для букета типу №1 передбачено внести в букет усі чотири назви квітів. А тип №4 – порожній букет. Тому всі чотири прапорці – зняті. В усіх обробниках використовують локальну

змінну *CString str*, яка ініціалізована вартістю букета, а потім виводиться в текстове вікно.

```
UpdateData(false);
// false означає, що відбувається занесення в текстове
// вікно, а true- зчитування з вікна.
```

9. Натиснувши *Ctrl+F5,* запускаємо програму.

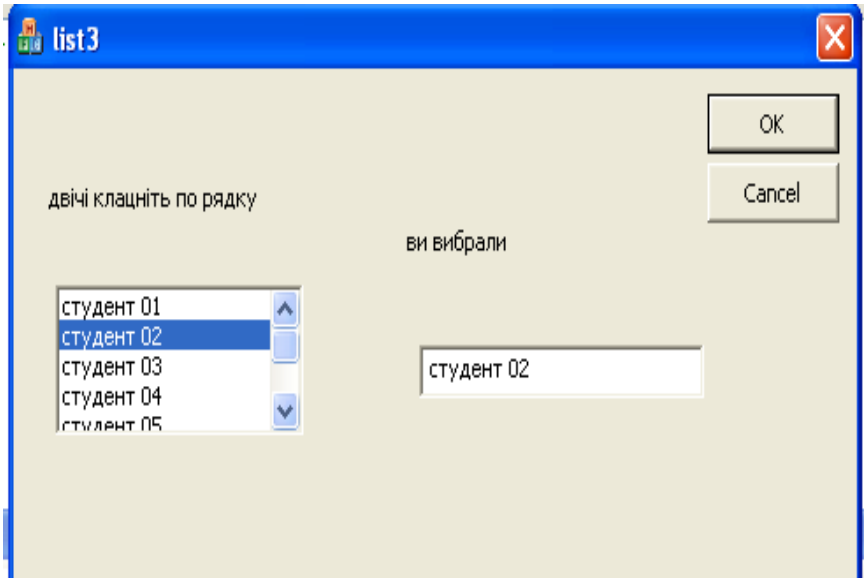
Контрольні запитання

1. Що таке групове вікно і навіщо воно потрібне?
2. Як змінити підпис у груповому вікні?
3. Як визначити поточний стан прапорця і задати потрібний?
4. Навіщо потрібно було вводити елементи даних *m_check1, m_edit1* та інші?
5. Навіщо потрібний метод *UpdateData()*?

Робота зі списками під час програмування на *C++* стандарту *ISO/ANSI*

Постановка задачі

Створити список із множиною рядків у діалоговому вікні. Розміри вікна зробити такими, щоб усі рядки не помістилися у вікні списку. В результаті праворуч від нього буде знаходитись смуга прокрутки. При подвійному клацанні на одному з рядків його вміст повинен з'являтися в текстовому вікні.



Загальні вказівки

Керуючі елементи можна відображувати і в звичайному (не діалоговому) вікні. Але в цьому випадку ви не зможете користуватися редактором діалогових вікон для їх створення і розміщення. Наприклад, щоб внести у вікно список, вам доведеться самостійно робити все, що, як правило, робить майстер *Class Wizard,* – редагувати схеми повідомлень, задавати ідентифікатори ресурсів і так далі.

Послідовність дій при створенні додатка

1. Создать проект / MFC / Приложение MFC, Имя – list1.

2. У вікні *Мастер приложений MFC - list1* для вкладки *Тип приложения* вибрати *На основе диалоговых окон* та зняти прапорець *Использовать библиотеки с поддержкой Юникода.* Клацнути на *Готово.* З'явиться діалогове вікно з кнопками *Ok* і *Отмена.*

3. У діалоговому вікні з *Панель элементов* розмістити керуючі елементи: *List Box* (список) і *ab|Edit Control* (текстове вікно). Над списком розмістити напис «Двічі клацніть по рядку», а над текстовим вікном – «Ви вибрали».

4. У клас *Clist1Dlg* додати елементи даних, відповідні списку і текстовому вікну. Для цього необхідно в діалоговому вікні виділити список, викликати контекстне меню, вибрати в ньому *Добавить переменную*. У вікні, що з'явилося, встановити доступ *private*, тип за замовчуванням *CListBox*, ім'я дати *m_list1*. Для текстового вікна ввести *private*, *CEdit, m_edit1*.

5. Додати у список рядки, з яких користувач буде вибирати. Ініціалізація даних діалогового вікна виконується в методі *OnInitDialog()* класу *Clist1Dlg.* У цьому методі ми повинні заповнити список рядками, щоб до моменту відображення діалогового вікна він містив необхідну інформацію. Конкретно занесемо 12

рядків «Студент 01», «Студент 02» і т.д. При цьому скористаємося методом *AddString()* класу *CListBox*. Нижче наведена частина цього методу, де відображається ініціалізація рядків.

```
BOOL Clist1Dlg::OnInitDialog()
{
CDialog::OnInitDialog();
    m_list1.AddString("Студент 01");
    m_list1.AddString("Студент 02");
    m_list1.AddString("Студент 03");
    m_list1.AddString("Студент 04");
    m_list1.AddString("Студент 05");
    m_list1.AddString("Студент 06");
    m_list1.AddString("Студент 07");
    m_list1.AddString("Студент 08");
    m_list1.AddString("Студент 09");
    m_list1.AddString("Студент 10");
    m_list1.AddString("Студент 11");
    m_list1.AddString("Студент 12");
 // Add "About..." menu item to system menu.
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
 return TRUE;
 // return TRUE unless you set the focus to
// a control
}
```

Можливо, виникне запитання: чому використовуємо запис «Студент 01» замість «Студент 1». Справа у тому, що рядки списку за замовчуванням сортуються в алфавітному порядку. Отже, якщо в другий рядок занести «Студент 2», то рядок «Студент 12» виявився б у списку перед ним. Якщо ви не хочете

сортувати, необхідно виділити вікно списку, викликати команду *Свойства*, перейти на вкладку *Сортировать* і у вікні, що відкрилося, замість True встановити *False*. Рядки, що заносяться у список, автоматично нумеруються, і надалі до них можна звертатися за індексом. Перший рядок має індекс 0, другий – 1 і т.д. Якщо запросити у об'єкта списку вибраний користувачем рядок, він повертає індекс цього рядка. Ініціалізація закінчена.

6. Тепер необхідно створити обробник подвійних клацань мишкою. За умовою задачі на вибраному рядку потрібно двічі клацнути лівою кнопкою мишки. При цьому рядок відображатиметься в текстовому вікні. Оскільки виділення рядка відбувається у списку, виділимо його вікно, викличемо контекстне меню, виберемо *Свойства*. У верхній частині вікна клацнути по вікну *События элементов управления* (помічений символом «блискавки»). У відкритому списку оброблюваних подій вибираємо *LBN_DBLCLK*. У віконці, що з'явилося, потрібно розкрити список. У ньому вибрати рядок *<Add>OnLnDblckList1* і клацнути по ньому. В результаті у файлі *list1.cpp* з'явиться заготовка для обробника подвійних клацань у списку. Нижче наведений текст обробника подвійних клацань.

```
void Clist1Dlg::OnLbnDblclkList1()
{
```

```
 CString text;
 m_list1.GetText(m_list1.GetCurSel(),text);
 m_edit1.SetWindowTextA(text);
 UpdateData(false);
// TODO: добавьте свой код обработчика
 // уведомлений
}
```

Тут функція *GetCurSel()* класу *ClistBox* отримує індекс вибраного рядка, а *GetText()* цього самого класу копіює рядок списку в буфер (локальна змінна *Cstring text)*. Потім метод *SetWindowTextA()* класу *CEdit* виводить змінну *text* у текстове вікно. При цьому потрібно пам'ятати, що параметр *false* функції *UpdateData()* означає введення у вікно.

Програма складена і її можна запустити, натиснувши *Ctrl+F5.*

Контрольні запитання

1. Як створити діалогове вікно?
2. Якого класу список і як створити його клас?
3. Як здійснити ініціалізацію списку ?
4. Навіщо потрібно створювати локальну змінну типу *CString* в обробнику подвійних клацань?
5. Навіщо потрібні функції *GetText()* і *GetCursel()*, якого вони класу?
6. Як працює обробник подвійних клацань?
7. Для чого використовується функція *UpdateData(),* яких значень набуває її параметр і що він означає?

8. Чим список відрізняється від текстового вікна?
9. Чому у списку рядок із текстом «Бутерброд 12» може виявитися попереду рядка «Бутерброд 2», хоча передбачалося їх сортування за збільшенням номерів ?
10. Як відмінити сортування у списку?

Робота з комбінованими полями під час програмування на *C++ ISO/ANSI*

Постановка завдання

Створити діалогове вікно. Розмістити в ньому комбіновані та текстові поля. Якщо клацнути на стрілці поряд із комбінованим полем, то в діалоговому вікні повинен розкритися список. Потрібно ініціалізувати цей список і запрограмувати так, щоб у випадку, якщо користувач вибирає один із рядків, він відображався в текстовому полі.

Загальні вказівки

Комбіноване поле також належить до керуючих елементів і є об'єктом класу *CCombobox*. Як правило, в програмі використовують методи цього класу, про які можна дізнатися у *Help*.

Послідовність дій при створенні програми

1. Создать проект / MFC / Приложение MFC, Имя – combo.

2. У вікні *Мастер приложений MFC - combo* для вкладки *Тип приложения* вибрати *На основе диалоговых окон* та зняти прапорець *Использовать библиотеки с поддержкой Юникода.* Клацнути на *Готово.* З'явиться діалогове вікно з кнопками *Ok* і *Отмена.*

3. У діалоговому вікні з *Панель элементов* розмістити керуючі елементи: *Combo Box* (комбіноване поле) і *ab| Edit Control* (текстове вікно). Над полем розмістити напис «Виберіть рядок», а над текстовим вікном – «Ви вибрали».

4. У клас *CComboDlg* додати елементи даних, відповідні комбінованому полю і текстовому вікну. Для цього необхідно в діалоговому вікні виділити комбіноване поле, викликати контекстне меню, вибрати в ньому *Добавить переменную*. У вікні, що з'явилося, встановити доступ *private*, тип за замовчуванням *CComboBox*, ім'я дати *m_combo1*. Для текстового вікна – *private, CEdit, m_edit1*. Ініціалізували комбіноване поле. Як і для списку ініціалізація даних виконується в методі *OnInitDialog().* Як і при роботі зі списком занесемо 12 рядків «Студент 01», «Студент 02» і т. д. При цьому скористаємося методом *AddString(),* але уже класу *CComboBox.*

Нижче наведена частина цього методу, яка відображує ініціалізацію рядків.

```
 BOOL CcomboDlg::OnInitDialog()
{
 CDialog::OnInitDialog();
  m_combo1.AddString("Студент 01");
  m_combo1.AddString("Студент 02");
  m_combo1.AddString("Студент 03");
  m_combo1.AddString("Студент 04");
  m_combo1.AddString("Студент 05");
  m_combo1.AddString("Студент 06");
  m_combo1.AddString("Студент 07");
  m_combo1.AddString("Студент 08");
  m_combo1.AddString("Студент 09");
  m_combo1.AddString("Студент 10");
  m_combo1.AddString("Студент 11");
  m_combo1.AddString("Студент 12");

 m_combo1.SetCurSel(0);
// Виклик рядка з індексом нуль
// Add "About..." menu item to system menu

 return TRUE;
// return TRUE  unless you set the focus to
// a control
}
```

У кінці ініціалізації запрограмоване виведення в комбіноване поле першого рядка при першій появі цього поля на екрані. Інакше воно виявиться порожнім.

5. При клацанні на комбінованому полі з'являється повідомлення *(Event) CBN_SELCHANGE*. Щоб створити

функцію-обробник цієї події, необхідно клацнути лівою кнопкою мишки. При цьому рядок відображатиметься в текстовому вікні. Оскільки виділення рядка відбувається у списку, виділимо його вікно, викличемо контекстне меню, виберемо *Свойства.* У верхній частині вікна клацнути по значку *События элемента управления* (помічений символом «блискавки»). У списку оброблюваних подій, що відкрився, вибираємо *CBN_SELCHANGE*. У вікні, що з'явилося, потрібно розкрити список. У ньому вибрати рядок *<Add>OnCbnSelchangeCombo1* і клацнути по ньому. У результаті у файлі *comboDlg.cpp* з'явиться заготовка для обробника вибору рядка у списку. Нижче наведений текст обробника.

```
void CcomboDlg:: OnCbnSelchangeCombo1()
{
 CString str;
m_combo1.GetLBText(m_combo1.GetCurSel(),str);
 m_edit1.SetWindowTextA(str);
 UpdateData(false);
// TODO: добавьте свой код обработчика
 // уведомлений
}
```

Тут функція *GetCurSel()* класу *CComboBox* отримує індекс вибраного рядка, а *GetLBText()* цього самого класу копіює рядок списку в буфер (локальна змінна *CString str*). Потім метод *SetWindowTextA()* класу *CEdit* виводить змінну *str* у текстове вікно. При цьому

потрібно пам'ятати, що параметр *false* функції *UpdateData()* означає занесення у вікно.

Програма складена і її можна запустити, натиснувши *Ctrl+F5.*

Контрольні запитання

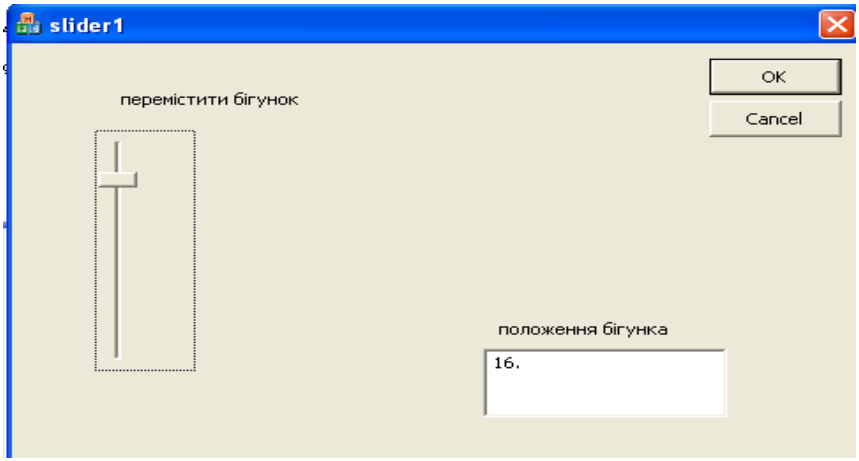
1. Що таке комбіноване поле?
2. Як ініціалізувати комбіноване поле?
3. Як здійснити ініціалізацію списку?
4. Для чого потрібні методи *GetCurSel(), SetCurSel()*?
5. Навіщо потрібні методи *GetLBText()* і *SetLBText(),* якому класу вони належать?
6. Для чого використовується функція *UpdateData()*, яких значень набуває її параметр і що вони означають?

Робота з бігунками під час програмування на *C++ ISO/ANSI*

Постановка задачі

Створити діалогове вікно. Розмістити в ньому бігунок, текстове поле і написи: «Перемістити бігунок» і «Положення бігунка» відповідно над бігунком і текстовим полем. Повзунок містить невеликий бігунок, який користувач переміщує уздовж шкали. Коли користувач перетягує бігунок мишкою, програма повинна виводити його нове

положення за шкалою від 1 до 100 (крайнє ліве положення - 1, крайнє праве - 100).



Загальні вказівки

Повзунок – керуючий елемент, який, як правило, застосовується для введення числових величин, наприклад, інтенсивності кольору (від 0 до 255). Це об'єкт класу *CSlider*. З методами цього класу можна ознайомитися в *Help.*

Послідовність дій при створенні програми

1. Создать проект / MFC / Приложение MFC, Имя – slider1.

2. У вікні *Мастер приложений MFC – slider1* для вкладки *Тип приложения* вибрати *На основе диалоговых окон* та зняти прапорець *Использовать библиотеки с поддержкой Юникода.* Клацнути на

Готово. З'явиться діалогове вікно з кнопками *Ok* і *Отмена.*

3. У діалоговому вікні з *Панель элементов* розмістити керуючі елементи: *Slider Control* (бігунок) і *ab| Edit Control* (текстове вікно). Над бігунком розмістити напис «Перемістити бігунок», а над текстовим вікном – «Положення бігунка».

4. У клас *Cslider1Dlg* додати елементи даних, відповідні бігунку і текстовому вікну. Для цього в діалоговому вікні виділити бігунок, викликати контекстне меню, вибрати в ньому *Добавить переменную*. У вікні, що з'явиться, встановити доступ *private*, тип за замовчуванням *CSliderCtrl*, дати ім'я *m_slider1.* Для текстового вікна ввести *private*, *CEdit, m_edit1.*

5. Ініціалізувати бігунок. Ініціалізація виконується в методі *OnInitDialog()*, але тепер уже класу *Cslider1Dlg*. Необхідно задати інтервал значень положення бігунка (1 – 100). Для цього використовуються методи *SetRangeMin()* і *SetRangeMax()* класу *CSlider*. У кожному з них по два параметри. Перший задає числове значення межі знизу або зверху, а другий набуває значень *true* (потрібно перерисовувати бігунок після зміни інтервалу) або *false* (відмовляємося від перерисовування). Щоб вивести в текстовому полі вихідне положення бігунка (позиція 1), використовується метод для виведення в текстове вікно *SetWindowTextA().* Нижче наведена частина

методу *OnInitDialog(),* де відображена ініціалізація бігунка.

```
BOOL Cslider1Dlg::OnInitDialog()
{
 CDialog::OnInitDialog();
   m_slider1.SetRangeMin(1,false);
   m_slider1.SetRangeMax(100,false);
   m_edit1.SetWindowTextA("1");
 // Add "About..." menu item to system menu.

 return TRUE;
 //return TRUE unless you set the focus to a
 //control
}
```

6. Запрограмувати обробку повідомлень від бігунка. При переміщенні бігунка елемент надсилає повідомлення *WM_HSCROLL* при горизонтальному переміщенні або *WM_VSCROLL* для вертикальних бігунків. Нехай бігунок буде горизонтальним. Для вибору орієнтації бігунка потрібно його виділити, викликати контекстне меню і вибрати *Properties*. У полі *Ориентация* вибрати *Горизонтально*. Щоб створити функцію-обробник події *WM_HSCROLL,* потрібно виділити все діалогове вікно, викликати контекстне меню, вибрати *Свойства*. У верхній частині вікна клацнути по *Сообщения.* У списку оброблюваних подій вибрати *WM_HSCROLL*, розкрити список, вибрати рядок *<Add>OnHScroll* і клацнути по

ньому. У результаті у файлі *slider1Dlg.cpp* з'явиться заготовка для обробника. Нижче наведений текст заготовки:

```
void Cslider1Dlg::OnHScroll (UINT nSBCode,
          UINT nPos, CScrollBar* pScrollBar)
{
 // TODO: Добавьте свой код программы
// обработки сообщения здесь

CDialog::OnHScroll(nSBCode,nPos,pScrollBar);
}
```

Тут *nSBCode* – код операції прокрутки. Всі можливі режими прокрутки показані в [1]. Останніми параметрами функції *OnHScroll()* є нова позиція бігунка і покажчик на керуючий елемент, від якого надійшло повідомлення. У нашому випадку необхідно перехоплювати і обробляти повідомлення з кодом *SB_THUMBPOSITION*, що надсилаються при переміщенні бігунка. Текст обробника наводиться нижче:

```
void Cslider1Dlg::OnHScroll(UINT nSBCode,
          UINT nPos, CScrollBar* pScrollBar)
{
 char s[5];
  if(nSBCode==SB_THUMBPOSITION)
  {
  gcvt(nPos,3,s);
  m_edit1.SetWindowTextA(s);
  UpdateData(false);
```

```
  }
  else
  // TODO: Добавьте свой код программы
 // обработки сообщения здесь

CDialog::OnHScroll(nSBCode,nPos,pScrollBar);
}
```

Тут використовується функція *gcvt()* для перетворення значення *nPos* у рядок символів *char s[5].* Довжина рядка символів залежить від максимального значення перетворюваного числа з урахуванням символу закінчення рядка. Потім метод *SetWindowTextA()* класу *CEdit* виводить вміст рядка *s* у текстове вікно. При цьому потрібно пам'ятати, що параметр *false* функції *UpdateData()* означає занесення рядка у вікно.

Програма готова і її можна запускати, натиснувши *Ctrl+F5.*

*Примітка***.** Занесення позиції бігунка в текстове вікно можна було зробити по-іншому:

*1.*Не створювати елемент даних *Cedit m_edit1*. Замість цього у клас *Cslider1Dlg* додати з доступом private елемент даних *CString m_text1.*

2.Майстер *Мастер классов* використовує спеціальні процедури, що дозволяють прив'язати створені ним елементи даних до полів діалогових вікон. Ці процедури мають назви «Обмін даними діалогового

вікна» і «Перевірка даних діалогового вікна» *(DDX/DDV, Dialog Data Exchange and Dialog Data Validation).* Коли користувач редагує поля діалогових вікон, процедури *DDV* перевіряють введені значення і блокують введення заборонених значень. Потім процедури *DDX* автоматично копіюють вміст полів діалогових вікон у прив'язані до них елементи даних класу. І навпаки, коли додаток змінює елементи даних класу, прив'язані до полів діалогового вікна, процедури *DDX* можуть відразу відображувати нові значення полів на екрані комп'ютера.

Ці процедури викликаються з методу *DoDataExchange()*. У нього необхідно додати виклик функції *DDX_Text(),* щоб пов'язати створений за нашим бажаням елемент даних *m_text1* з автоматично створеним ідентифікатором текстового вікна *IDC_EDIT1*.

Фрагмент методу *DoDataExchange()* наведено нижче:

```
void Cslider1Dlg::DoDataExchange(CDataExchange*pDX)
{
  CDialog::DoDataExchange(pDX);
  DDX_Control(pDX, IDC_SLIDER1, m_slider1);
  DDX_Text(pDX,IDC_EDIT1,m_text1);
// Додано. Без цього не буде виведення в текстове вікно.
}
```

3. Елемент даних *m_text1* використовується для занесення позиції бігунка. Тепер обробник *OnHScroll* має такий вигляд:

```
void Cslider1Dlg::OnHScroll(UINT nSBCode,
          UINT nPos, CScrollBar* pScrollBar)
{
 if(nSBCode==SB_THUMBPOSITION)
 {
  m_text1.Format("%ld",nPos);
  UpdateData(false);
 // TODO: Добавьте свой код программы
 // обработки сообщения здесь;
 }
 else

CDialog::OnHScroll(nSBCode,nPos,pScrollBar);
}
```

Програма готова.

Контрольні запитання

1. Навіщо в програмах використовуються повзунки?
2. Як задати орієнтацію повзунка?
3. Як здійснити ініціалізацію повзунка?
4. Як можна відображати позицію бігунка в текстовому полі?

Серіалізація стандартних об’єктів під час роботи з файлами

Постановка задачі

Створити однодокументну W*indows*-програму на *C++ ISO*|*ANSI* із використанням бібліотеки *MFC*. Передбачити читання рядка символів, запис його на диск і подальше читання з диска.

Що таке серіалізація

Документ програми на базі бібліотеки *MFC* – це об’єкт класу, який може бути досить складним. Як правило, він містить велике різноманіття об’єктів, кожен з яких містить інші об’єкти. І така структура може поширюватися на декілька рівнів.

Об'єкти класу містять функції-члени та змінні-члени, кожен з яких має специфікатори доступу. Щоб зберегти такі об'єкти у зовнішньому файлі, інформація, що записується в цей файл, повинна містити повні специфікації усіх структур класів. Процес читання з файла також повинен бути досить інтелектуальним, щоб повністю синтезувати вихідні об'єкти на основі даних файла.

Бібліотека *MFC* підтримує механізм, який називається *серіалізацією*. *Серіалізація* – це процес запису на диск або читання з нього об’єктів класу. У більшості програм на *Visual C++* уся робота з даними відбувається з документом. Основна ідея, що лежить

в основі серіалізації, полягає в тому, що будь-який серіалізований клас сам піклується про збереження та відновлення самого себе. Таким чином, щоб класи додатка стали серіалізованими, вони повинні записуватися у файл.

Основна можливість серіалізації з самого початку вбудовується *Мастером приложения* у програми. Обробники для пунктів меню *Файл/Открыть, Файл/Сохранить, Файл/Сохранить как…* передбачають, що буде серіалізація документів, і уже містять код, необхідний для її підтримки. Конкретно використовуються член-функція *Serialize(),* два макроси і порожній конструктор для документа. Подальше викладання наводиться для прикладу, коли створюється програма з ім'ям *serialstandart*. У цьому випадку майстер додатка створює класи *CserialstandartApp, MainFraim, CserialstandartDoc, CserialstandartView.*

Нижче наведено файл заголовків *serialstandartDoc.h.*

```
#pragma once

class CserialstandartDoc : public CDocument
{
protected:

 CserialstandartDoc();//Порожній конструктор
DECLARE_DYNCREATE(CserialstandartDoc)//Макрос
```

```
// Атрибути
public:
// Операції
public:
// Перевизначення
public:
 virtual BOOL OnNewDocument();
 virtual void Serialize (CArchive& ar);
                                  //Функція для серіалізації
// Реалізація
public:
 virtual ~CserialstandartDoc();
```

Необхідно звернути увагу на виділені жирним шрифтом фрагменти.

1. *CserialstandartDoc();*
2. *DECLARE_DYNCREATE(CserialstandartDoc);*
3. *virtual void Serialize(CArchive& ar);.*

Прототип порожнього конструктора *CserialstandartDoc()* використовується каркасом програми для синтезу об'єкта при читанні з файла.

Макрос *DECLARE_DYNCREATE (CserialstandartDoc)* дозволяє динамічно створювати об'єкти класу *CserialstandartDoc* каркасом програми під час серіалізації процесу введення. Параметром макроса є ім'я серіалізованого класу. У реалізації класу його доповнює макрос *IMPLEMENT_DYNCREATE (CserialstandartDoc, CDocument)* (див. файл *CserialstandartDoc.cpp).* Цей макрос також створюється автоматично майстром і свідчить, що клас *CserialstandartDoc* похідний від *CDocument*. У

свою чергу, *CDocument* є похідним від *CObject*. Необхідно зважати на те, що серіалізація можлива для об'єктів лише тих класів, які є прямими або непрямими спадкоємцями класу *CObject.*

```
virtual void Serialize (CArchive& ar);
{
 if (ar.IsStoring())
 {
  // TODO: добавьте код сохранения

 }
 else
 {
  // TODO: добавьте код загрузки

 }
}
```

Функція *virtual void Serialize(CArchive& ar)* серіалізує елементи даних класу. Оскільки *Мастер приложений* не знає, які дані містить ваш документ, процес запису і читання цієї інформації перебуває у вашій компетенції, про що зазначено в коментарях.

Аргументом функції є посилання *ar* на клас *CArchive*. Цей клас керує серіалізацією. Він є *MFC*-еквівалентом потокових операцій *C++,* які використовуються в консольних програмах для читання з клавіатури і запису на екран. Об'єкт цього класу представляє механізм того, щоб надсилати ваші об'єкти на запис у файл або на читання, при цьому реконструюючи їх. Він включає асоційований з ним

об'єкт класу *CFile*, який забезпечує можливості для *введення/виведення* бінарних файлів, а також дійсне підключення до конкретного фізичного файла. Під час серіалізації об'єкт *CFile* піклується про деталі операцій файлового введення/виведення, а об'єкт *CArchive* має справу з логікою структуризації даних об'єкта, що підлягають запису або реконструкції об'єктів на основі прочитаної інформації.

Клас *CArchive* перевантажує операції << і >>, необхідні для введення або виведення об'єктів класу, успадкованих від *CObject*, плюс діапазон базових типів даних. Конкретно це такі типи:

bool, float, double, BYTE, char, wchar_t, int, short, long, LONG, LONG LONG, WORD, unsigned int, DWORD, CObject , CString, SIZE, CSize, POINT, CPoint, RECT, CRect*.

Таким чином, функція серіалізації має такий вигляд:

```
void CserialstandartDoc::Serialize(CArchive&ar)
{
 if (ar.IsStoring())
 {
  ar<< елемен_даних_1;
  ar<< елемен_даних_2;
  ar<< елемен_даних_3;
  // TODO: добавьте код сохранения
 }
```

```
  else
  {
   ar>> елемен_даних_1;
   ar>> елемен_даних_2;
   ar>> елемен_даних_3;

   // TODO: добавьте код загрузки
  }
}
```

Якщо елементом даних серіалізованого класу (у нашому випадку класу *CserialstandartDoc)* є об'єкт якого-небудь іншого класу, для якого не перевантажені операції << і >>, то у функції *Serialize()* елементи даних цього іншого класу включаються в списки нарівні з елементами даних серіалізованого об'єкта. Об'єкт *CArchive* конструюється або для збереження об'єктів, або для їх прочитання. Функція *IsStoring()* повертає *TRUE*, якщо об'єкт призначений для запису у файл, і *FALSE*, якщо для читання.

Приклад. Створити програму з ім’ям *serstandart*.

1. *Создать проект : MFC, Приложение MFC*
2. У клас *CserstandartDoc* додати елемент даних *public: CString str.* Для цього потрібно виділити клас *CserstandartDoc*, викликати контекстне меню, *Добавить/Добавить переменную*. У вікні, що з'явиться, ввести елемент даних.
3. Рядок символів *str* потрібно вводити з клавіатури. Для цього необхідно створити обробник події –

введення символу. Щоб його створити, необхідно вибрати вкладку *Окно классов*, виділити клас *CserstandartView,* у контекстному меню вибрати *Свойства* і клацнути по піктограмі *Сообщения.* Клацнути по *WM_CHAR*, активізувати *<Add> OnChar().* У результаті майстер створить обробник

```
void CserstandartView::OnChar(UINT nChar,
                  UINT nRepCnt, UINT nFlags)
{
 // TODO: добавьте свой код обработчика сообщений или вызов
// стандартного
 CView::OnChar(nChar, nRepCnt, nFlags);
}
```

У ньому потрібно забезпечити введення з клавіатури рядка символів *str* класу *CserstandartDoc.* Для цього створюється покажчик *pDoc* на клас *CserstandartDoc*. Адресу в нього заносить функція *GetDocument().* Нижче наведено повний текст обробника:

```
void CserstandartView::OnChar(UINT nChar,
                  UINT nRepCnt, UINT nFlags)
{
 CserstandartDoc*pDoc=GetDocument();
 ASSERT_VALID(pDoc);
 pDoc->str+=(char)nChar;
                  //Посимвольне заповнення рядка
 Invalidate();
             //Обов'язково! Інакше не буде занесення
pDoc->SetModifiedFlag();
```

```
// TODO: добавьте свой код обработчика сообщений или вызов
// стандартного

 CView::OnChar(nChar, nRepCnt, nFlags);
}
```

Необхідно ще раз нагадати, що функція *Invalidate()* оголошує існуючий вид недійсним (очищає вікно виду). Отже, його можна відновити, що і відбувається. Щоб повідомити про зміну даних, викликають метод *SetModifiedFlag().* Тепер при виході з програми з'являтиметься питання: зберігати зміни чи ні.

4. Далі потрібно забезпечити виведення рядка символів *str* на екран у клієнтську область. Як уже неодноразово повідомлялося, для цього існує функція *OnDraw().* Майстер створив таку її заготовку.

```
void CserstandartView::OnDraw(CDC* /*pDC*/)
{
 CserstandartDoc* pDoc = GetDocument();
 ASSERT_VALID(pDoc);
 if (!pDoc)
  return;
 // TODO: добавьте здесь код отрисовки для собственных
// данных
 }
```

Зверніть увагу, що тут уже створений покажчик *pDoc*. Усе, що потрібно зробити, це в дужках зазначити ім'я покажчика на клас *CDC* і викликати

функцію *TextOut()* для виведення на екран рядка символів *str*. У результаті функція набуває вигляду:

```
void CserstandartView::OnDraw(CDC*pDC/*pDC*/)
{
 CserstandartDoc* pDoc = GetDocument();
 ASSERT_VALID(pDoc);
 if (!pDoc)
  return;
 pDC->TextOutA(0,20,pDoc->str);
                          //Виведення рядка на екран
 // TODO: добавьте здесь код отрисовки для
// собственных данных
}
```

5. Заповнити функцію *Serialize().* Для цього вибрати вкладку *Окно классов*, виділити клас *CserstandartDoc*, з методів цього класу вибрати *Serialize()* і додати в неї коди виведення на диск і читання з нього. Текст функції наведений нижче:

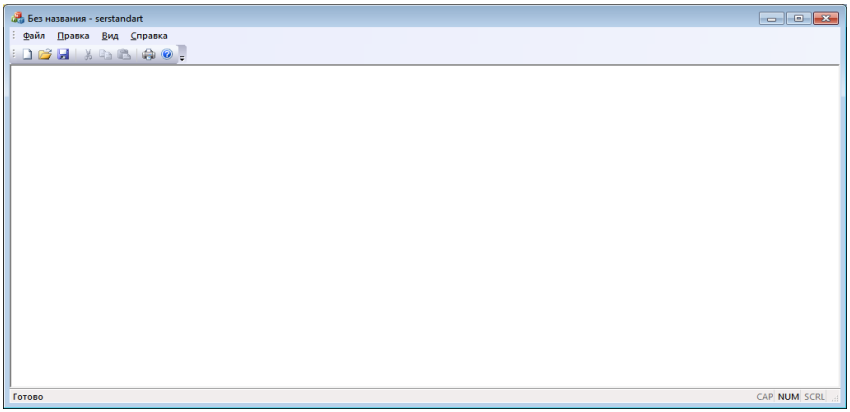
```
void CserstandartDoc::Serialize(CArchive& ar)
{
 if (ar.IsStoring())
 {
  ar<<str;              //Виведення str у файл на диск
 // TODO: добавьте код сохранения

 }
 else
 {
  ar>>str;                    //зчитування str з файла
```

```
// TODO: добавьте код загрузки
 }
}
```

6. Необхідно забезпечити очищення екрана при створенні нового документа. Для цього виділити клас *CserstandartDoc,* розкрити метод *OnNewDocument().* У ньому необхідно обнулити рядок символів *str =" ";* а також викликати функцію *UpdateAllViews(NULL).* Вона інформує всі види, пов'язані з документом, про те, що цей документ був модифікований і повинно бути оновлення всіх видів програми новими даними документа.

```
BOOL CserstandartDoc::OnNewDocument()
{
 if (!CDocument::OnNewDocument())
  return FALSE;
 str="";
UpdateAllViews(NULL);
// TODO: добавьте код повторной инициализации


 return TRUE;
}
```

Програма готова.



Контрольні запитання

1. Що таке серіалізація і навіщо вона потрібна?
2. Навіщо використовується порожній конструктор при серіалізації?
3. Які макроси використовуються при серіалізації і в яких файлах вони знаходяться?
4. Що собою являє об'єкт класу *CArchive* і навіщо він потрібний?
5. Які класи можуть бути серіалізованими?
6. Назвати типи об'єктів, для яких клас *CArchive* перевантажує операції << і >>.
7. Який вигляд має функція *Serialize()*?
8. Як створити обробник *OnChar()*?
9. Що робить метод *Invalidate()*?
10. Навіщо потрібний метод *SetModifiedFlag ()* ?
11. Що відбувається при виконанні функції *UpdateAllViews(NULL)* ?

Серіалізація результатів табуляції функції при використанні діалогу

Постановка завдання

Створити однодокументну *Windows*-програму на *C++ ISO/ANSI* із використанням бібліотеки *MFC.* Передбачити запуск із меню *Файл* діалогового режиму, читання з діалогового вікна вхідних даних для табуляції функції, запуск процесу табуляції, виведення результатів у вікно списку, а також у файл при виборі пункту меню *Файл/Сохранить* або *Файл /Сохранить как…* і подальше прочитання з диска *Файл /Открыть*.

Вхідними даними задатися самостійно.

Загальні вказівки

Серіалізація – процес запису на диск або читання з нього об’єктів класу. Дана робота є продовженням роботи «Серіалізація стандартних об'єктів при роботі з файлами». Розглянемо приклад створення додатка з ім'ям *ser_dia_tabul.*

Уведення вихідних даних *(double xn, dx, xk)* передбачається здійснювати з клавіатури в діалоговому режимі. Діалогове вікно з'являтиметься в разі вибору пункту меню *Показать диалог*. У цьому вікні, крім текстових вікон, має бути кнопка *«Пуск»*. При її натисненні запуститься процес табуляції. Результати виводитимуться у вікно *List Box*, а також

привласнюються змінній *CString str, яку необхідно створити для* діалогового класу *Dlg*. При натисненні кнопки *OK* виконується копіювання рядка *str* у рядок *CString m_stroka* класу *Cser_dia_tabulDoc.* Цей рядок за допомогою функції *Serialize()* можна зберігати на диску і читати, а за допомогою *OnDraw()* виводити на екран.

1.*Создать : Проект , Тип проекта - MFC, Шаблон– Приложение MFC, Имя – ser_dia_tabul, Тип приложения – Один документ.* Прибрати прапорець *Использовать библиотеки с поддержкой Юникода.*

2.Додати пункт меню *Файл/Показать диалог….*

3.Створити обробник події вибору цього пункту меню і залишити його порожнім (COMMAND/C*ser_dia_tabulView*).

4.Створити діалогове вікно (*Окно ресурсов /Dialog/Вставить Dialog*). Крім кнопок, що є за замовчуванням, розмістити в ньому кнопку *«Пуск»*, три текстові вікна *ab| Edit Control* і одне *List Box.* Проти кожного з вікон *ab|Edit Control* у вікнах *Aa Static Text* розмістити символічні імена відповідних змінних.

5.Клацнути на діалоговому вікні, викликати контекстне меню, вибрати *Добавить класс*. У вікні, що з'явиться, ввести ім'я діалогового класу – *Dlg*.

6.У клас *Dlg* додати змінну *Cstring str* та елементи даних: *medit1, medit2, medit3* для трьох вікон *ab|Edit Control* відповідно*.* Передбачається, що в них

заноситимуться *xn*, *dx, xk*. Для *List Box* додати елемент даних *mlist1*. Викликати для нього *Свойства* і вимкнути сортування рядків (у рядку *Сортировка* встановити *False*).

7.Створити обробник події – вибору кнопки *«Пуск»*. У ньому запрограмувати табуляцію функції.

Наприклад, для *y = log(x)* обробник має вигляд:

```
 void Dlg::OnBnClickedButton1()
{
 CString s1;
 double x,xn,dx,xk,y;
 medit1.GetWindowTextA(s1);
 xn=atof(s1);
 medit2.GetWindowTextA(s1);
 dx=atof(s1);
 medit3.GetWindowTextA(s1);
 xk=atof(s1);
 s1.Format("Результати:");
 for(x=xn;x<=xk;x+=dx)
 {
  if(x>0)
  {
   y=log(x);
   s1.Format(" x=%lf y=%lf",x,y);
   mlist1.AddString(s1);

  }
  else
  {
  s1.Format("Немає результату при x=%lf",x);
  mlist1.AddString(s1);
  }
```

```
str+=s1;
 }
 // TODO: добавьте свой код обработчика сообщений
 // или вызов стандартного
}
```

Щоб він міг працювати, у файли *Dlg.cpp* та C*ser_dia_tabulView.cpp* потрібно додати *# include "Dlg.h"* та *# include<math.h>* .

Тут використовується локальна змінна *CString s1*. Слід звернути увагу на те, що в циклі формується значення змінної *str* типу *CString*, що є елементом даних класу *Dlg.*

8. Заповнити обробник події-вибору пункту меню *«Показать диалог».*

Нижче наводиться його текст:

```
void Cser_dia_tabulView::OnFileShowdialog()
{
 Dlg  dlg1;
 int rezult=(int)dlg1.DoModal();
 if(rezult==IDOK)
 {
  Cser_dia_tabulDoc*pDoc=GetDocument();
  ASSERT_VALID(pDoc);
  if(!pDoc)
   return;
  pDoc->m_stroka=dlg1.str;
  Invalidate();
 }
 //TODO: добавьте свой код обработчика сообщений или
 // вызов стандартного
```

```
}
```

Тут відбувається копіювання інформації з елемента даних *str* класу *Dlg* в елемент даних *m_stroka* класу *Cser_dia_tabulDoc*. Як уже було зазначено, це потрібно для того, щоб здійснити серіалізацію отриманих результатів. Метод *Serialize()* класу *Cser_dia_tabulDoc* не може безпосередньо працювати з елементом даних класу *Dlg*. Копіювання відбувається при натисканні на кнопку *Ok*. Про те, що вона натискається, відомо за результатом роботи методу *DoModal()* діалогового класу. Для роботи з цим методом і з елементом даних *str* створено локальний об'єкт *dlg1* типу *Dlg*.

Щоб викликати елемент даних *m_stroka* класу *Cser_dia_tabulDoc,* створено покажчик *pDoc*.

9.Тепер необхідно забезпечити виведення рядка символів *m_stroka* на екран у клієнтську область. Як уже неодноразово зазначалося, для цього існує функція *OnDraw().* Майстер створив таку її заготовку:

```
void Cser_dia_tabulView::OnDraw(CDC*/*pDC*/)
{
 Cser_dia_tabulDoc* pDoc = GetDocument();
 ASSERT_VALID(pDoc);
 if (!pDoc)
  return;
 // TODO: добавьте здесь код отрисовки для собственных
// данных
```

```
}
```

Необхідно звернути увагу, що тут уже створений покажчик *pDoc*. Усе, що потрібно зробити, це в дужках зазначити ім'я покажчика на клас *CDC* і викликати функцію *TextOut()* для виведення на екран рядка символів *m_stroka.*

У результаті функція набуває вигляду:

```
void Cser_dia_tabulView::OnDraw(CDC*pDC/*pDC*/)
   {
    Cser_dia_tabulDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
     return;
    pDC->TextOutA(0,0,pDoc->m_stroka);
// TODO: добавьте здесь код отрисовки для собственных данных
   }
```

10. Заповнити функцію *Serialize().* Для цього вибрати вкладку *Окно классов*, виділити клас *Cser_dia_tabulDoc*, із методів цього класу вибрати *Serialize()* і додати в нього коди виведення на диск і читання з нього. Текст функції наведено нижче:

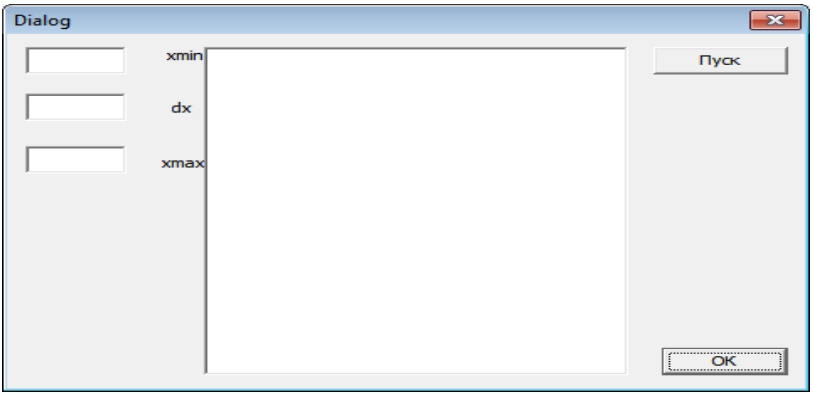
```
void Cser_dia_tabulDoc::Serialize(CArchive&ar)
{
 if (ar.IsStoring())
 {
 ar<<m_stroka; //Виведення m_stroka у файл на диск
 // TODO: добавьте код сохранения
 }
```

```
 else
 {
  ar>>m_stroka;                    // Прочитання з файла
// TODO: добавьте код загрузки
 }
}
```

11. Необхідно забезпечити очищення екрана при створенні нового документа. Для цього виділити клас *Cser_dia_tabulDoc,* розкрити метод *OnNewDocument().* У нього додати обнулення рядка символів *m_stroka =””;* а також функцію *UpdateAllViews(NULL).* Вона інформує всі види, пов'язані з документом, що цей документ був модифікований і має бути оновлення цих видів програми новими даними документа.

```
BOOL Cser_dia_tabulDoc::OnNewDocument()
{
 if (!CDocument::OnNewDocument())
  return FALSE;
 m_stroka="";
UpdateAllViews(NULL);
 // TODO: добавьте код повторной инициализации

 return TRUE;
}
```

Програма готова.



Контрольні запитання

1. Що таке серіалізація і навіщо вона потрібна?
2. Як у додатку забезпечити табуляцію функції?
3. Як формується рядок результатів?
4. Як здійснюється серіалізація результатів?
5. Як розпізнається вибір кнопки *Ok*?
6. Навіщо використовується локальний об'єкт діалогового типу?
7. Який вигляд має функція *Serialize()*?
8. В яких випадках створюється локальний покажчик на клас «Документ»?
9. Що відбувається при виконанні функції *UpdateAllViews(NULL)*?

Серіалізація нестандартних об'єктів під час роботи з файлами

Постановка завдання

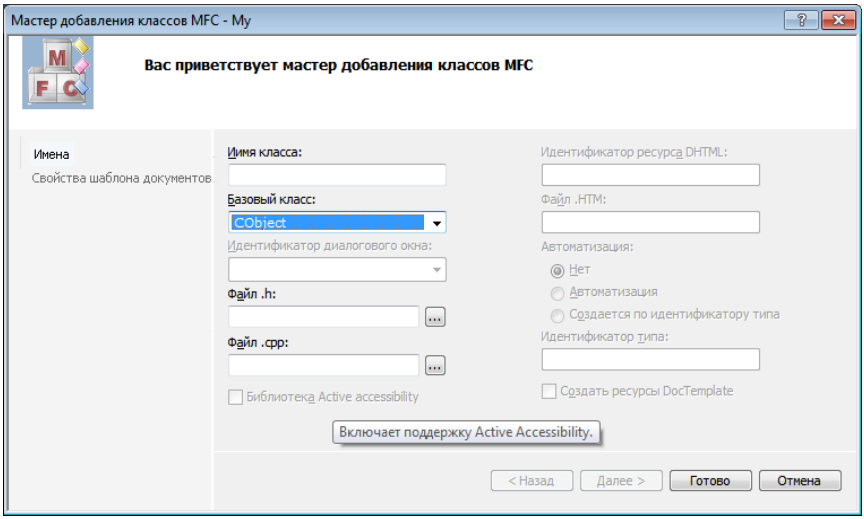
Створити однодокументну *Windows*-програму на *C++ISO|ANSI* з використанням бібліотеки *MFC*. Передбачити створення користувачем нестандартного, тобто власного класу. У ньому передбачити елемент даних типу *CString*. У свою чергу, внести об'єкт цього класу як елемент даних у клас документа. Забезпечити процес серіалізації документа. Тобто в конкретному випадку потрібно забезпечити читання рядка символів, запис її на диск і подальше читання з диска.

Загальні вказівки

Серіалізація – процес запису на диск або читання з нього об’єктів класу. У попередній лабораторній роботі була розглянута серіалізація об'єктів стандартних класів. У проекті, названому *My*, необхідно створити свій власний клас *CDog***,** похідний від класу *CObject*. У ньому є елемент даних *m_poroda* і методи: *vvod*() **–** введення даних, *vivod()* **–** виведення на екран, *clear()* **–** очищення екрана. Об’єкт класу *CDog* є елементом даних класу «Документ» (конкретно створеного майстром класу *CMyDoc*). Потрібно забезпечити процес занесення документа на диск і читання з нього.

1. Створюємо програму *My*. *Создать проект*, тип проекту *MFC*, шаблон – *Приложение MFC, Имя – My, Один документ.* Вимкнути *Использовать библиотеки с поддержкой Юникода*.

2. Додати в нього клас *Dog*. Для цього в меню *Проект* вибрати *Добавить класс*. У вікні, що відкриється, вибрати категорію *MFC* і шаблон *Класс MFC* . Клацнути по *Добавить*. Розкриється діалогове вікно. Ввести *Имя класса – CDog*. У рядку *Базовый класс* замість встановленого за замовчуванням базового класу *CWnd* встановити *CObject*. Слід зауважити, що одночасно майстром створюються файли *Dog.h* і *Dog.cpp* (без літери «С» попереду). Клацніть по *Готово.*



3. Внести в клас *CDog* елемент даних *m_poroda* і методи. Для цього вибрати вкладку *Окно классов*, розкрити папку *My*, виділити клас *CDog*, викликати контекстне меню і в ньому *Добавить/ Добавить переменную*. У вікні ввести *public: CString m_poroda*. Потім викликати *Добавить/Добавить функцию*, ввести методи класу *CDog:*

```
void vvod(CString  ch)
{
m_poroda+=ch;
}

void vivod (CDC*pDC)
{
pDC->TextOutA(0,10, m_poroda);
}

void clear()
{
m_poroda="";
}
```

4. Внести у клас *CMyDoc* елемент даних *public: CDog about_dog.* Потрібно виділити клас *CMyDoc* і ввести зазначені елементи.

5. Створити обробник *OnChar()* натисканням клавіші. Для цього виділити клас *CMyView.* У його *Свойствах* розкрити *Сообщения*. Зі списку, що розкриється, вибрати *WM_CHAR*, розкрити ще один список. У ньому буде *<Add>OnChar().* Клацнути на

цьому рядку. В результаті з’явиться заготовка обробника *OnChar().*

Оскільки потрібно ввести породу собаки, у функції *OnChar()* потрібно викликати метод *vvod()* класу *CDog*. Для цього потрібний об'єкт цього класу. Таким об'єктом є *about_dog*. Нагадаємо, що він є елементом даних класу *CMyDoc*. А отже, щоб його використовувати, потрібний покажчик *CMyDoc *pDoc*. Адреса в нього буде занесена за допомогою функції *GetDocument().* Текст функції *OnChar()* наведено нижче:

```
void CMyView::OnChar(UINT nChar, UINT nRepCnt,
                     UINT nFlags)
{
  CMyDoc*pDoc=GetDocument();
  ASSERT_VALID(pDoc);
  if(!pDoc)
   return;
  pDoc->about_dog.vvod(CString((char)nChar));
  Invalidate();                    // Не забувати!
  pDoc->SetModifiedFlag();


 CView::OnChar(nChar, nRepCnt, nFlags);
}
```

Щоб при виході з програми виникло запитання: зберігати нові дані чи ні, використовується метод *SetModifiedFlag().* Для виведення на екран використовується функція *OnDraw().* Виділимо клас

CMyView і в ньому - цей метод. Додамо до нього рядок:

```
 pDoc->about_dog.vivod(pDC);
```

Текст функції має вигляд

```
void CMyView::OnDraw(CDC*pDC /*pDC*/)
{
 CMyDoc* pDoc = GetDocument();
 ASSERT_VALID(pDoc);
 if (!pDoc)
  return;
 pDoc->about_dog.vivod(pDC);
// TODO: добавьте здесь код отрисовки для собственных данных
}
```

При створенні нового документа потрібно очистити екран. Для цього до заготовки методу OnNewDocument() класу CMyDoc потрібно додати два рядки, виділені жирним шрифтом:

```
BOOL CMy3Doc::OnNewDocument()
{
 if (!CDocument::OnNewDocument())
  return FALSE;
 about_dog.clear();
 UpdateAllViews(NULL);
 // TODO: добавьте код повторной инициализации


 return TRUE;
}
```

Перший із них – виклик функції *clear()* класу *CDog*. Метод *UpdateAllViews(NULL)* інформує всі види, пов'язані з документом, про те, що цей документ був модифікований.

Тепер безпосередньо займемося серіалізацією. Для цього спочатку потрібно забезпечити серіалізацію об'єктів класу *CDog*. А отже, потрібно до класу *CDog* додати метод *Serialize()*. З цією метою потрібно виділити клас *CDog*, викликати контекстне меню і в ньому – *Добавить/Добавить функцию*. У вікні, що відкрилося, ввести функцію *virtual void Serialize(CArchive & ar).* У ній серіалізувати елемент даних цього класу m_poroda. У результаті функція має вигляд

```
void CDog::Serialize(CArchive& ar)
{
 CObject::Serialize(ar);
 if(ar.IsStoring())
  ar<<m_poroda;
 else
  ar>>m_poroda;
}
```

Крім функції *Serialize(),* потрібно додати два макроси. В опис класу *CDog* (конкретно у файл *Dog.h*) необхідно вставити макрос *DECLARE_SERIAL(CDog).* Він може міститися де завгодно, але в межах опису класу, наприклад:

```
class CDog : public CObject
{
 DECLARE_SERIAL(CDog)
   // Тут крапка з комою не потрібні, оскільки
   // це виклик макроса, а не оператор C++
};
```

Аргументом макроса є ім'я створеного користувачем нестандартного класу *(CDog).* Макрос підтримує інформацію про цей клас під час виконання, а також забезпечує динамічне створення об'єктів і їх серіалізацію.

Ще один макрос *IMPLEMENT_SERIAL()* необхідно розмістити у файлі *Dog.cpp*. Він може міститися як на початку, так і в кінці файла. Найзручніше розміщувати його в першому рядку безпосередньо після директив *#include* препроцесора.

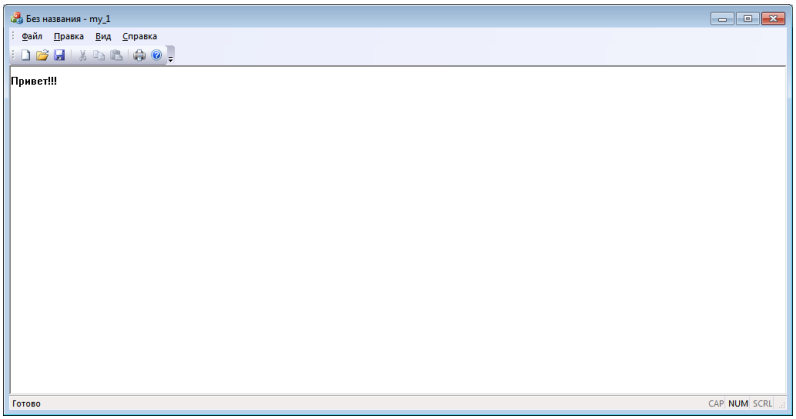
Макрос *IMPLEMENT_SERIAL(CDog, CObject, 0)* має три аргументи: ім'я класу, ім'я безпосереднього базового класу і беззнакове 32-розрядне ціле, яке ідентифікує номер схеми або номер версії вашої програми. Цей номер схеми дозволяє захистити процес серіалізації від проблем, які можуть виникнути, якщо ви пишете об'єкти в одній версії програми, а читаєте в іншій, в якій класи можуть відрізнятися. Якщо ви послідовно модифікуєте визначення класу, то повинні при цьому змінювати номер схеми. Якщо програма спробує прочитати дані,

які були записані з номером схеми, відмінним від номера поточної активної програми, то відбувається виключення.

Щоб виконати серіалізацію об'єкта *about_dog*, що є елементом даних класу *CMyDoc*, необхідно викликати його метод *Serialize()* усередині методу *Serialize()* документа *CMyDoc.*

```
void CMyDoc::Serialize(Carchive& ar)
{
 about_dog.Serialize(ar);
 /*if (ar.IsStoring())
 {
  // TODO: добавьте код сохранения

 }
 else
 {
  // TODO: добавьте код загрузки
} */
}
```

Необхідно зауважити, що клас *CObject* може бути непрямим базовим класом. Зокрема, це стосується ієрархії класів, коли для класу найвищого рівня *CObject* є безпосереднім базовим класом, а для інших – непрямим. Для коректної роботи серіалізації кожен клас в ієрархії повинен мати доданий макрос серіалізації в класі верхнього рівня.



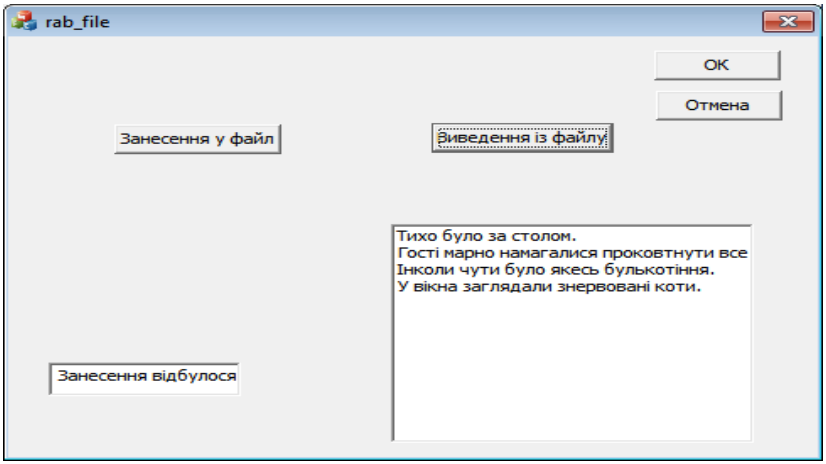
Контрольні запитання

1. Що таке серіалізація і навіщо вона потрібна?
2. Які макроси повинні бути додані до серіалізованого нестандартного класу і де вони повинні розміщуватися?
3. Який вигляд має функція *Serialize()* для нестандартного класу?
4. Що повинно бути у функції *Serialize()* класу «Документ»?
5. Що робить метод *Invalidate()*?
6. Навіщо потрібний метод *SetModifiedFlag()*?
7. Що відбувається при виконанні функції *UpdateAllViews(NULL)*?

Введення/виведення записів під час роботи з файлами з використанням бібліотеки MFC

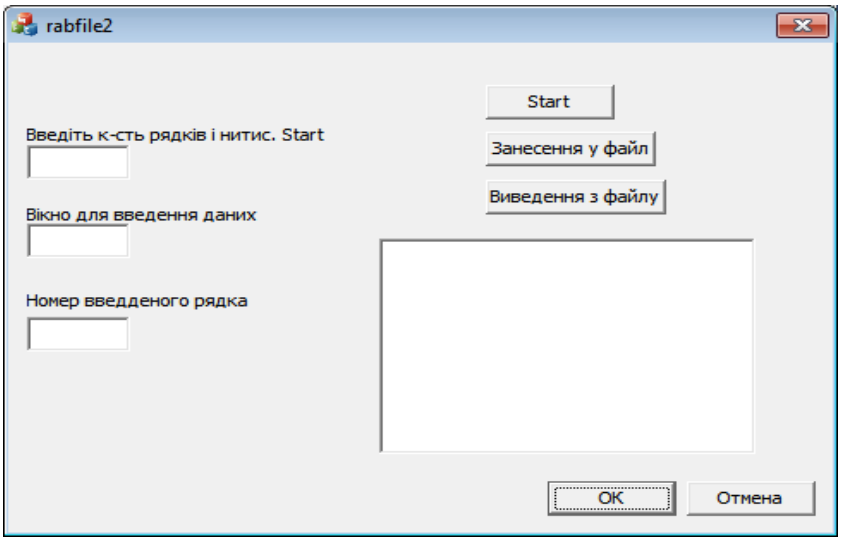
Постановка завдання

Необхідно створити дві діалогові програми для обробки записів. У першій із них передбачити ініціалізацію чотирьох рядків довжиною не більше 80 символів. У діалоговому вікні розмістити, крім *OK* і *Cancel,* дві кнопки: «Занесення до файла» і «Виведення із файла». Крім того, розмістити одне текстове вікно й один список. При натисканні першої кнопки в текстовому вікні повинне з'явитися повідомлення «Занесення відбулося». Натиснення другої кнопки повинне викликати появу у вікні списку чотирьох рядків після їх прочитання з файла.



У другій програмі замість ініціалізації передбачити спочатку занесення окремого рядка символів до текстового вікна, а потім його запис у файл шляхом натиснення кнопки «Занесення до файла». Кількість записуваних рядків попередньо заносити в спеціально призначене для цього текстове вікно. При записі у файл передбачити виведення в інше текстове вікно номера запису.

При натисненні кнопки «Виведення із файла» всі записи повинні бути виведені у вікно списку.



Послідовність дій при створенні першої програми

1. *Создать: Проект*. Вибрати тип проекту *MFC*, шаблон – *Приложение MFC , Имя-rabfile1*.

2. У вікні *Тип приложения* вибрати *Несколько документов/На основе диалоговых окон*. Зняти прапорець *Использовать библиотеки с поддержкой Юникода*. Натиснути на Г*отово.*

3. Вибрати вкладку *Окно ресурсов*. Розкрити папку *Dialog.* Двічі клацнути на *IDD_RABFILE1_DIALOG*. З'явиться діалогове вікно. Витерти в ньому текст *TODO ….*

4. Для виведення лише одного повідомлення досить використовувати текстове вікно. Проте для виведення чотирьох рядків потрібне уже вікно списку. Тому в діалоговому вікні *Панель элементов* необхідно розмістити керувальні елементи: текстове вікно (*ab| Edit Control*), вікно списку (*List Box*) і дві кнопки «Занесення до файла» і «Виведення із файла».

5. У клас *Crabfile1Dlg* додати елементи даних, що відповідають списку і текстовому вікну. Необхідно в діалоговому вікні виділити вікно списку, викликати контекстне меню, вибрати в ньому *Добавить переменную,* установити доступ *private*, тип за замовчуванням – *ClistBox,* ім'я дати *m_list1.* Для текстового вікна ввести *private: CEdit, m_edit1.*

6. Крім зазначених, до класу *Crabfile1Dlg* потрібно ще додати елементи даних: *char OutString[4][80]* (рядки, що заносяться до файла) і *char InString[80]* (рядок, в який послідовно здійснюватиметься зчитування з файла). Ці масиви не є керуювальними елементами. Тому їх додавання здійснюється як

завжди: у вкладці *Окно классов* виділяється *Crabfile1Dlg,* викликається контекстне меню, а в ньому – *Добавить/Добавить переменную*. Розмірність масивів задавати у віконці *Variable type*, наприклад, *char[4][80].* Доступ до цих масивів – *private.*

7. Ініціалізація даних діалогового вікна виконується в методі *OnInitDialog()* класу *Crabfile1Dlg*. Після занесення до масиву *OutString* чотирьох довільних символьних рядків фрагмент цього методу має вигляд

```
BOOL Crabfile1Dlg::OnInitDialog()
{
 CDialogEx::OnInitDialog();
 strcpy_s(OutString[0],"Тихо було за
                        столом.");
 strcpy_s(OutString[1],"Гості марно
    намагалися проковтнути все і відразу.");
 strcpy_s(OutString[2],"Інколи чути було
                       якесь булькотіння.");
 strcpy_s(OutString[3],"У вікна заглядали
                       знервовані коти.");

}
 return TRUE;
}
```

Необхідно звернути увагу, що для копіювання рядків символів використовується сучасніша функція *strcpy_s(),* хоча допускається вживання і *strcpy().*

А ще потрібно згадати про те, що рядки списку за замовчуванням сортуються в алфавітному порядку. Щоб записи, що виводяться з файла, розміщувалися у вікні списку в тій самій послідовності, в якій заносилися, сортування необхідно відключити. Для цього необхідно виділити вікно списку, викликати контекстне меню, вибрати команду *Свойства*, перейти на вкладку *Сортировка й* у вікні, що відкрилося, замість *True* встановити *False*.

8. Тепер необхідно створити обробник для кнопки «Занесення до файла». У програмі вона автоматично отримала символічне ім'я *IDC_BUTTON1*. У діалоговому вікні потрібно виділити кнопку, викликати контекстне меню і в ньому – *Добавить обработчик событий*. У вікні, що з'явилося, за замовчуванням у *Тип сообщений* вибране повідомлення *BN_CLICKED*, у *Список классов* – *Crabfile1Dlg*, а в *Имя функции-обработчика* – ім'я обробника. Залишається клацнути по кнопці *Добавить/Править.*

У результаті у файлі *Crabfile1Dlg.cpp* з'явиться заготовка для обробника події – клацання по кнопці «Занесення до файла». Після її заповнення отримаємо функцію-обробник, наведену нижче:

```
void Crabfile1Dlg::OnBnClickedButton1()
  {
  CFile to_file("f1.dat",
        CFile::modeCreate|CFile::modeWrite);
```

```
 for(int i=0; i<4; i++)
 {
   to_file.Seek(i*80,CFile::begin);
   to_file.Write(OutString[i],80);
 }
m_edit1.SetWindowTextA("Занесення відбулося");
// TODO: добавьте свой код обработчика уведомлений

 to_file.Close();                    // Необов'язково
}
```

Спочатку створюється об'єкт *to_file* класу *CFile*. Запрограмовано створення *(CFile::modeCreate)* файла *f1.dat* для запису в нього *(CFile::modeWrite).*

У циклі спочатку покажчик позиції файла встановлюється в позицію i*80 від його початку *(CFile::begin).* Потім відбувається запис у *OutString[i]* рядка завдовжки 80 символів. Після запису чотирьох рядків до текстового вікна, подане елементом даних *m_edit1* заноситься повідомлення «Занесення відбулося».

9. Створити обробник для кнопки «Виведення із файла». У програмі вона автоматично отримала символічне ім'я *IDC_BUTTON2*. Виконавши дії, наведені в попередньому пункті, отримаємо заготовку для обробника. Після внесення до неї необхідних операторів функція має вигляд

```
void Crabfile1Dlg::OnBnClickedButton2()
{
 CFile from_file("f1.dat",CFile::modeRead);
```

```
 for(int i=0; i<4; i++)
 {
   from_file.Seek(i*80, CFile::begin);
   from_file.Read(InString, 80);
   m_list1.AddString(InString);
 }
 // TODO: добавьте свой код обработчика уведомлений
}
```

Створюється об'єкт *from_file* класу *CFile*. При цьому існуючий (а отже, не потрібно писати *CFile::modeCreate*) файл *f1.dat* переводиться в режим лише для читання *(CFile::modeRead).*

У циклі покажчик позиції файла спочатку встановлюється в позицію *i*80* від його початку. Потім відбувається читання з файла в масив *InString* рядка завдовжки *80* символів. Після чого рядок заноситься до вікна списку, поданого елементом даних *m_list1.*

Перша програма складена і її можна запустити, натиснувши *Ctrl+F5.*

Послідовність дій при створенні другої програми

1. Виконати описані вище пункти 1–3 для створення діалогового вікна проекту, названого *rabfile2*. У ньому розмістити три текстові вікна. Одне – для попереднього введення кількості рядків, що заносяться до файла (ввести напис «Введіть кількість рядків і натисніть кнопку *«Start»*), друге – для послідовного введення символьних рядків (підписати «Вікно для введення даних») і третє – для

відображення поточного номера рядка, що вводиться. Крім того, ввести вікно списку для відображення виведених із файла рядків, а також три кнопки*: «Start», «Занесення до файла*» і «*Виведення з файла*».

2. До класу *Crabfile2Dlg* додати елементи даних з іменами *m_edit1, m_edit2 і m_edit3,* що відповідають текстовим вікнам, а також *m_list1*, що відповідає вікну списку.

3. Вибрати вкладку *Окно классов*, виділити *Crabfile2Dlg*, викликати контекстне меню *Добавить/Добавить переменную* та додати ряд змінних із доступом *private:*

- *char OutString[40][80]* – рядки, що заносяться до файла;
- *char InString[80]* – рядок, в який послідовно здійснюватиметься читання з файла;
- *int kil_poliv* – кількість записів (не більше 40, при роботі зі структурами це кількість її полів);
- *int nomer* – номер запису при занесенні до файла;
- *to_file і from_file* типу *CFile.*

Про останні дві змінні необхідно сказати окремо. Використовуваний файл, наприклад *f1.dat,* не можна відкривати в обробнику натисненням кнопки «*Занесення до файла*», інакше при кожному занесенні до файла він очищатиметься. Тому краще це зробити при натисненні кнопки *«Start».* Проте

використовується файл в обробниках кнопок «*Занесення до файла*», «*Виведення із файла*». А це означає, що змінна типу *CFile* не може бути локальною. Тому *to_file* і *from_file* додані до класу *Crabfile2Dlg* як елементи даних, а безпосередньо використовуваний файл буде відкритий в обробнику кнопки *«Start»* за допомогою функції *open()*. У принципі можна було б обійтися однією змінною типу *CFile*, але наявність двох ідентифікаторів *(to_file, from_file)* полегшує сприйняття коду програми.

4. Створити обробник для кнопки *«Start».* У ньому передбачити відкриття файла, обнулення номера запису *(nomer=0),* що заноситься до файла, і прочитання з текстового вікна (елемент даних *m_edit1*) заздалегідь записаної в нього кількості записів *(kil_poliv).* Нижче наводиться його код:

```
void Crabfile2Dlg:: OnBnClickedButton1()
{
to_file.Open("f1.dat",CFile::modeCreate|CFile:
  :modeWrite);
  nomer=0;
  CString s;
  m_edit1.GetWindowTextA(s);
  kil_poliv=atoi(s);
 // TODO: добавьте свой код обработчика уведомлений
}
```

За допомогою об'єкта *to_file* викликано метод *Open()* класу *CFile.* Створено файл *f1.dat* для запису в

нього. З текстового вікна, якому відповідає елемент даних *m_edit1*, читається рядок символів (кількість записів), який перетворюється на число *kil_poliv.*

5. Створити обробник для кнопки «*Занесення до файла*».

```
void Crabfile2Dlg:: OnBnClickedButton2()
{
  CString s;
  m_edit2.GetWindowTextA(s);
  strcpy_s(OutString[nomer],s);
   to_file.Seek(nomer*80,CFile::begin);
   to_file.Write(OutString[nomer],80);
   char str[10];
   _gcvt(nomer+1,3,str);
   m_edit3.SetWindowTextA(str);
   m_edit2.SetWindowTextA("");
  nomer++;
  if(nomer>=kil_poliv)
  {
   nomer=0;
m_edit2.SetWindowTextA("Занесення закінчилося");
  to_file.Close();
  }
 // TODO: добавьте свой код обработчика уведомлений
}
```

З текстового вікна, якому відповідає елемент даних *m_edit2*, читається рядок символів, який копіюється в масив *OutString[nomer].* Покажчик позиції файла встановлюється в позицію *nomer*80* від його початку, після чого *OutString[nomer]* заноситься

до файла. Збільшений на одиницю номер запису для зручності користувача за допомогою функції *_gcvt()* перетворюється на рядок символів і виводиться в третє текстове вікно *(m_edit3).* У друге текстове вікно заноситься пустий символ для його очищення. Після того як будуть введені всі рядки, обнулюється номер запису і в друге вікно виводиться повідомлення "Занесення закінчилося".

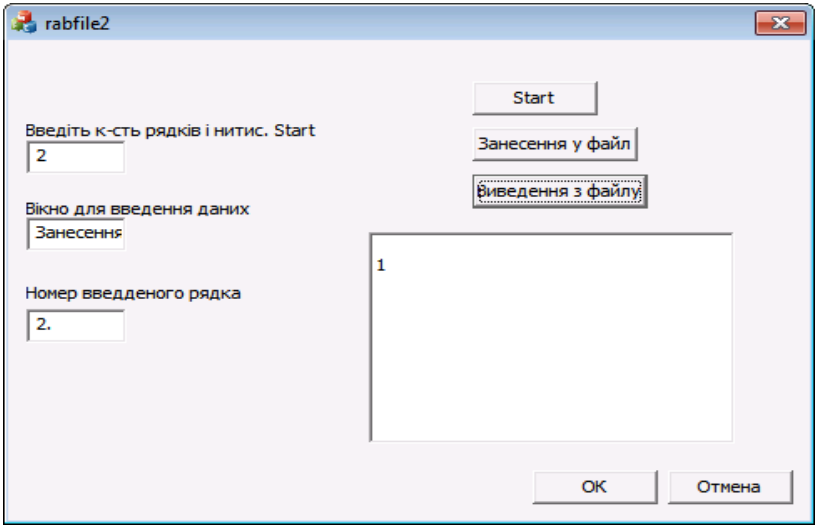
6. Створити обробник для кнопки «Виведення із файла».

```
void Crabfile2Dlg::OnBnClickedButton3()
{
 from_file.Open("f1.dat",CFile::modeRead);
 CString s;
 for(int i=0;i<kil_poliv;i++)
 {
  from_file.Seek(i*80,CFile:: begin);
  from_file.Read(InString,80);
  //strcpy(s,InString);

  m_list1.AddString(InString);
 }
 // TODO: добавьте свой код обработчика уведомлений
}
```

Викликається функція *Open().* Файл *f1.dat* тепер використовується для читання. У циклі покажчик позиції файла спочатку встановлюється в позицію *i*80* від його початку. Потім відбувається прочитання із файла в масив *InString* рядка завдовжки *80* символів.

Після чого рядок заноситься у вікно списку, подане елементом даних *m_list1*.

Друга програма також складена і її можна запустити, натиснувши *Ctrl+F5*.



Контрольні запитання

1. Як створити діалогове вікно?
2. Як створити файл і вибрати його режим роботи?
3. Як здійснити ініціалізацію рядків ?
4. Як занести до файла рядки, які ініціалізовані?
5. Як прочитати з файла послідовність рядків?
6. Яким чином можна забезпечити занесення до файла довільної кількості символьних рядків?
7. Як здійснюється читання/запис рядка символів під час роботи з текстовим вікном?

8. Як відключити сортування символьних рядків у вікні списку?

Створення простого графічного редактора

Постановка завдання

Створити *Windows*-програму для рисування типових графічних елементів: ліній, прямокутників, еліпсів, кіл, дуг із вибором пера і пензликів. Передбачити зафарбування або штрихування деяких фігур та інших замкнених областей. У меню *Tools* включити пункти рисування: *Draw freehand* (рисування довільної фігури), *Line, Rectangle, Ellipse, Fill figure* (зафарбовування замкненої області). До панелі інструментів додати відповідні до цих пунктів меню кнопки. Якщо вибраний пункт меню *Line*, то відповідна кнопка панелі інструментів повинна бути «втоплена». Якщо натиснути кнопку *Line*, то проти пункту меню *Line* повинна з'явитися «позначка». При рисуванні натиснення мишки визначає початкову точку, а при відпусканні – кінцеву. При відпусканні – рисується відповідна фігура.

Загальні вказівки

1. Створити однодокументну *(SDI)* програму *graf_painter. Для цього Создать проект*, тип проекту *MFC*, шаблон – *Приложение MFC, Имя – graf_painter,*

Один документ. Вимкнути *Использовать библиотеки с поддержкой Юникода*.

2. Кожній фігурі поставити у відповідність прапорець: *bool bDrawFlag*, *bool bLineFlag* і т. д. Для цього *Cgraf_painterViev|Добавить переменную|public, bool, bDrawFlag* і т. д. Для вибраної фігури прапорець з первинного *False* переводитиметься в *True*.

Крім прапорців, передбачити змінні:

```
CPoint Anchor;      // початкова точка;
CPoint DrawTo;      // кінцева точка;
CPoint OldPoint;    // попередня точка.
```

Ввести їх як елементи даних у клас *Cgraf_painterView* з доступом *protected*.

3. У клас *Cgraf_painterView* також додати метод *MakeAllFlagsFalse()*. У ньому всі п'ять прапорців необхідно установити в *False*. Цей метод викликати з конструктора класу

```
Cgraf_painterView:: Cgraf_painterView()
  {
  MakeAllFlagsFalse( );
  }
```

Це приводить до відключення всіх режимів рисування при запуску програми.

4. Викликати редактор меню і додати пункт меню *Tools*. Внести до нього п'ять підпунктів: *Draw freehand, Line, Rectangle, Ellipse, Fill figure*.

5. Додати до панелі інструментів відповідні до підпунктів меню кнопки.

6. Пов'язати кнопки з пунктами меню. Для цього необхідно, знаходячись в редакторі панелі інструментів, клацнути двічі по кнопці *Line*. У вікні *Свойства* вибрати вікно *ИД* і клацнути по ньому. Із списку, що розкриється, вибрати *ID_TOOLS_LINE*. Це ідентифікатор меню *Line*. Таким чином, кнопка *Line* зв'язується з відповідним пунктом меню. Нижче в полі *Подпись* необхідно ввести *Line*\n*Line*. Це означає, що коли користувач затримає покажчик мишки над даною кнопкою, у рядку стану з'явиться текст *Line*. Крім того, текст за символом \n буде виведений як екранна підказка біля кнопки.

Ці самі операції виконати і з іншими доданими кнопками.

7. Створити обробники для всіх п'яти підпунктів меню *Tools*. У кожному з них спочатку викликати метод *MakeAllFlagsFalse(),* а потім присвоїти відповідному прапорцю значення *True*. Нижче наводиться обробник для *Line*.

```
void Cgraf_painterView:: OnToolsLine()
   {
    MakeAllFlagsFalse();
    bLineFlag=true;
// TODO: добавьте свой код обработчика уведомлений
   }
```

Так здійснюється зв'язування прапорців із засобами призначеного для користувача інтерфейсу.

8. Позначити команди меню. Перед відображенням меню програма викликає для кожної команди спеціальний метод оновлення призначеного для користувача інтерфейсу, в якому можна встановити позначку для команди, що відповідає активному режиму рисування. Для цього потрібно виділити пункт меню, наприклад *Line*, викликати правою кнопкою контекстне меню, а в ньому – *Добавить обработчик событий*. У вікні, що з'явиться, вибирати не *COMMAND,* а *UPDATE_COMMAND_UI,* а у списку *Список классов – Cgraf_painterView.*

Увага! Спочатку вибирати клас, а потім команду. Інакше може мати місце самовільний перехід на *COMMAND*.

У кожен обробник включити метод *SetCheck* (*ім'я_прапорця*). Крім позначки команд активного режиму в меню *Tools*, ці методи змушують відповідну кнопку на панелі інструментів мати вигляд «натисненої».

Нижче наведений приклад для пункту меню *Line*.

```
void Cgraf_painterView::OnUpdateToolsLine(CCmdUI*pCmdUI)
{
 pCmdUI->SetCheck(bLineFlag);
```

```
}
```

9. Обробити повідомлення про натиснення кнопки мишки. Натиснення лівої кнопки миші задає початкову точку для рисування фігури. Створити обробник події натисненням цієї кнопки. Для цього вибрати *Cgraf_painterView,* викликати контекстне меню *Свойства/Сообщения.* У списку, що з'явиться, вибрати *WM_LBUTTONDOWN*. З'явиться *<Add>OnLButtonDown,* ще раз клацнути, що приведе до появи обробника.

Необхідно звернути увагу на те, що формальний параметр *CPoint point* - це координати поточної точки, над якою розміщена мишка. Залишилося лише надати ці координати точці *Anchor*, з якої почнеться рисування фігури.

```
void Cgraf_painterView::
  OnLButtonDown(UINT nFlags, CPoint point)
  {
  Anchor.x=point.x;
  Anchor.y=point.y;


  CView::OnLButtonDown(nFlags, point);
}
```

10. Запрограмувати рисування фігур. Рисування фігур відбувається при відпусканні лівої кнопки мишки.

Додати обробник *OnLButtonUp().* Це робиться як і в попередньому випадку.

```
void Cgraf_painterView::OnLButtonUp(UINT nFlags,
                                    CPoint point)
  {
  Cgraf_painterDoc*pDoc=GetDocument();
  ASSERT_VALID(pDoc);
  DrawTo.x=point.x;
  DrawTo.y=point.y;          // Задали кінцеву точку
  CClientDC*pDC=new CClientDC(this);
  /* Створюється контекст пристрою для виду в довільний
  момент. Це можна робити не лише в методі OnDraw().
  Клас CClientDC є похідним від класу CDC. Достатньо йому
  передати покажчик this на поточний об'єкт, щоб
  створити контекст пристрою. У ньому можемо
  рисувати  точно так, як і в методі OnDraw( ). */

  if ( bLineFlag)
  {
  pDC->MoveTo(Anchor.x, Anchor.y);
  // Курсор у початковій точці.

  pDC->LineTo(DrawTo.x, DrawTo.y);
  }
  if( bRectanglFlag)
  {
  pDC->MoveTo (Anchor.x, Anchor.y);
  pDC->Rectangle(Archor.x, Archor.y,
                 DrawTo.x, DrawTo.y );
  }
  if  (bEllipseFlag)
  {
  pDC->MoveTo  (Anchor.x, Anchor.y);
```

```
   pDC->Ellipse(Archor.x, Archor.y,
                DrawTo.x, DrawTo.y);
   }
   // Зафарбовування фігур. Використовується FloodFill( ) і
   // чорний пензлик.
   if ( bFillFlag)
   {
   pDC->SelectStockObject(BLACK_BRUSH);
   // Чорний пензлик.

   pDC->FloodFill(Anchor.x, Anchor.y, RGB(0,0,0));
   }
   delete pDC; // Видаляємо покажчик на поточний
               // вигляд.
   CView::OnLButtonUp(nFlags, point);
   // Обробник базового класу.
}
```

11. Запрограмувати рисування фігури довільної форми. При натисненні лівої кнопки мишки запам'ятати початкову точку. Коли мишка переміщається в нову точку, буде рисуватися лінія від початкової точки в поточну. Потім поточна точка робиться початковою для подальшого переміщення мишки. Таким чином, обробник переміщення миші спочатку має вигляд

```
void Cgraf_painterView::OnMouseMove(UINT nFlags,
                                    CPoint point)
{
   CClientDC*pDC=new CClientDC(this);
   if(nFlags && MK_LBUTTON && bDrawFlag)
```

```
  {
  pDC->MoveTo(Anchor.x,Anchor.y);
   pDC->LineTo(point.x,point.y);
  Anchor.x=point.x;
  Anchor.y=point.y;
  }
  delete pDC;


  CView::OnMouseMove(nFlags, point);
}
```

Необхідно звернути увагу, що рисування можливе при одночасному виконанні умов, зазначених у *if()*.

12. Змінити покажчик мишки. Для зручності рисування фігур довільної форми створимо новий покажчик у вигляді хрестика. Для цього змінимо вміст інформаційної структури *(CREATESTRUCT &cs)* вікна в методі *PreCreateWindow(CREATESTRUCT &cs),* наявному в класі виду. Задамо в ній новий покажчик у вигляді хрестика, *IDC_CROSS*.

```
BOOL Cgraf_painterView::PreCreateWindow(CREATESTRUCT&cs)
{
cs.lpszClass=AfxRegisterWndClass(CS_DBLCLKS,
  AfxGetApp()->LoadStandrdCursor (IDC_CROSS),
  (HBRUSH)(COLOR_WINDOW+1),
  AfxGetApp()->LoadIcon(IDR_MAINFRAME));
return CView::PreCreateWindow(cs);
}
```

13. Забезпечити розтягання графічних фігур. Код для імітації розтягання фігур внесений в *OnMouseMove().*

Почнемо з розтягання ліній. Щоб провести лінію від початкової точки до поточної, необхідно спочатку стерти старе її зображення. Але перш ніж стирати, потрібно запам'ятати положення стертої лінії. Для цього додамо до програми елемент даних класу *CPoint* з ім'ям *OldPoint*. У ньому зберігатиметься положення покажчика мишки при виклику *OnMouseMove().* При натисканні кнопки мишки необхідно занести в *OldPoint* поточні координати.

Тепер обробник натискання мишки має вигляд

```
void Cgraf_painterView::OnLButtonDown(UINT nFlags,
                                      CPoint point)
{
   Anchor.x=point.x;
   Anchor.y=point.y;
   OldPoint.x=Anchor.x;
   OldPoint.y=Anchor.y;
   //TODO: Добавьте код программы обработки
  // сообщения.

  CView:: OnLButtonDown(nFlags, point);
}
```

Майбутню роботу можна розділити на три етапи:

1. Стирання лінії від початкової точки *Anchor* до попередньої точки *OldPoint* (там, де раніше закінчувалася лінія).

2. Рисування нової лінії від *Anchor* до поточного положення курсора.

3. Копіювання координат поточної точки в *OldPoint* для подальшого переміщення мишки.

Усі ці етапи будуть реалізовані в обробнику *OnMouseMove().*

Щоб стерти лінію від *Anchor* до *OldPoint,* необхідно вибрати в контексті пристрою бінарний растровий режим *R2_NOT* і знову нарисувати ту саму лінію. Цей режим означає, що під час рисування кожного нового пікселя екранний піксель інвертується. Оскільки в програмі лінії мають чорний колір, а фон – білий, то після повторного прорисовування в цьому режимі лінія стане білою, тобто фактично виявиться стертою. Крім того, якщо розтягувана лінія пройде над чорною ділянкою, її відповідний відрізок стане білим, щоб його було видно. Перед тим як установити растровий режим *R2_NOT*, поточний режим запам'ятовується в локальній змінній *nOldMode*.

Аналогічний вигляд має код для розтягання прямокутників та еліпсів. Тепер обробник переміщення мишки має вигляд

```
void Cgraf_painterView::OnMouseMove(UINT nFlags,
                                    CPoint point)
{
  int nOldMode;        // Колишній растровий режим.
  CClientDC*pDC=new CClientDC(this);
```

```
 if(nFlags && MK_LBUTTON && bDrawFlag)
  {
   pDC->MoveTo(Anchor.x,Anchor.y);
   pDC->LineTo(point.x,point.y);
   Anchor.x=point.x;
   Anchor.y=point.y;
  }
 if(nFlags && MK_LBUTTON && bLineFlag)
    {
     nOldMode=pDC->GetROP2();
     // Запам'ятали поточний растровий режим.

     pDC->SetROP2(R2_NOT);
     /*Установлення нового режиму, коли колір пікселя є
     інвертованим кольором екрана. */

     pDC-> MoveTo(Anchor.x,Anchor.y);
     pDC-> LineTo(OldPoint.x,OldPoint.y);
     // Відбулося стирання.

     pDC->MoveTo(Anchor.x,Anchor.y);
     pDC-> LineTo(point.x,point.y);
     // Провели лінію в поточну точку.
    OldPoint.x=point.x;
    OldPoint.y=point.y;
    // Відновили OldPoint.

    pDC->SetROP2(nOldMode);
    // Відновили попередній режим.
     }
if(nFlags && K_LBUTTON && bRectangleFlag)
   {
    CClientDC*pDC=new CClientDC(this);
    nOldMode=pDC-> GetROP2();
```

```
    // Зам'ятали поточний растровий режим.

    pDC->SetROP2(R2_NOT);
    // Установлення нового режиму, коли колір пікселя є
    // інвертованим кольором екрана.

    pDC-> SelectStockObject(NULL_BRUSH);
    pDC->Rectangle(OldPoint.x,OldPoint.y,
                          Anchor.x,Anchor.y);
    pDC->Rectangle(Anchor.x,Anchor.y,
                             point.x,point.y);
     OldPoint.x=point.x;
     OldPoint.y=point.y;   // Відновили OldPoint.
     pDC->SetROP2(nOldMode);
     // Відновили попередній режим.
   }
if(nFlags && MK_LBUTTON && bEllipseFlag)
 {
    CClientDC*pDC=new CClientDC(this);
    nOldMode=pDC-> GetROP2();
    // Запам'ятали поточний растровий режим.

    pDC->SetROP2(R2_NOT);
    // Установлення нового режиму, коли колір пікселя є
    // інвертованим кольором екрана.

    pDC-> SelectStockObject(NULL_BRUSH);
    pDC->Ellipse(OldPoint.x,OldPoint.y,
                     Anchor.x,Anchor.y);
    pDC->Ellipse(Anchor.x,Anchor.y,
                     point.x,point.y);
      OldPoint.x=point.x;
      OldPoint.y=point.y;  // Відновили OldPoint.
    pDC->SetROP2(nOldMode);
```

```
        // Відновили попередній режим.
    }

    delete pDC;

    CView::OnMouseMove(nFlags,point);
}
```

14. Забезпечити відновлення зображення.

Як відомо, для цього використовується функція *OnDraw().* Вона оновлює зображення у вікні програми. Проте у складних програмах із написанням цієї функції виникають проблеми, тому що необхідно відтворити все зображення, яким би складним воно не було. Наприклад, користувач може нарисувати багато фігур різних типів. Щоб не запам'ятовувати кожне переміщення мишки, пропонується використовувати метафайли.

Метафайлом називають об'єкт, що зберігається в пам'яті та підтримує власний контекст пристрою. Будь-які операції з контекстом необхідно продублювати в метафайлі. Якщо пізніше потрібно буде повторити ці дії (при оновленні зображення), досить відтворити цей метафайл. У результаті всі графічні операції повторюються, що приводить до автоматичного створення потрібних зображень.

Існує клас *CMetaFileDC*. Його конструктор *CMetaFileDC()* конструює об'єкт цього класу.

Метод *Create()* створює контекст пристрою *Windows* для метафайла і зв'язує його з об'єктом *CMetaFileDC*. Після створення контексту пристрою для метафайла в ньому можна рисувати, дублюючи графічні операції у вікні. Коли потрібно відновити зображення, просто необхідно відтворити цей метафайл.

Метод *Close()* закриває контекст пристрою і (що дуже важливо!) створює логічний номер метафайла.

Метод *PlayMetaFile (int nomer)* відтворює метафайл за його логічним номером.

Отже, для оновлення зображення у вікні потрібно:

1. Внести до заголовного файла документа покажчик *pMetaFileDC* на контекст пристрою метафайла, який використовуватиметься в програмі. Для цього виділити клас *Cgraf_painterDoc*, викликати контекстне меню, вибрати *Добавить переменную*. Визначити тип змінної *CMetaFileDC** і ввести ім'я покажчика.
2. У конструктор *Cgraf_painterDoc()* додати виділення пам'яті і виклик методу *Create()* для створення нового об'єкта.

```
Cgraf_painterDoc::Cgraf_painterDoc():pMetaFileDC(NULL)
{
 pMetaFileDC=new CMetaFileDC();
 pMetaFileDC->Create();
 //TODO: добавьте код конструкции здесь
}
```

У результаті створено контекст пристрою для метафайла, в якому дублюватимуться всі дії користувача.

3. Додати до обробників подій натискання кнопки мишки і її переміщення дублювання графічних операцій в метафайлі. Все, що користувач рисує на екрані, повинно дублюватися в метафайлі. Це означає, *що при кожному виклику методу для контексту пристрою вигляду необхідно викликати аналогічний метод для контексту метафайла.*

У результаті обробник відпускання лівої кнопки мишки матиме вигляд

```
void Cgraf_painterView::OnLButtonUp(UINT nFlags,
                                    CPoint point)
{
  Cgraf_painterDoc*pDoc=GetDocument();
  ASSERT_VALID(pDoc);
  DrawTo.x=point.x;
  DrawTo.y=point.y;
  CClientDC*pDC=new CClientDC(this);
  if(bLineFlag)
  {
   pDC-> MoveTo(Anchor.x,Anchor.y);
   pDC-> LineTo(DrawTo.x,DrawTo.y);
   pDoc->pMetaFileDC->MoveTo(Anchor.x,Anchor.y);
   pDoc->pMetaFileDC->LineTo(DrawTo.x,DrawTo.y);
  }
  if(bRectangleFlag)
  {
  pDC-> MoveTo(Anchor.x,Anchor.y);
  pDC->Rectangle(Anchor.x,Anchor.y,
```

```
                                DrawTo.x,DrawTo.y);
    pDoc->pMetaFileDC->MoveTo(Anchor.x,Anchor.y);
  pDoc->pMetaFileDC->Rectangle(Anchor.x,Anchor.y,
                                DrawTo.x,DrawTo.y);
    }
    if(bEllipseFlag)
    {
    pDC->MoveTo(Anchor.x,Anchor.y);
    pDC->Ellipse(Anchor.x,Anchor.y,
                                DrawTo.x,DrawTo.y);
    pDoc->pMetaFileDC->MoveTo(Anchor.x,Anchor.y);
    pDoc->pMetaFileDC->Ellipse(Anchor.x,Anchor.y,
                                  DrawTo.x,DrawTo.y);
    }
    if(bFillFlag)
    {
     pDC->SelectStockObject(BLACK_BRUSH);
    pDC->FloodFill(Anchor.x,Anchor.y,RGB(0,0,0));
    pDoc->pMetaFileDC->SelectStockObject(BLACK_BRUSH);
    pDoc->pMetaFileDC->FloodFill(Anchor.x,Anchor.y,
                                          RGB(0,0,0));
    }

    delete pDC;   // Видаляємо покажчик на поточний
                  // вигляд.
    // TODO: Добавьте свой код программы
  // обработки сообщения здесь

    CView::OnLButtonUp(nFlags, point);
}
```

Необхідно звернути увагу на те, що показник *pMetaFileDC* є елементом даних класу *Cgraf_painterDoc*. Тому, щоб за його допомогою

викликати методи класу *CMetaFileDC*, аналогічні методам для контексту пристрою вигляду, описаний показник

```
Cgraf_painterDoc*pDoc=GetDocument();
```

Довільна фігура рисується при переміщенні мишки. Тому аналогічне дублювання потрібно додати до відповідного обробника.

Природно, що при розтяганні фігур необхідності в дублюванні немає.

```
void Cgraf_painterView::OnMouseMove(UINT nFlags,
                                    CPoint point)
{
  Cgraf_painterDoc*pDoc=GetDocument();
   ASSERT_VALID(pDoc);
   int nOldMode;   // Колишній растровий режим.
   CClientDC*pDC=new CClientDC(this);
   if(nFlags && MK_LBUTTON && bDrawFlag)
   {
    pDC->MoveTo(Anchor.x,Anchor.y);
    pDC->LineTo(point.x,point.y);
   pDoc->pMetaFileDC->MoveTo(Anchor.x,Anchor.y);
   pDoc->pMetaFileDC->LineTo(point.x,point.y);
    Anchor.x=point.x;
    Anchor.y=point.y;
   }
   if(nFlags && MK_LBUTTON && bLineFlag)
   {
    nOldMode=pDC-> GetROP2();
    // Запам'ятали поточний растровий режим.
```

```
   pDC->SetROP2(R2_NOT);
   // Установлення нового режиму, де колір пікселя є
   //  інвертованим кольором екрана.

   pDC-> MoveTo(Anchor.x,Anchor.y);
    pDC-> LineTo(OldPoint.x,OldPoint.y);
   // Відбулося стирання.

   pDC-> MoveTo(Anchor.x,Anchor.y);
   pDC-> LineTo(point.x,point.y);
// Провели лінію в поточну точку.

   OldPoint.x=point.x;
   OldPoint.y=point.y;       // Відновили OldPoint.
   pDC->SetROP2(nOldMode);
// Відновили попередній режим.
  }
  if(nFlags && MK_LBUTTON && bRectangleFlag)
  {
  CClientDC*pDC=new CClientDC(this);
  nOldMode=pDC->GetROP2();
  // Запам'ятали поточний растровий режим.
  pDC->SetROP2(R2_NOT);
  // Установлення нового режиму, де колір пікселя є
  //  інвертованим кольором екрана.
   pDC-> SelectStockObject(NULL_BRUSH);
   pDC->Rectangle(OldPoint.x,OldPoint.y,
                        Anchor.x,Anchor.y);
  // Відбулося стирання.
  pDC->Rectangle(Anchor.x,Anchor.y,
                    point.x,point.y);
   OldPoint.x=point.x;
   OldPoint.y=point.y;
```

```
    // Відновили OldPoint.
     pDC->SetROP2(nOldMode);
    // Відновили попередній режим.
    }
    if(nFlags && MK_LBUTTON && bEllipseFlag)
    {
     CClientDC*pDC=new CClientDC(this);
     nOldMode=pDC->GetROP2();
    // Запам'ятали поточний растровий режим.

    pDC->SetROP2(R2_NOT);
    // Установлення нового режиму, де колір пікселя є
    // інвертованим кольором екрана.

     pDC-> SelectStockObject(NULL_BRUSH);
     pDC-> Ellipse(OldPoint.x,OldPoint.y,
                        Anchor.x,Anchor.y);
  // Відбулося стирання.

 pDC->Ellipse(Anchor.x,Anchor.y,point.x,point.y);
     OldPoint.x=point.x;
     OldPoint.y=point.y;   // Відновили OldPoint.
     pDC->SetROP2(nOldMode);
    // Відновили попередній режим.
    }
    delete pDC;
```

Отже, в метафайлі міститься повний запис зображення і його можна використовувати для відновлення зображення на екрані.

4. Запрограмувати у функції *OnDraw()* відновлення зображення. Фактично завдання зводиться до відтворення метафайла.

```
void Cgraf_painterView::OnDraw(CDC*pDC/*pDC*/)
 {
  Cgraf_painterDoc*pDoc = GetDocument();
  ASSERT_VALID(pDoc);
  if (!pDoc)
   return;
 HMETAFILE Nomer=pDoc->pMetaFileDC->Close();
  // Закрили і дізналися номер метафайла. Тепер у нього
  // записувати не можна.

  pDoc->pMetaFileDC-> PlayMetaFile(Nomer);
  // Відтворили його за номером.

  CMetaFileDC*New_metafile=newCMetaFileDC();
  // Створюємо покажчик на новий метафайл і виділяємо
  // йому пам'ять.

  New_metafile->Create();
  // Створюємо новий метафайл.

 New_metafile->PlayMetaFile(Nomer);
  /* Відтворюємо його за відомим номером. Тепер старий
метафайл можна замінити новим, а старий видалити. */

  DeleteMetaFile(Nomer);
  delete pDoc->pMetaFileDC;
  pDoc->pMetaFileDC=New_metafile;
 }
```

Спочатку необхідно закрити метафайл, щоб дізнатися його логічний номер. Він використовуватиметься надалі. Потім відтворити метафайл за отриманим номером, оскільки в ньому міститься необхідне зображення. Після закриття метафайла уже не зможемо записувати в нього. Щоб при подальшому відновленні не втратити все, що було нарисоване після закриття, необхідно створити новий метафайл і записати в нього старий (для нього уже є номер). Після цього можна старий метафайл замінити новим і видалити старий. У результаті відтворили графічне зображення з клієнтської області і створили новий метафайл для подальшого відновлення.

Примітка. Якщо ви використовуєте графічні методи, що повертають інформацію про контекст пристрою (наприклад, відомості про висоту або ширину), то відтворення метафайла в іншому контексті може бути невдалим (оскільки розміри необов'язково залишаться колишніми). Щоб пов'язати метафайл з контекстом пристрою того типу, в якому він повинен відтворюватися, потрібно використовувати метод *SetAttribDC()* класу *CMetaFileDC*.

Зображення, що міститься в метафайлі, можна зберегти на диску.

15. Для збереження графічних файлів використовується метод *CopyMetaFile()* класу *CMetaFileDC.*

Необхідно поєднати обробники з трьома командами меню *Файл: Новый, Сохранить* і *Открыть*. Вони створюватимуть новий документ програми *graf_painter*, зберігатимуть зображення у файлі *graf_painter.wmf* (це стандартне розширення для метафайлів) і завантажуватимуть збережений раніше метафайл.

1. Відкрити метод *OnFileSave()* і заповнити його.

```
void Cgraf_painterView::OnFileSave()
 {
  Cgraf_painterDoc*pDoc=GetDocument();
  ASSERT_VALID(pDoc);
 HMETAFILE Nomer=pDoc->pMetaFileDC->Close();
  CopyMetaFile(Nomer,"graf_painter.wmf");
  CMetaFileDC*Newmetafile=new CMetaFileDC();
  Newmetafile->Create();
  Newmetafile->PlayMetaFile(Nomer);
  DeleteMetaFile(Nomer);
  delete pDoc->pMetaFileDC;
  pDoc->pMetaFileDC=Newmetafile;
 }
```

Як і у попередньому випадку, спочатку закрили метафайл, щоб дізнатися його логічний номер. Потім скопіювали його на диск у файл *graf_painter.wmf.* Далі створюється новий метафайл і відтворюється в

ньому старий. Замінюється старий метафайл новим, після чого старий видаляється.

2. Заповнити обробник, що забезпечує завантаження графічного файла. Використовуватиметься метод *GetMetaFile().*

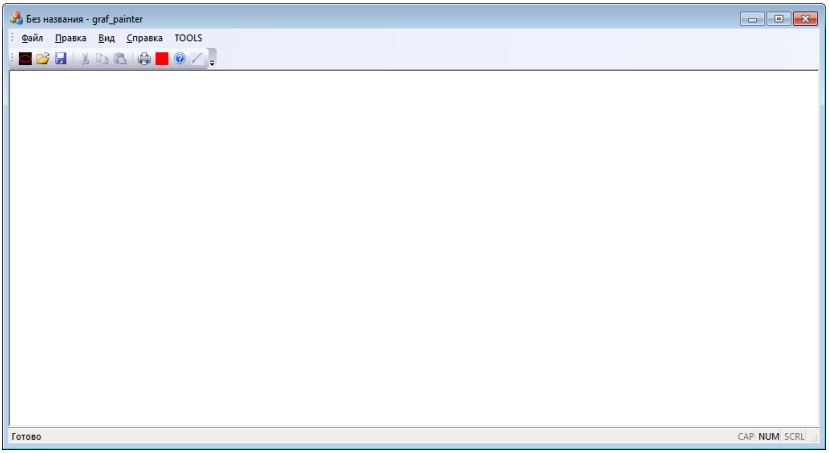
```
void Cgraf_painterView:: OnFileOpen()
 {
   Cgraf_painterDoc*pDoc=GetDocument();
   ASSERT_VALID(pDoc);
 HMETAFILE Nomer=GetMetaFile("graf_painter.wmf");
  // Створюється новий логічний номер метафайла.

  CMetaFileDC*Newmetafile=new CMetaFileDC();
   Newmetafile->Create();
   Newmetafile-> PlayMetaFile(Nomer);
   DeleteMetaFile(Nomer);
   delete pDoc->pMetaFileDC;
   pDoc->pMetaFileDC=Newmetafile;
  Invalidate();
  // Забезпечує відображення нового метафайла в об'єкті
  // вигляду програми.

  }
```

Як і раніше, за номером створюємо новий об'єкт класу *CMetaFileDC*, відтворюємо в ньому файл, завантажений з диска, замінюємо старий метафайл новим. Метод *Invalidate()* забезпечує відображення нового метафайла в об'єкті виду.

3. Заповнити обробник, що створює новий документ.

Вибираючи команду *Файл/ Новый,* користувач хоче створити новий документ. Тому створюємо порожній метафайл, установлюємо його у своїй програмі і викликаємо *Invalidate()* для його відтворення. Метод *Invalidate()* очищує вікно виду, і користувач може приступити до створення нового зображення в порожньому вікні і в порожньому метафайлі.

```
void Cgraf_painterView:: OnFileNew()
 {
   Cgraf_painterDoc*pDoc=GetDocument();
   ASSERT_VALID(pDoc);
   if (!pDoc)
    return;
   CMetaFileDC*nomer=new CMetaFileDC();
   nomer->Create();
   delete pDoc->pMetaFileDC;
   pDoc->pMetaFileDC=nomer;
   Invalidate();
 }
```



Контрольні запитання

1. Як розтягнути лінію?
2. Що таке бінарні растрові операції?
3. Що таке метафайл і навіщо він потрібний?
4. Як відтворити метафайл?
5. Як зберегти графічний файл?
6. Як відкрити графічний файл?
7. Як нарисувати прямокутник?
8. Як створити новий документ?

Діагностичні засоби MFC

Бібліотека *MFC* надає програмісту потужний набір засобів для налагодження додатків будь-якої складності. Розроблення додатків завжди передбачає написання систем, що працюють стабільно та перевірки вхідних параметрів функцій, вертаних значень, отриманих показників і т. ін. Але за стабільність доводиться платити уповільненням роботи програми. *MFC* пропонує такий підхід до проблеми: розробник вставляє в код набір *діагностичних макровизначень*, які при невиконанні заданих умов повідомляють ім'я вихідного файла з помилкою, номер рядка і зупиняють роботу програми. При цьому дані макровизначення виконуються тільки при налагоджувальному складанні проекту (*Debug build*). Іншими словами, у

код поміщаються перевірки, що виконуються лише в налагоджувальній версії програми і не включаються в код при остаточному складанні *(Release build).* За час роботи з налагоджувальною версією програми з'ясовуються і усуваються всі можливі помилкові ситуації.

Розглянемо декілька діагностичних макровизначень.

ASSERT і VERIFY

ASSERT – один із найбільш часто вживаних макросів. Беручи за аргумент булево значення, *ASSERT* продовжує роботу програми, якщо це значення дорівнює *TRUE*, і перериває роботу програми в іншому випадку. При цьому *ASSERT* виводить інформаційне вікно з ім'ям вихідного файла і номером рядка, що містить спрацьований макрос, і надає розробникові вибір – остаточно перервати роботу програми (*Abort*), перемкнутися у вікно налагоджувача (*Retry)* або продовжити роботу (*Ignore*).

Як приклад використання *ASSERT* можна навести перевірку вхідного значення функції:

```
void CPerson::SetPersonAge(int nAge)
{
ASSERT((nAge>=0)&&(nAge<200));    /*спрацює
при любих x<0 або x>199 */
    m_nAge = x;
```

```
};
```

При спрацьовуванні макроса (тобто при передаванні неправильного *nAge)* у розробника є можливість перемкнутися у вікно налагоджувача і через список викликів (*Call Stack*) визначити, звідки був переданий помилковий параметр.

ASSERT розгорнеться в код лише при *Debug-складанні*. Щоб забезпечити обчислення параметра і в остаточній версії проекту (у разі коли в *ASSERT* викликається потрібна функція), використовуйте макровизначення *VERIFY*. При *Debug-складанні* цей макрос повністю ідентичний *ASSERT*, але на відміну від нього при *Release-складанні VERIFY* розгортається в код і обчислює значення свого аргументу, хоча при цьому ніяк не впливає на хід виконання програми.

ASSERT_KINDOF і ASSERT_VALID

Ці макроси призначені для діагностики стану об'єктів. *ASSERT_KINDOF* набуває два параметри – ім'я класу і покажчик на об'єкт та спрацьовує (перериваючи виконання програми подібно *ASSERT*) у разі, коли об'єкт, переданий за вказівником, не є об'єктом цього класу або одного з нащадків даного класу. Приклад використання макроса

```
CPerson::CPerson(CPerson &newPerson)
```

```
{
   ASSERT_KINDOF(CPerson, &newPerson);
/*спрацює, якщо в конструктор
 було передано об’єкт не того класу */

};
```

ASSERT_KINDOF повністю ідентичний такій конструкції:

```
ASSERT(newPerson.IsKindOf(RUNTIME_CLASS(CPerson)));
```

Щоб отримати інформацію про клас у процесі виконання, цей клас повинен бути успадкований від *CObject* (або одного з його нащадків), і для нього повинні бути використані макроси *DECLARE_DYNAMIC (classname)* і *IMPLEMENT_DYNAMIC (classname, baseclass)* (інакше звернення до *ASSERT_KINDOF* призведе до помилки порушення захисту). Це стосується і об'єкта, що перевіряється, і класу.

ASSERT_VALID служить для перевірки внутрішнього стану об'єктів. Цей макрос набуває один параметр – покажчик на об'єкт, що перевіряється, а також перевіряє валідність покажчика, перевіряє його на рівність *NULL* і викликає функцію об'єкта *AssertValid*.

AssertValid реалізована майже в усіх класах *MFC* (успадкованих від *CObject*), але розробник може реалізувати її і в своєму класі, дотримуючись певних

правил. По-перше, *AssertValid* повинна бути перевизначеною віртуальною функцією класу *CObject*. Ця функція описана як *const*, тому всередині неї не можна змінювати дані класу. По-друге, для індикації факту невалідности об'єкта функція повинна використовувати макрос *ASSERT*. І по-третє, в *AssertValid* бажано викликати цю саму функцію класу-батька.

Таким чином, розробник може використовувати *ASSERT_VALID* для реалізації будь-яких алгоритмів перевірки стану об'єкта. Наприклад, ось так:

```
void CPerson::AssertValid() const
{
   CObject::AssertValid();
// тут CPerson успадкований від CObject
   ASSERT((m_nAge>=0) && (m_nAge<200));
};
```

При цьому *ASSERT_KINDOF* і *ASSERT_VALID* розгорнуться в код лише при *Debug-складанні*.

Робота з налагоджувальною інформацією

При налагодженні додатків у розробника часто виникає необхідність дізнатися значення будь-якої змінної або результат, повернений функцією. Для цього використовують або покрокове виконання ділянки коду з переглядом змінних у *Watch-вікні*

налагоджувача, або виведення інформації за допомогою інформаційних вікон (функції *MessageBox* і *AfxMessageBox*)

У Windows передбачений ще один спосіб отримання інформації від програми під час її виконання – функція *OutputDebugString*. Функція приймає *LPCTSTR-рядок* і посилає його налагоджувачу, під керуванням якого виконується додаток. У разі запуску програми з *Visual C++* посланий рядок потрапляє у вікно *Output*. Перевага цього способу в тому , що він не вимагає переривати роботу програми для відстеження значень і не вимагає видаляти зайвий код після налагодження потрібної ділянки (на відміну від методу з *MessageBox*).

OutputDebugString можна використовувати так:

```
void CPerson::SetPersonAge(int nAge)
{
   ASSERT((nAge>=0) && (nAge<200));
   m_nAge = x;
   CString str;
   str.Format(_T("New age = %d\n"),nAge);
   OutputDebugString(str);
};
```

MFC спрощує виведення налагоджувальної інформації, визначаючи глобальний об'єкт *afxDump*

класу *CDumpContext*. Інформація при цьому виводиться так:

```
void CPerson::SetPersonAge(int nAge)
{
  ASSERT((nAge>=0) && (nAge<200));
  m_nAge = x;
 afxDump<<_T("New age=")<<nAge<<_T("\n");
};
```

Необхідно пам’ятати, що виведення налагоджувальної інформації через *afxDump* працює лише у *Debug-версії* програми.

Виведення інформації про внутрішній стан об'єктів

Об'єкт *afxDump* дозволяє виводити у вікні налагоджувача не лише змінні, а й цілі об'єкти (породжені від *CObject*). Конструкція

```
afxDump << & myPerson;
```

розгорнеться у виклик

```
myPerson.Dump (afxDump);
//віртуальна функція класу CObject
```

Розробляючи власний клас, програміст може перевизначити цю функцію, реалізувавши свій метод виведення внутрішньої інформації об'єкта, наприклад, так:

```
// CPerson успадковано від CObject
void CPerson::Dump(CDumpContext &dc)const
{
   CObject::Dump(dc);
   dc << T("Age = ") << m_nAge;
};
```

Виклик батьківської функції *CObject :: Dump (dc)* виведе на контекст ім'я класу у разі, якщо для реалізації цього класу використовується зв'язка макросів *DECLARE_DYNAMIC / IMPLEMENT_DYNAMIC* (або *DECLARE_SERIAL / IMPLEMENT_SERIAL*).

Необхідно пам’ятати, що виведення відлагодженої інформації про об'єкт через функцію *Dump* буде працювати лише у *Debug-версії* програми, але тут розробник повинен сам подбати про виконання цього обмеження – і оголошення, і реалізацію, і виклики функції *Dump* потрібно обрамляти перевірками на *Debug-складання* проекту:

```
class CPerson : public CObject
{
   . . .
public:
#ifdef _DEBUG
   void Dump(dc) const;
   void AssertValid() const;
/*це саме стосується оголошення
AssertValid, але не використання макроса
ASSERT_VALID */
```

```
#endif
    . . .
};
```

Після виконання усіх цих дій залишається лише скинути в *afxDump* покажчик на наш об'єкт і вивчати отриману інформацію в *Output-вікні* налагоджувача.

Більш детальну інформацію з даної тематики можна знайти в *MSDN* або вихідних кодах прикладів самої *MFC*.

Список літератури

1. Хортон Айвор. VISUAL C++ 2010: полный курс / Айвор Хортон. – М. : Вильямс, 2011.
2. Зубенко В. В. Програмування. Поглиблений курс / В. В. Зубенко, Л. Л. Омельчук. – К. : ВПЦ "Київ. ун-т", 2011.
3. Майо Дж. Самоучитель Microsoft Visual Studio 2010 / Дж. Майо.– СПб. : БХВ-Петербург, 2011.
4. Культин Н. Б. Основы программирования в Microsoft Visual C++2010 / Н. Б. Культин. – СПб. : БХВ-Петербург, 2010.
5. Visual Studio 2010 для профессионалов / Н. Рендольф, Д. Гарднер, К. Андерсон, М. Минутилло. – М. : Диалектика, 2011.
6. Холзнер С. Visual C++ 6 / С. Холзнер. – Санкт-Петербург; Москва; Харьков; Минск : Питер, 2001.
7. Авраменко В. В. Програмування на Visual C++ 2005 із застосуванням MFC. Практикум / В. В. Авраменко, А. М. Скаковська. – Суми : СумДУ, 2011.

Предметний покажчик

А

Д

К

М

О

П

Е

К

М