Лабораторна робота №3

Розробка інтерфейсу користувача на C++

Мета: Мета роботи – отримати вміння та навички використовувати інкапсуляцію, абстракцію типів, успадкування та поліморфізм на основі класів C++, запрограмувавши графічний інтерфейс користувача.

Завдання:

1. Створити у середовищі MS Visual Studio C++ проект Win32 з ім'ям **Lab3**.
2. Написати вихідний текст програми згідно варіанту завдання.
3. Скомпілювати вихідний текст і отримати виконуваний файл програми.
4. Перевірити роботу програми. Налагодити програму.
5. Проаналізувати та прокоментувати результати та вихідний текст програми.
6. Оформити звіт.

Методичні рекомендації

Створення панелі інструментів (Toolbar)

Для створення Toolbar зручно використати бібліотеку **COMCTL32** елементів керування загального користування (***Common Control Library***).

Для того, щоб використати якійсь елемент з цієї бібліотеки, потрібно включити заголовочний файл

```
#include <commctrl.h>
```

Крім того, лінкеру потрібно посилання на статичну бібліотеку **COMCTL32.** Це можна зробити так:

```
#pragma comment(lib, "comctl32.lib")
```

При роботі програми потрібно спочатку викликати функцію

```
InitCommonControls();
```

потім можна створювати Toolbar потрібного вигляду.

Створити дочірнє вікно Toolbar з 4 кнопками можна шляхом використання функції **CreateToolbarEx**, наприклад, так:

```
TBBUTTON tbb[4];                              //масив опису кнопок вікна Toolbar

ZeroMemory(tbb, sizeof(tbb));
tbb[0].iBitmap = STD_FILENEW;                 //стандартне зображення
tbb[0].fsState = TBSTATE_ENABLED;
tbb[0].fsStyle = TBSTYLE_BUTTON;              //тип елементу - кнопка
tbb[0].idCommand = ID_TOOL_FILE_NEW;          //цей ID буде у повідомленні WM_COMMAND

tbb[1].iBitmap = STD_FILEOPEN;
tbb[1].fsState = TBSTATE_ENABLED;
tbb[1].fsStyle = TBSTYLE_BUTTON;
tbb[1].idCommand = ID_TOOL_FILE_OPEN;

tbb[2].iBitmap = STD_FILESAVE;
tbb[2].fsState = TBSTATE_ENABLED;
tbb[2].fsStyle = TBSTYLE_BUTTON;
tbb[2].idCommand = ID_TOOL_FILE_SAVEAS;

tbb[3].iBitmap = STD_PRINT;
tbb[3].fsState = TBSTATE_ENABLED;
tbb[3].fsStyle = TBSTYLE_BUTTON;
tbb[3].idCommand = ID_TOOL_FILE_PRINT;

hwndToolBar = CreateToolbarEx(hWndParent,       //батьківське вікно
                              WS_CHILD | WS_VISIBLE | WS_BORDER
                              | WS_CLIPSIBLINGS | CCS_TOP,
                              IDC_MY_TOOLBAR,   //ID дочірнього вікна Toolbar
                              1, HINST_COMMCTRL, IDB_STD_SMALL_COLOR,
                              tbb,
                              4,                //кількість кнопок
                              0,0,0,0,          //розташування та розміри
                              sizeof(TBBUTTON));
```

Дочірнє вікно Toolbar розташовується на поверхні якогось вікна, HWND якого вказується параметром **hWndParent.**

Зазвичай Toolbar у якості дочірнього вікна розташовується на головному вікні програми.

Створити дочірнє вікно Toolbar можна також і викликом функції **CreateWindow**

```
hwndToolBar = CreateWindow(TOOLBARCLASSNAME,        //ім'я класу вікна
                           NULL,
                           WS_CHILD | WS_VISIBLE | WS_BORDER
                           | WS_CLIPSIBLINGS | CCS_TOP,
                           0,0,0,0,              //розташування та розміри
                           hWndParent,              //батьківське вікно
                           (HMENU)IDC_MY_TOOLBAR,   //ID дочірнього вікна
                           hInst,
                           0);
```

або **CreateWindowEx**

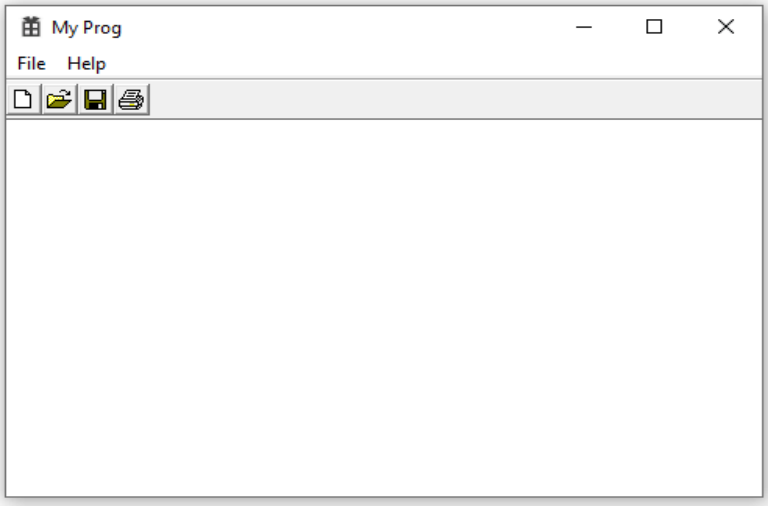
```
hwndToolBar = CreateWindowEx(0,
                             TOOLBARCLASSNAME,         //ім'я класу вікна
                             NULL,
                             WS_CHILD | WS_VISIBLE | WS_BORDER
                             | WS_CLIPSIBLINGS | CCS_TOP,
                             0,0,0,0,              //розташування та розміри
                             hWndParent,               //батьківське вікно
                             (HMENU)IDC_MY_TOOLBAR,    //ID дочірнього вікна
                             hInst,
                             0);
```

Тут у якості імені класу використана символічна константа **TOOLBARCLASSNAME**, замість якої автоматично підставляється **"ToolbarWindow32"**

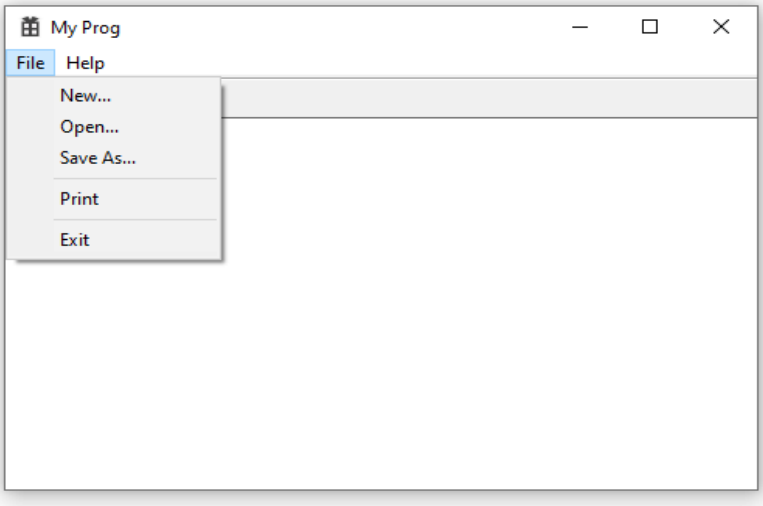
Якщо Toolbar створювався викликом функції **CreateWindow** або **CreateWindowEx,** то спочатку буде порожній Toolbar. Далі потрібно у нього вставити потрібні елементи – це можна зробити так:

```
hwndToolBar = CreateWindow...

SendMessage(hwndToolBar, TB_BUTTONSTRUCTSIZE, (WPARAM)sizeof(TBBUTTON), 0);

TBBUTTON tbb[4];
TBADDBITMAP tbab;

tbab.hInst = HINST_COMMCTRL;
tbab.nID = IDB_STD_SMALL_COLOR;
SendMessage(hwndToolBar, TB_ADDBITMAP, 0, (LPARAM)&tbab);

ZeroMemory(tbb, sizeof(tbb));
tbb[0].iBitmap = STD_FILENEW;            //стандартне зображення
tbb[0].fsState = TBSTATE_ENABLED;
tbb[0].fsStyle = TBSTYLE_BUTTON;         //тип елементу - кнопка
tbb[0].idCommand = ID_TOOL_FILE_NEW;     //цей ID буде у повідомленні WM_COMMAND

tbb[1].iBitmap = STD_FILEOPEN;
tbb[1].fsState = TBSTATE_ENABLED;
tbb[1].fsStyle = TBSTYLE_BUTTON;
tbb[1].idCommand = ID_TOOL_FILE_OPEN;

tbb[2].iBitmap = STD_FILESAVE;
tbb[2].fsState = TBSTATE_ENABLED;
tbb[2].fsStyle = TBSTYLE_BUTTON;
tbb[2].idCommand = ID_TOOL_FILE_SAVEAS;

tbb[3].iBitmap = STD_PRINT;
tbb[3].fsState = TBSTATE_ENABLED;
tbb[3].fsStyle = TBSTYLE_BUTTON;
tbb[3].idCommand = ID_TOOL_FILE_PRINT;

SendMessage(hwndToolBar, TB_ADDBUTTONS, 4, (LPARAM)&tbb);
```

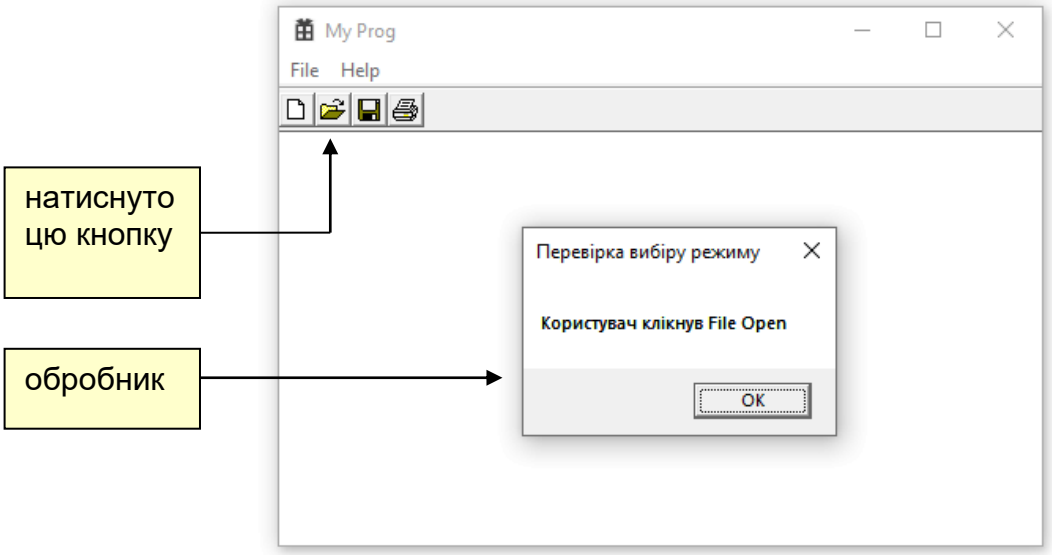
У результаті отримаємо такий Toolbar



Кнопки можуть дублювати деякі пункти меню



При натискуванні на кнопку повинна викликатися відповідна функція-обробник такої події (а точніше кажучі, обробник відповідного повідомлення)



Скелет програми

```
HWND hwndToolBar = NULL;
. . .
```

```
int APIENTRY _tWinMain(. . .)
{
InitCommonControls();
. . .
}
```

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
switch (message)
   {
   case WM_CREATE:
       OnCreate(hWnd);          //тут створимо Toolbar
       break;

   case WM_COMMAND:
       switch (wmId)
          {
          case IDM_NEW:
          case ID_TOOL_FILE_NEW:
              OnFileNew(hWnd);
              break;

          case IDM_OPEN:               //ID пункту меню
          case ID_TOOL_FILE_OPEN:      //ID кнопки Toolbar
              OnFileOpen(hWnd);        //функція-обробник
              break;

          case IDM_SAVEAS:
          case ID_TOOL_FILE_SAVEAS:
              OnFileSaveAs(hWnd);
              break;

          . . .
```

```
void OnCreate(HWND hWnd)
{
TBBUTTON tbb[4];      //для Toolbar з чотирма кнопками
ZeroMemory(tbb, sizeof(tbb));
tbb[0].iBitmap = ...
. . .
hwndToolBar = CreateToolbarEx(hWnd,              //батьківське вікно
                              WS_CHILD | WS_VISIBLE | WS_BORDER | WS_CLIPSIBLINGS | CCS_TOP,
                              IDC_MY_TOOLBAR,    //ID дочірнього вікна Toolbar
                              1, HINST_COMMCTRL, IDB_STD_SMALL_COLOR,
                              tbb,               //масив опису кнопок
                              4,                 //кількість кнопок
                              0,0,0,0,           //розташування та розміри
                              sizeof(TBBUTTON));
}
```

```
void OnFileNew(HWND hWnd)
{
   . . .
}
```

```
void OnFileOpen(HWND hWnd)
{
   MessageBox(hWnd, L"Користувач клікнув File Open", L"Перевірка вибіру режиму", MB_OK);   //тест
}
```

і так далі

Для підтримки такого Toolbar можна записати на початку головного файлу *.cpp наступні рядки

```
#include <commctrl.h>
#pragma comment(lib, "comctl32.lib")

//--toolbar support--
HWND hwndToolBar = NULL;
#define IDC_MY_TOOLBAR 1
#define ID_TOOL_FILE_NEW 1
#define ID_TOOL_FILE_OPEN 2
#define ID_TOOL_FILE_SAVEAS 3
#define ID_TOOL_FILE_PRINT 4
```

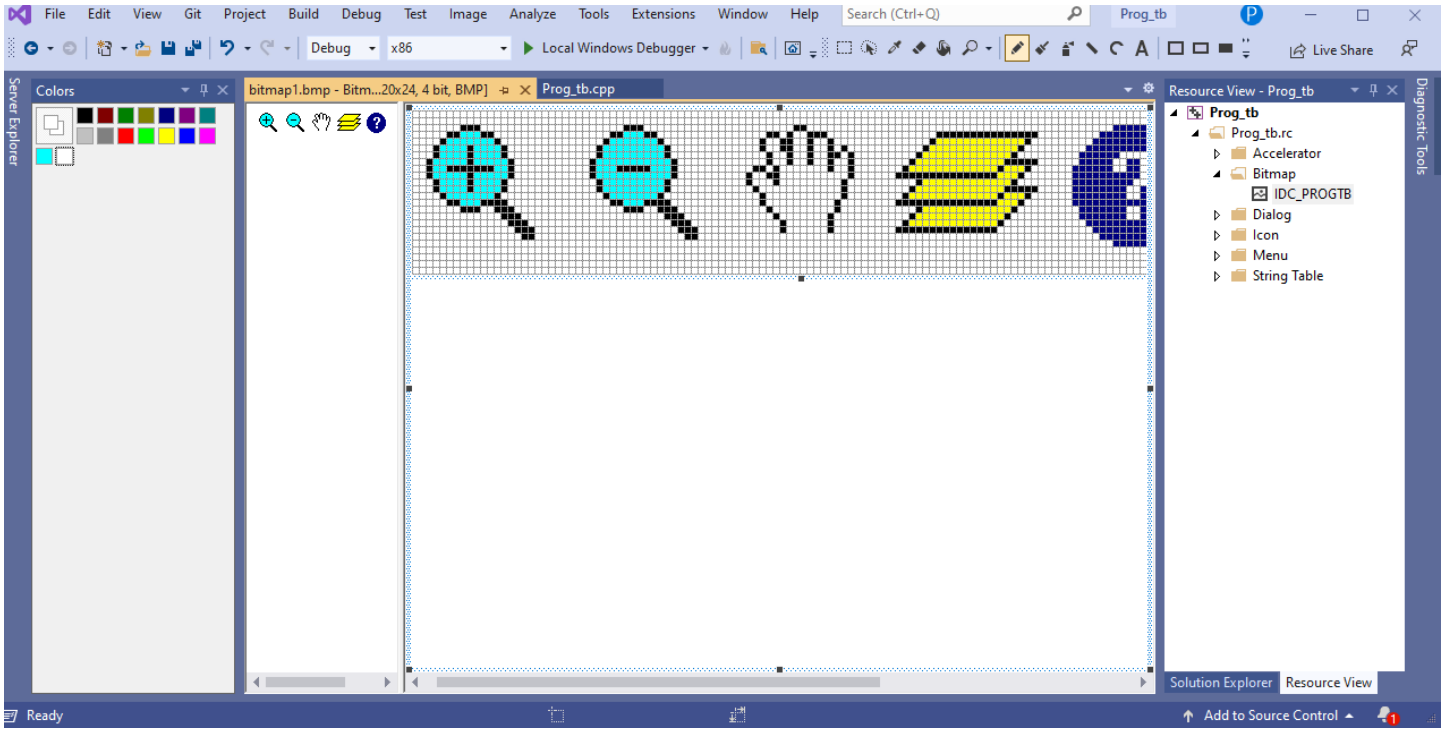
Врахування змін розмірів батьківського вікна

Якщо батьківське вікно раптом змінить розміри, то дочірнє вікно саме не зобов'язане також змінюватися. У першу чергу це стосується відповідності ширини Toolbar ширині головного вікна. Один з варіантів вирішення проблеми – належна обробка повідомлення **WM_SIZE** батьківського вікна. Нижче наведений код обробника, який змінює ширину вікна Toolbar відповідно ширині клієнтської частини батьківського вікна викликом функції **MoveWindow**.

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
switch (message)
   {
   case WM_CREATE:
      OnCreate(hWnd);
      break;

   case WM_SIZE:      //це повідомлення надсилається, якщо вікно змінить розмір
      OnSize(hWnd);
      break;
. . .
}

//---обробник повідомлення WM_SIZE---
void OnSize(HWND hWnd)
{
RECT rc,rw;

if (hwndToolBar)
   {
   GetClientRect(hWnd, &rc);              //нові розміри головного вікна
   GetWindowRect(hwndToolBar, &rw);       //нам потрібно знати висоту Toolbar
   MoveWindow(hwndToolBar, 0,0, rc.right-rc.left, rw.bottom-rw.top, FALSE);
   }
}
```

Власні зображення на кнопках Toolbar

Для цього можна створити **ресурс-bitmap**, який буде містити потрібні зображення, які потім можна відобразити на кнопках. Щоб створити такий ресурс, треба у вікні Solution Explorer клікнути на файлі ресурсів, потім додати у ресурси Bitmap. Потрібно вказати розміри цього BITMAP у пікселах. Тут потрібно розуміти, що картинки кнопок у растрі будуть розташовані по горизонталі, тому то для 5 кнопок 24×24 потрібен BITMAP розмірами 120×24. Цей BITMAP буде зберігатися у папці проекту у файлі формату BMP. Після створення нового BITMAPу треба намалювати потрібні зображення кнопок – засобами редактора ресурсів, або ще якось.

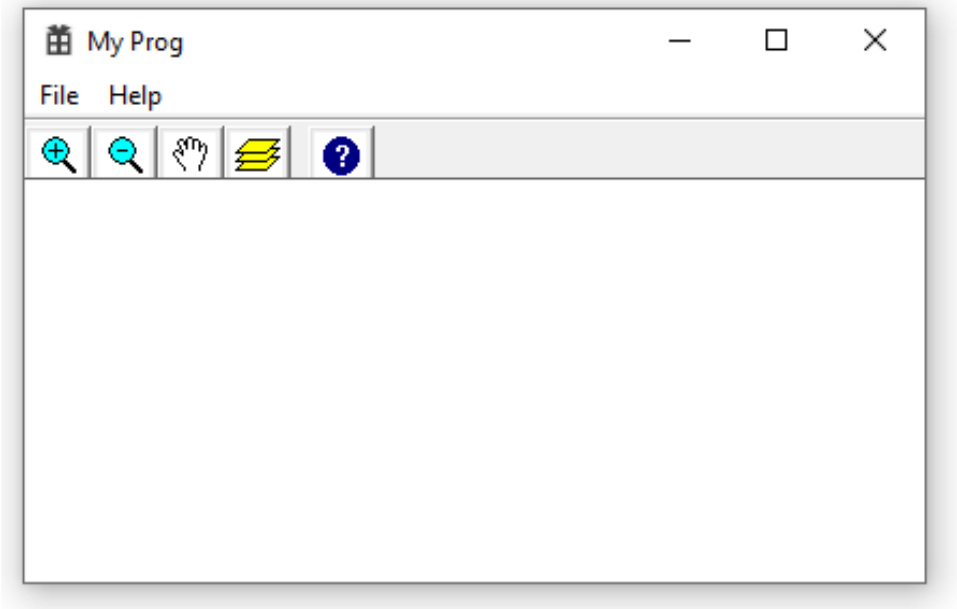


Створення та редагування бітмапу для кнопок Toolbar у середовищі Visual Studio

Запрограмувати прикріплення власних зображень до кнопок Toolbar можна так:

```
TBBUTTON tbb[6];

ZeroMemory(tbb, sizeof(tbb));
tbb[0].iBitmap = 0;
tbb[0].fsState = TBSTATE_ENABLED;
tbb[0].fsStyle = TBSTYLE_BUTTON;
tbb[0].idCommand = ID_TOOL_ZOOMPLUS;

tbb[1].iBitmap = 1;
tbb[1].fsState = TBSTATE_ENABLED;
tbb[1].fsStyle = TBSTYLE_BUTTON;
tbb[1].idCommand = ID_TOOL_ZOOMINUS;

tbb[2].iBitmap = 2;                 //індекс зображення у BITMAP
tbb[2].fsState = TBSTATE_ENABLED;
tbb[2].fsStyle = TBSTYLE_BUTTON;
tbb[2].idCommand = ID_TOOL_MOVE;

tbb[3].iBitmap = 3;
tbb[3].fsState = TBSTATE_ENABLED;
tbb[3].fsStyle = TBSTYLE_BUTTON;
tbb[3].idCommand = ID_TOOL_LAYERS;

tbb[4].iBitmap = 0;
tbb[4].fsState = TBSTATE_ENABLED;
tbb[4].fsStyle = TBSTYLE_SEP;       //роздільник груп кнопок
tbb[4].idCommand = 0;

tbb[5].iBitmap = 4;
tbb[5].fsState = TBSTATE_ENABLED;
tbb[5].fsStyle = TBSTYLE_BUTTON;
tbb[5].idCommand = IDM_ABOUT;

hwndToolBar = CreateToolbarEx(hWnd,
                              WS_CHILD | WS_VISIBLE | WS_BORDER | WS_CLIPSIBLINGS | CCS_TOP,
                              IDC_MY_TOOLBAR,
                              5,              //кількість зображень у BITMAP
                              hInst,
                              IDB_BITMAP1,    //ID ресурсу BITMAP
                              tbb,
                              6,              //кількість кнопок (разом з роздільником)
                              24,24,24,24,    //розміри кнопок та зображень BITMAP
                              sizeof(TBBUTTON));
```

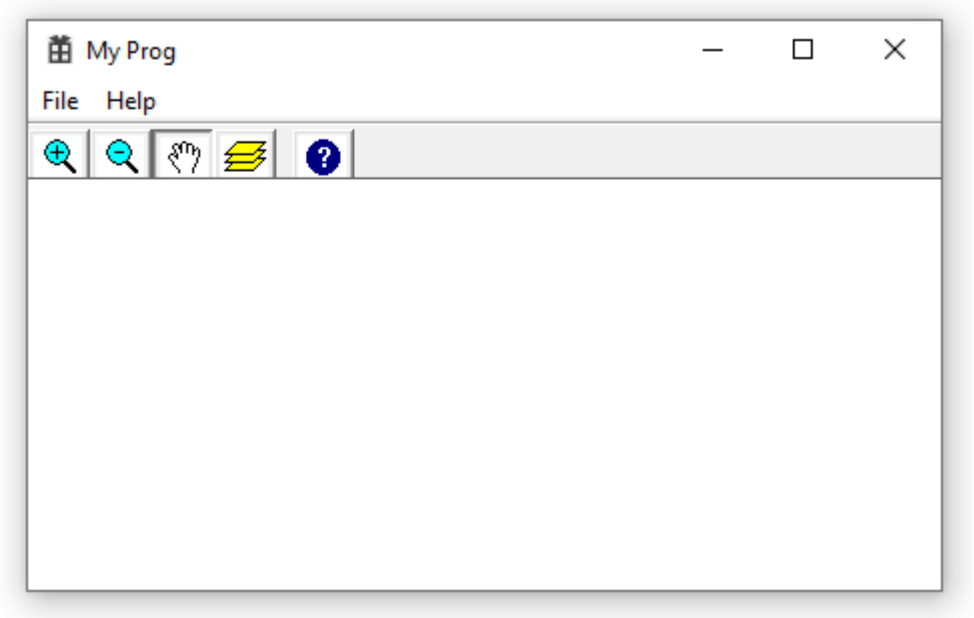
Результат:



Керування станом кнопок

Вікну Toolbar і навіть кожній кнопці окремо можна надсилати повідомлення. Назви цих повідомлень починаються з **TB_**
Наприклад, якщо надіслати повідомлення **TB_PRESSBUTTON**, то можна "програмно" натиснути або віджати кнопку. Нижче наведено реалізацію кнопки, яка фіксується

```
int press = 0;
. . .

void OnToolMove(HWND hWnd)
{
press = !press;
SendMessage(hwndToolBar,TB_PRESSBUTTON, ID_TOOL_MOVE, press);
}
```



Підказки для кнопок (tooltips)

Для цього потрібно при створенні Toolbar вказати стиль **TBSTYLE_TOOLTIPS**

```
hwndToolBar = CreateToolbarEx(hWnd,
                              WS_CHILD | WS_VISIBLE | WS_BORDER |
                              WS_CLIPSIBLINGS | CCS_TOP |
                              TBSTYLE_TOOLTIPS,
                              IDC_MY_TOOLBAR,
                              5,
                              hInst,
                              IDB_BITMAP1,
                              tbb,
                              6,
                              24,24,24,24,
                              sizeof(TBBUTTON));
```

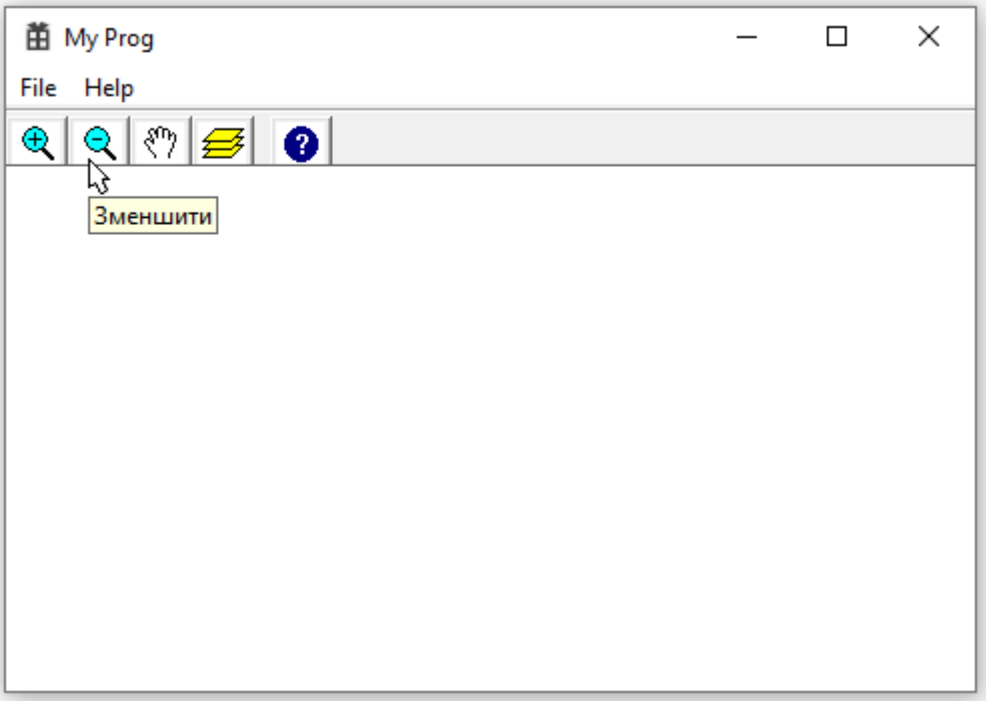
Але це ще не все. Потрібно ще зробити обробник повідомлення **WM_NOTIFY**

```
LRESULT CALLBACK WndProc(HWND hWnd,UINT message,WPARAM wParam,LPARAM lParam)
{
switch (message)
   {
   case WM_CREATE:
      OnCreate(hWnd);
      break;

   case WM_SIZE:
      OnSize(hWnd);
      break;

   case WM_NOTIFY:           //повідомлення від кнопок
      OnNotify(hWnd,wParam,lParam);
      break;

   case WM_COMMAND:
      . . .
```

Обробник повідомлення **WM_NOTIFY** для підказок можна запрограмувати так:

```
void OnNotify(HWND hWnd, WPARAM wParam, LPARAM lParam)
{
    LPNMHDR pnmh = (LPNMHDR)lParam;

    if (pnmh->code == TTN_NEEDTEXT)
    {
       LPTOOLTIPTEXT lpttt = (LPTOOLTIPTEXT)lParam;
       switch (lpttt->hdr.idFrom)
       {
       case ID_TOOL_ZOOMPLUS:
          lstrcpy(lpttt->szText, L"Збільшити");
          break;
       case ID_TOOL_ZOOMINUS:
          lstrcpy(lpttt->szText, L"Зменшити");
          break;
```

```
        case ID_TOOL_MOVE:
           lstrcpy(lpttt->szText, L"Пересунути");
           break;
        case ID_TOOL_LAYERS:
           lstrcpy(lpttt->szText, L"Вибрати");
           break;
        case IDM_ABOUT:
           lstrcpy(lpttt->szText, L"Довідка");
           break;
        default: lstrcpy(lpttt->szText, L"Щось невідоме");
        }
    }
}
```



Таким чином, ми повинні обробляти вже достатньо багато видів повідомлень, які надсилаються головному вікну нашої програми.

Варіанти завдань та основні вимоги

1. У звіті повинна бути схема успадкування класів – **діаграма класів**

2. Усі методи-обробники повідомлень, зокрема, і метод **OnNotify**, повинні бути функціями-членами деякого класу (класів).

3. Документи звіту – тексти, діаграми, схеми тощо оформлювати у електронному форматі так, щоб їх легко було сприймати і у надрукованому звіті. Забороняється текст або графіка "світле на світлому фоні" або "темне на темному фоні". Тільки чорний текст та чорні лінії на білому фоні. Оформлення звіту впливатиме на оцінку.

4. Для вибору типу об’єкту в графічному редакторі Lab3 повинно бути **вікно Toolbar з кнопками відповідно типам об'єктів.** Кнопки дублюють підпункти меню "Об’єкти". Кнопки мають бути з підказками (tooltips). Меню "Об’єкти" повинно бути праворуч меню "Файл" та ліворуч меню "Довідка". Підпункти меню "Об’єкти" містять назви геометричних форм українською мовою. Геометричні форми згідно варіанту завдання.

5. Для вибору варіанту використовується Ж = $Ж_{лаб2}$ + 1, де $Ж_{лаб2}$ – номер студента в журналі, який використовувався для попередньої лаб. роботи №2.

6. Масив вказівників для динамічних об’єктів типу Shape
- динамічний масив Shape **pcshape;
- статичний масив Shape *pcshape[N];

кількість елементів масиву вказівників як для статичного, так і динамічного має бути N = Ж+100.
Динамічний масив обирають студенти, у яких (Ж mod 3 = 0). Решта студентів – статичний масив. Позначка mod означає залишок від ділення.

7. "Гумовий" слід при вводі об’єктів
- суцільна лінія чорного кольору для студентів, у яких (Ж mod 4 = 0)
- суцільна лінія червоного кольору для студентів, у яких (Ж mod 4 = 1)
- суцільна лінія синього кольору для студентів, у яких (Ж mod 4 = 2)
- пунктирна лінія чорного кольору для студентів, у яких (Ж mod 4 =3)

8. Чотири геометричні форми (крапка, лінія, прямокутник, еліпс) можуть мати наступні різновиди вводу та відображення.

8.1. Прямокутник
Ввід прямокутника:
- по двом протилежним кутам для студентів, у яких (Ж mod 2 = 0)
- від центру до одного з кутів для (Ж mod 2 = 1)

Відображення прямокутника:
- чорний контур з білим заповненням для (Ж mod 5 = 0)
- чорний контур з кольоровим заповненням для (Ж mod 5 = 1 або 2)

- чорний контур прямокутника без заповнення для (Ж mod 5 = 3 або 4)

Кольори заповнення прямокутника:
- жовтий для (Ж mod 6 = 0)
- світло-зелений для (Ж mod 6 = 1)
- блакитний для (Ж mod 6 = 2)
- рожевий для (Ж mod 6 = 3)
- помаранчевий для (Ж mod 6 = 4)
- сірий для (Ж mod 6 = 5)

8.2. Еліпс
Ввід еліпсу:
- по двом протилежним кутам охоплюючого прямокутника для (Ж mod 2 = 1)
- від центру до одного з кутів охоплюючого прямокутника для (Ж mod 2 = 0)

Відображення еліпсу:
- чорний контур з білим заповненням для (Ж mod 5 = 1)
- чорний контур з кольоровим заповненням для (Ж mod 5 = 3 або 4)
- чорний контур еліпсу без заповнення для (Ж mod 5 = 0 або 2)

Кольори заповнення еліпсу:
- жовтий для (Ж mod 6 = 1)
- світло-зелений для (Ж mod 6 = 2)
- блакитний для (Ж mod 6 = 3)
- рожевий для (Ж mod 6 = 4)
- помаранчевий для (Ж mod 6 = 5)
- сірий для (Ж mod 6 = 0)

9. Позначка поточного типу об’єкту, що вводиться
- в меню (метод OnInitMenuPopup) для студентів (Ж mod 2 = 0)
- в заголовку вікна для (Ж mod 2 = 1)

10. Приклад вибору варіанту. Для 8-го студента у списку (Ж = $Ж_{лаб2}$ + 1 = 9) буде:
- динамічний масив для Shape (9 mod 3 = 0) обсягом 109 об’єктів
- "гумовий" слід (9 mod 4 = 1) – суцільна лінія червоного кольору
- прямокутник:
  - ввід від центру до одного з кутів (9 mod 2 = 1)
  - чорний контур прямокутника без заповнення (9 mod 5 = 4)
- еліпс:
  - по двом протилежним кутам охоплюючого прямокутника (9 mod 2 = 1)
  - чорний контур з кольоровим заповненням (9 mod 5 = 4)
  - колір заповнення: блакитний (9 mod 6 = 3)
- позначка поточного типу об’єкту: в заголовку вікна (9 mod 2 = 1)

Контрольні запитання

1. Обробку яких повідомлень потрібно виконувати у програмі Лаб3?
2. Що таке абстрактний клас і скільки їх у цій програмі?
3. Як забезпечити відповідність пунктів меню і кнопок Toolbar?
4. Як запрограмувати показ власних зображень на кнопках Toolbar?
5. Як створити власні зображення кнопок і де вони зберігаються?
6. Як запрограмувати текст підказок (tooltips)?

У ході захисту-прийняття роботи викладач може також запитувати інше, що стосується виконання роботи.

Зміст звіту

1. Титульний аркуш
2. Варіант завдання
3. Вихідний текст головного файлу .cpp (фрагменти, що ілюструють власний код), та вихідні тексти власних модулів
4. Схеми, діаграми згідно завданню
5. Ілюстрації (скріншоти)
6. Висновки