Тема 5. Створення багатодокументного інтерфейсу (MDI Multiple Document Interface)

1. Характеристики Windows-додатку, який працює з декількома документами

2. Перетворення звичайного SDI проекту з однією формою в проект MDI

3. **Операції перетворення типів**

4. Асоціювання документів з MDI-програмами, що їх обробляють

- 4.1.Схема роботи Mdi-програми та її реалізація
- 4.2 Побудова інтегрованого середовища мови програмування SPL

1. Характеристики Windows-додатку, який працює з декількома документами

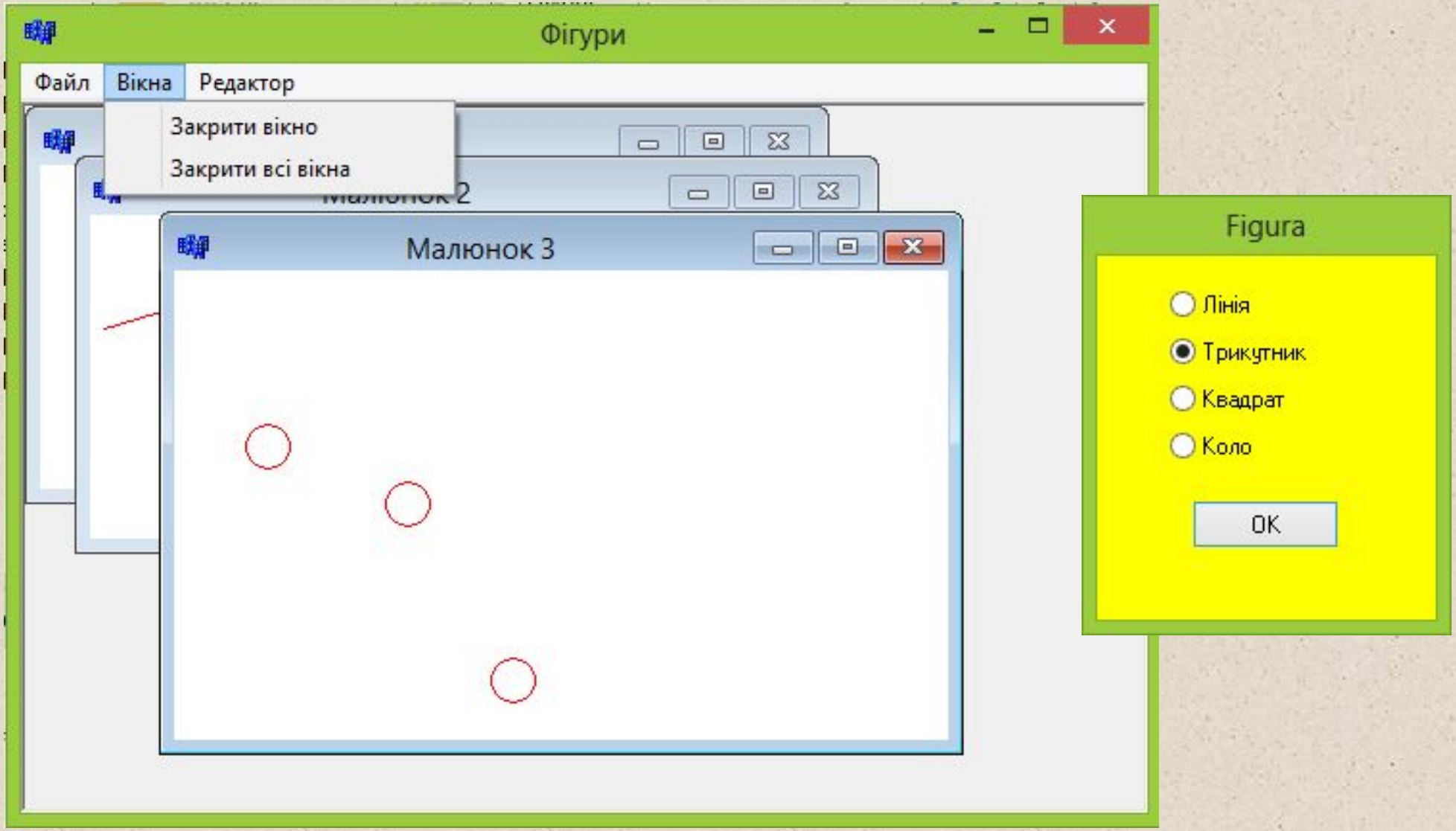
- **Multiple document interface** (**MDI**) - спосіб організації графічного інтерфейсу користувача, який передбачає використання віконного інтерфейсу, в якому більшість вікон (за виключенням, як правило, тільки модальних вікон) розташовані всередині одного загального вікна. Цим він і відрізняється від SDI, в якому вікна розташовуються незалежно один від одного.

- **Single document interface** (**SDI**) - спосіб організації графічного інтерфейсу додатків в окремих вікнах. Не існує «головного» або «батьківського» вікна, що містить меню або панелі інструментів для інших вікон - кожне вікно несе в собі ці елементи і є незалежним.

Multiple document interface (MDI)

- Будь-який документ, що відкривається займає своє вікно, яке називається дочірнім вікном. Воно є частиною робочої області основного вікна MDI;
- Дочірніх вікон може бути багато, батьківських тільки одне;
- Тільки одне дочірнє вікно може бути активним, його рядок в заголовку виділяється (наприклад у Word-і);
- Дочірнє вікно можна пересувати, змінювати розміри, але воно належить батьківському вікну (в границях його робочої області);
- Якщо максимізувати дочірнє вікно, то воно повністю покриває робочу область батьківського вікна, але не весь екран;
- При створенні додатку, який працює більш ніж з одним однотипним документом одночасно бажано створювати батьківську форму (MDI-форму) і одну дочірню форму.

2. Перетворення звичайного простого проекту з однією формою в проект MDI

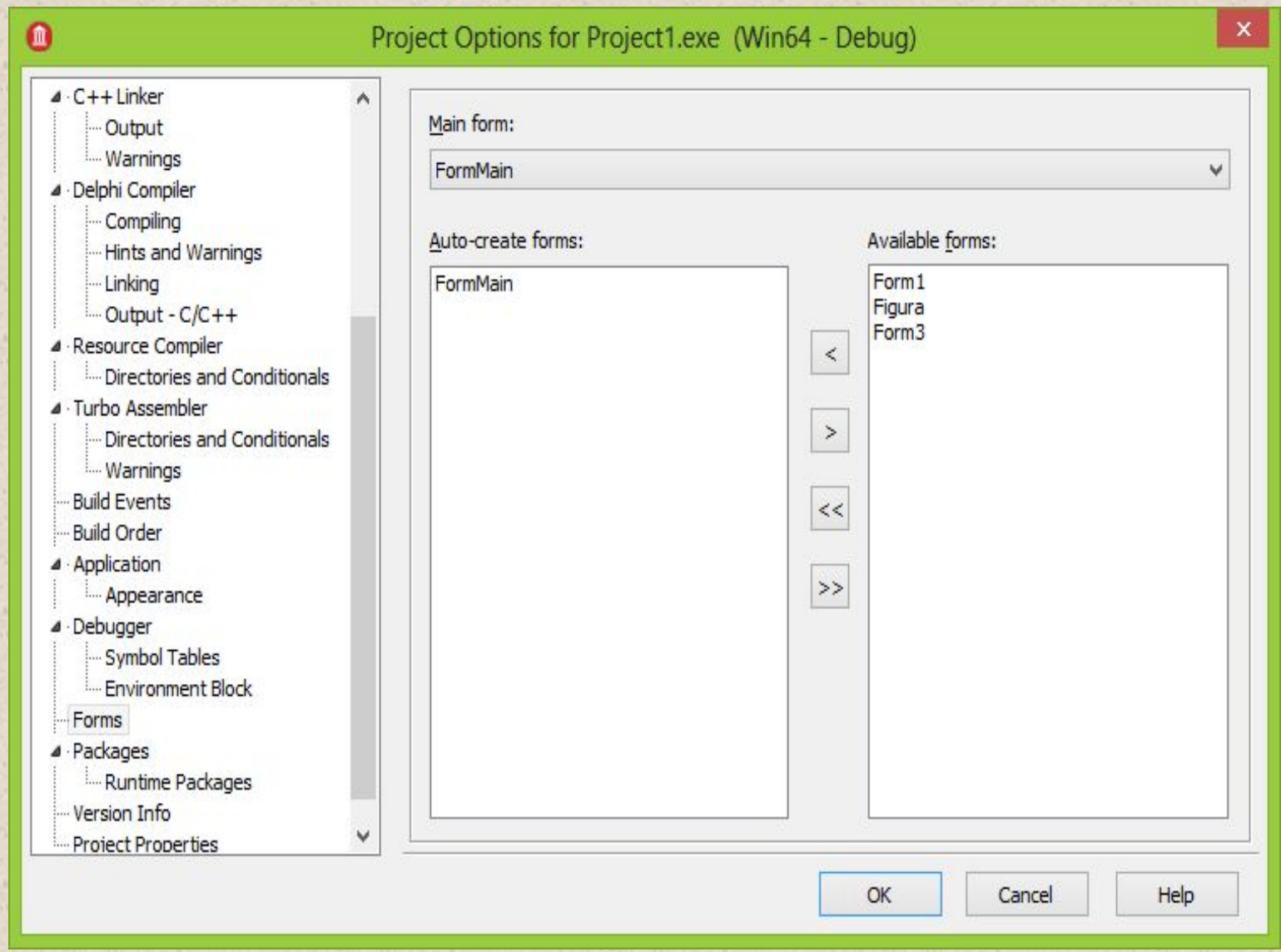


SDI -> MDI

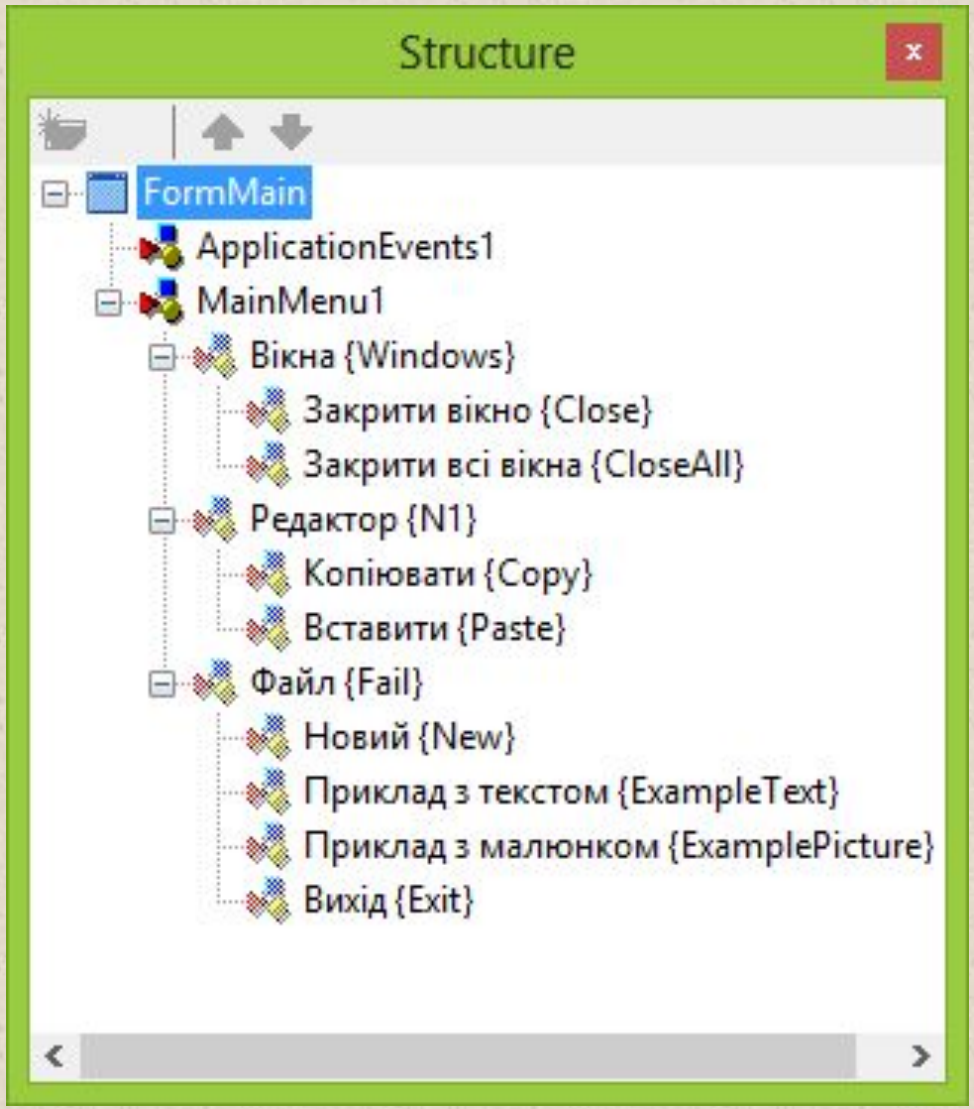
- Скопіюємо проект в інший каталог (не використовувати Save Project As, бо новий проект буде використовувати файли з старого каталогу).
- Створимо нову форму, яка буде батьківською
  - **Name – FormMain**
  - **Caption – Фігури**
  - **FormStyle – fsMDIForm (була fsNormal)**
- Зробимо нашу стару форму (Form1) дочірньою
  - **FormStyle – fsMDIChild**
- Дочірні форми найчастіші всі –одного типу, але можливі й різні типи форм (у нашому випадку можна завести ще дочірню форму для вимальовування тексту)

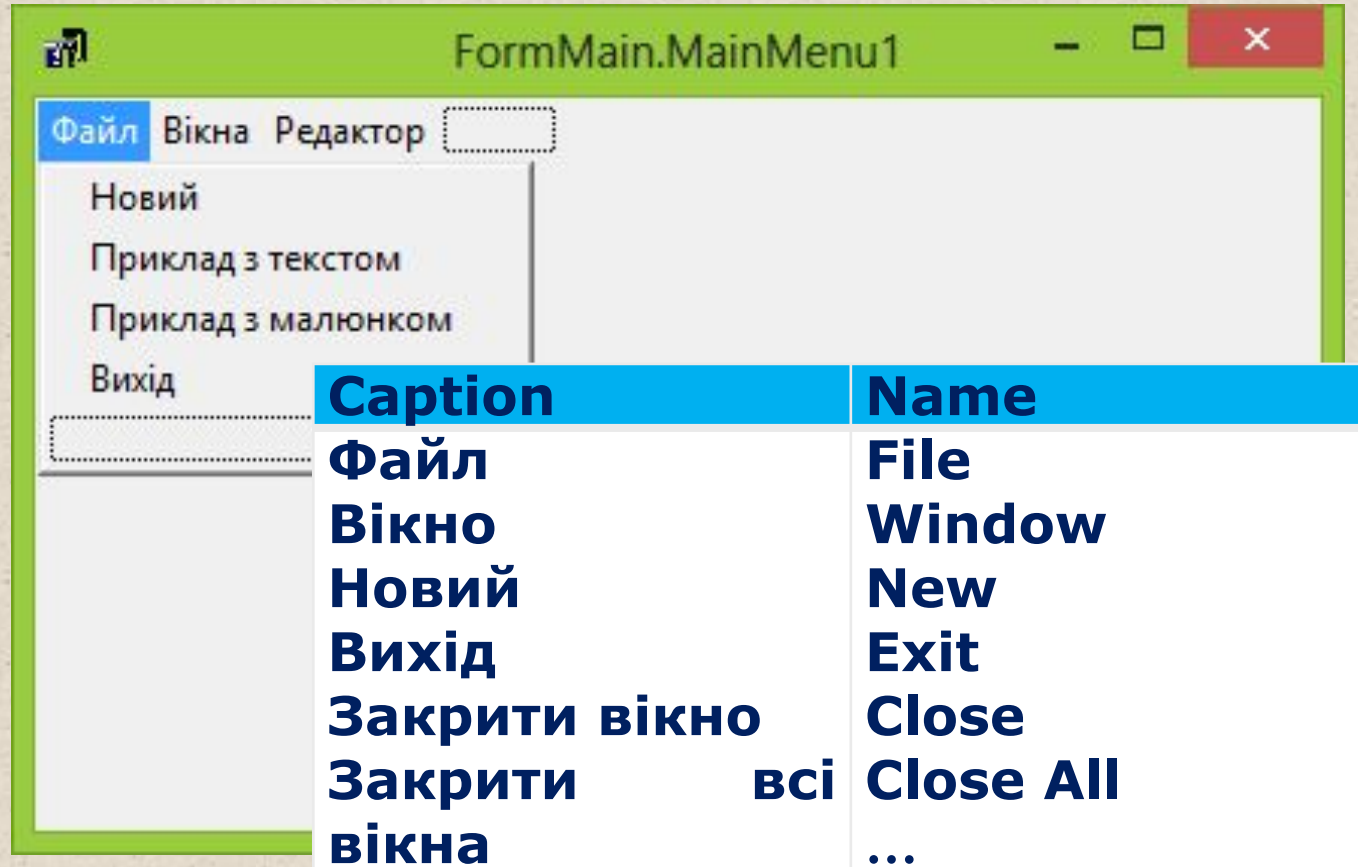
Вікно властивостей проекту (Embarcadero)

- Треба зробити форму FormMain першою (головною формою проекту)
  - Для цього (в CB6 **Project | Options**, в Embarcadero **Project | Options |Form**)
  - В вікні встановити Main form вибрати зі списку FormMain
  - З вікна Auto-Creat-Form перетягнути дочірні форми (Form1, Form3) в праве вікно. Це зробить форми такими, що не відкриються автоматично при відкритті батьківського вікна.



Батьківська форма з головним меню

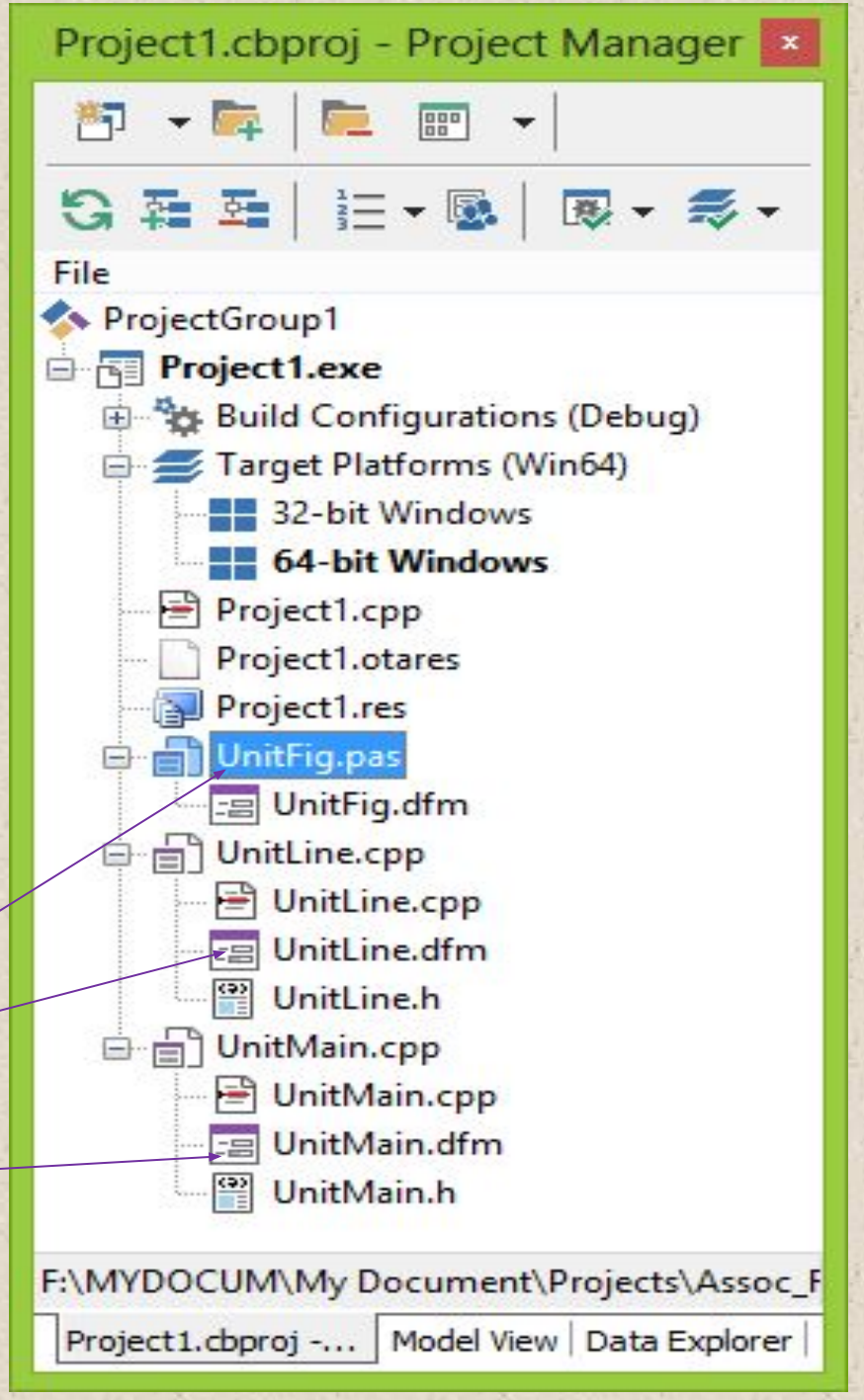




Caption	Name
Файл	File
Вікно	Window
Новий	New
Вихід	Exit
Закрити вікно	Close
Закрити всі вікна	Close All
…	…

Складові частини проекту

- Познайомимо батьківську і дочірню форми.
  - Для цього при активному модулі з розширенням .cpp
  - виберемо пункт **File | include Unit Hdr**
  - зі списку виберемо потрібний модуль.
  - це добавить рядок #include до модуля .cpp.

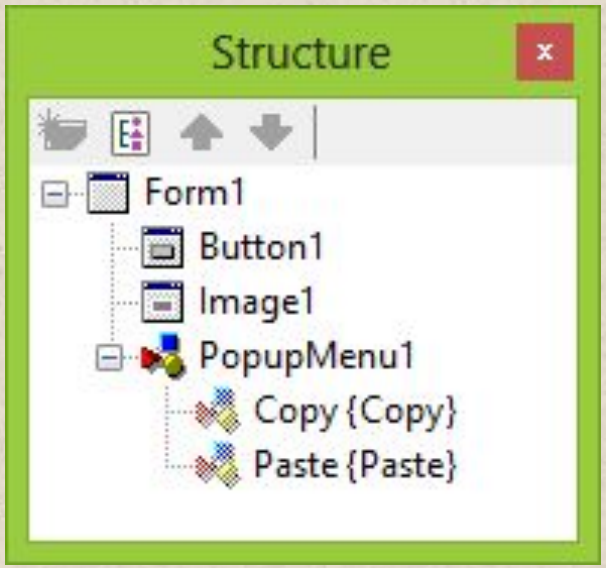


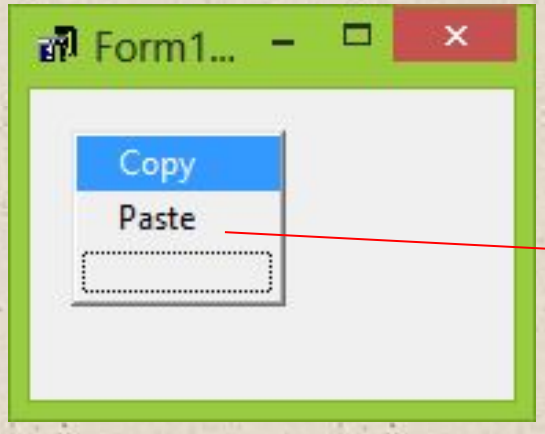
модальне вікно (паскаль)

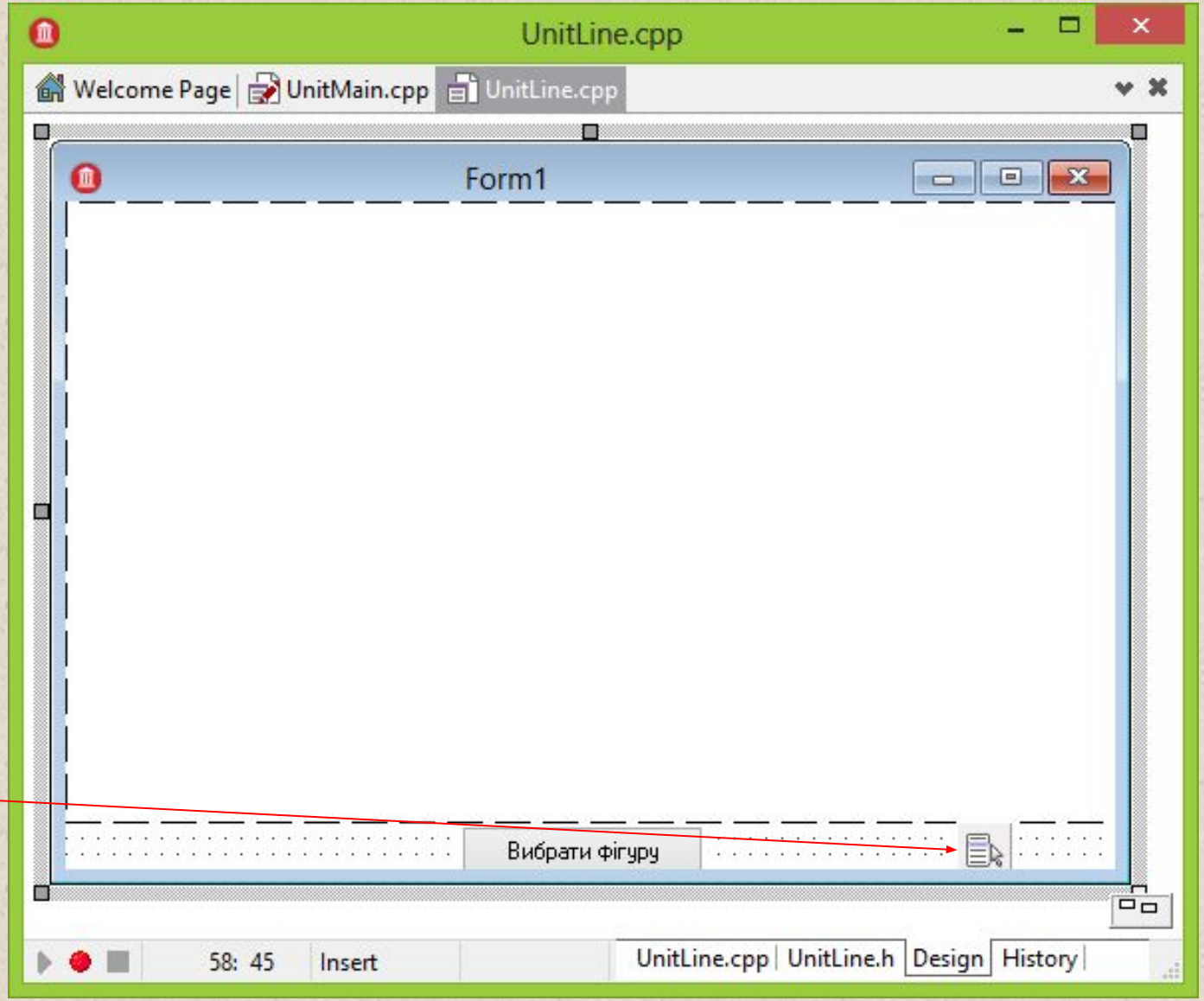
дочірня

батьківська

Дочірня форма з контекстним меню

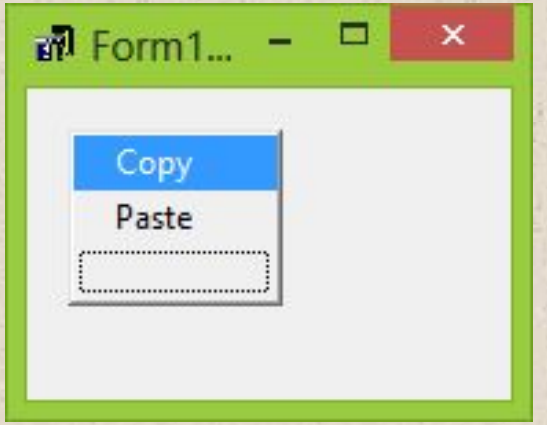






Обробники подій дочірньої форми (робота з буфером)

```
#include <Clipbrd.hpp>
…
void __fastcall TForm1::CopyClick(TObject *Sender)
{
Graphics::TBitmap *p =Image1->Picture->Bitmap;
Clipboard()->Assign(p);
Paste->Enabled=true;
}
void __fastcall TForm1::PasteClick(TObject *Sender)
{
 Image1->Picture->Assign(Clipboard());
}
```



Обробники подій батьківської форми

```
//Пункт підменю Новий
void __fastcall TFormMain::NewClick(TObject *Sender)
{
static int n=1;  //для номера дочірнього вікна
// вказівник на нову дочірню форму типу TForm1
TForm1 *pForm1=new TForm1(Application);
pForm1->Caption="Малюнок  "+String(n++);
}
// Пункт підменю Вихід
void __fastcall TFormMain::ExitClick(TObject *Sender)
{
Application->Terminate();
}
```

Обробник події OnCloseClick батьківської форми

- Для батьківського вікна є властивість **MDIChildCount**, яка доступна під час виконання і означає кількість дочірніх вікон. Властивість **MDIChildren** представляє собою масив вказівників на дочірні вікна, причому **MDIChildren**[0]- вказує на активне вікно, **MDIChildren**[1] – на перше, і т.д.
- Для того, щоб закрити форму, яка створювалась динамічно є оператор Release.

```
void __fastcall TFormMain::CloseClick(TObject *Sender)
{
MDIChildren[0] -> Release(); // закриття активного дочірнього вікна
}
```

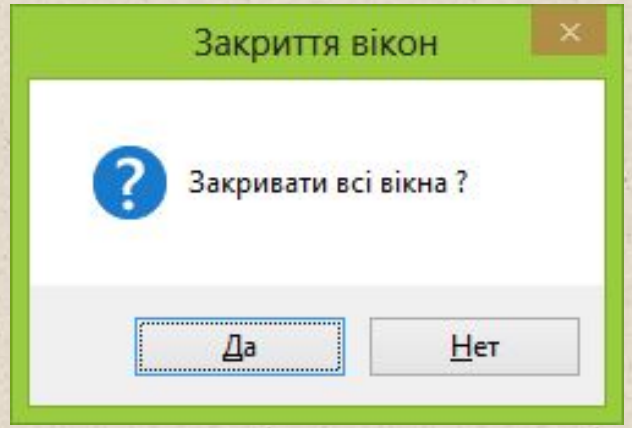
Обробник події OnFormClose дочірньої форми

Закрити дочірню форму можна і за допомогою ✕ на дочірній формі, але тоді форма буде згортатись. Тому напишемо обробник події OnClose для дочірніх форм. Якщо не написати, то дочірні вікна будуть згортатись.

```
void __fastcall TForm1::FormClose(TObject *Sender,
TCloseAction &Action)
{
Release();
}
```

Обробник події OnCloseAllClick батьківської форми

```
void __fastcall TFormMain::CloseAllClick(TObject *Sender)
{
int btn=Application->MessageBox("Закривати всі вікна ?",
 L"Закриття вікон", MB_YESNO+MB_ICONQUESTION); // XE
if(btn==IDYES)
   for(int i=0;i<MDIChildCount;++i)
       {
         TForm *pForm =static_cast<TForm*> (MDIChildren[i]);
    if(pForm)pForm->Release();
       }
}
```



Обробники подій батьківської форми (робота з буфером)

```
#include <Clipbrd.hpp>
…
void __fastcall TFormMain::CopyClick(TObject *Sender)
{
Graphics::TBitmap *p =((TForm1*)MDIChildren[0])->Image1->Picture->Bitmap;
//TForm1 *pF=static_cast<TForm1 *>(MDIChildren[0]);
// if(pF) Graphics::TBitmap *p =pF->Image1->Picture->Bitmap;

Clipboard()->Assign(p);
Paste->Enabled=true;
}
void __fastcall TFormMain::PasteClick(TObject *Sender)
{
   ((TForm1*)MDIChildren[0])->Image1->Picture->Assign(Clipboard());
}
```

Обробники подій батьківської форми для створення заповнених інформацією дочірніх вікон

```
void __fastcall TFormMain::ExampleTextClick(TObject *Sender)
{
TForm1 *pForm=new TForm1(Application);
pForm->Caption="Приклад 1";
TImage*  pImage=pForm->Image1;
TCanvas* pCanvas = pForm->Image1->Canvas;
pCanvas->Brush->Color = clBlue;   pCanvas->Font->Color =clWhite;
pCanvas->FillRect(TRect(TPoint(1,1),TPoint(pImage->Width,pImage->Height)));
pCanvas->TextOutA(200,250,"HELLO !");
}
void __fastcall TFormMain::ExamplePictureClick(TObject *Sender)
{
 TForm1 *pForm=new TForm1(Application);  pForm1->Caption="Приклад 2";
 pForm->Image1->Picture->LoadFromFile("MyPicture.bmp");
 pForm->Image1->Stretch=true;
}
```

3. Операції перетворення типів

- const_cast
- dynamic_cast
- reinterpret_cast
- static_cast
- (тип)

Перетворення з базового класу в похідний називають таким, що *понижує* (downcast), з похідного класу в базовий - таким, що *підвищує* (upcast), а приведення типів між класів одного рівня ієрархії - *перехресним* (crosscast).

Операція приведення типів в стилі С

тип (вираз) або **(тип) вираз**

```
int a = 2; float b = 6.8;
printf ("% lf% d", double (a), (int) b);
```

Явне перетворення типу є джерелом можливих помилок, залишено в С++ тільки для низхідної сумісності, і *використовувати його не* рекомендується

Операція const_cast

const_cast <тип> (вираз)

Використовується для видалення модифікатора const. Як правило, використовується при передачі у функцію константного покажчика на місце формального параметра, що не має модифікатора const. Формує результат зазначеного типу.

Необхідність введення цієї операції обумовлена тим, що правила C ++ забороняють передачу константного покажчика на місце звичайного. Операція const_cast використовується в тому випадку, коли програміст впевнений, що в тілі функції значення, на яке посилається вказівни, не змінюється. Природно, якщо є можливість додати до опису формального параметра модифікатор const, це переважніше

Операція dynamic_cast

dynamic_cast <тип *> (вираз)

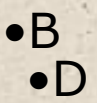
Застосовується для перетворення вказівників споріднених класів ієрархії.

- Вираз має бути вказівником або посиланням на клас, тип - базовим або похідним для цього класу.
- У разі успішного виконання операція формує результат заданого типу, інакше для вказівника результат дорівнює NULL, а для посилання породжується виключення bad _ cast.
- Якщо заданий тип і тип виразу не відносяться до однієї ієрархії, перетворення не допускається.
- Якщо вираз дорівнює NULL, результат також дорівнює NULL.
- Аргумент операції dynamic_cast повинен бути полиморфного типу.

Результат застосування операції dynamic_cast до вказівника завжди потрібно перевіряти явним чином.

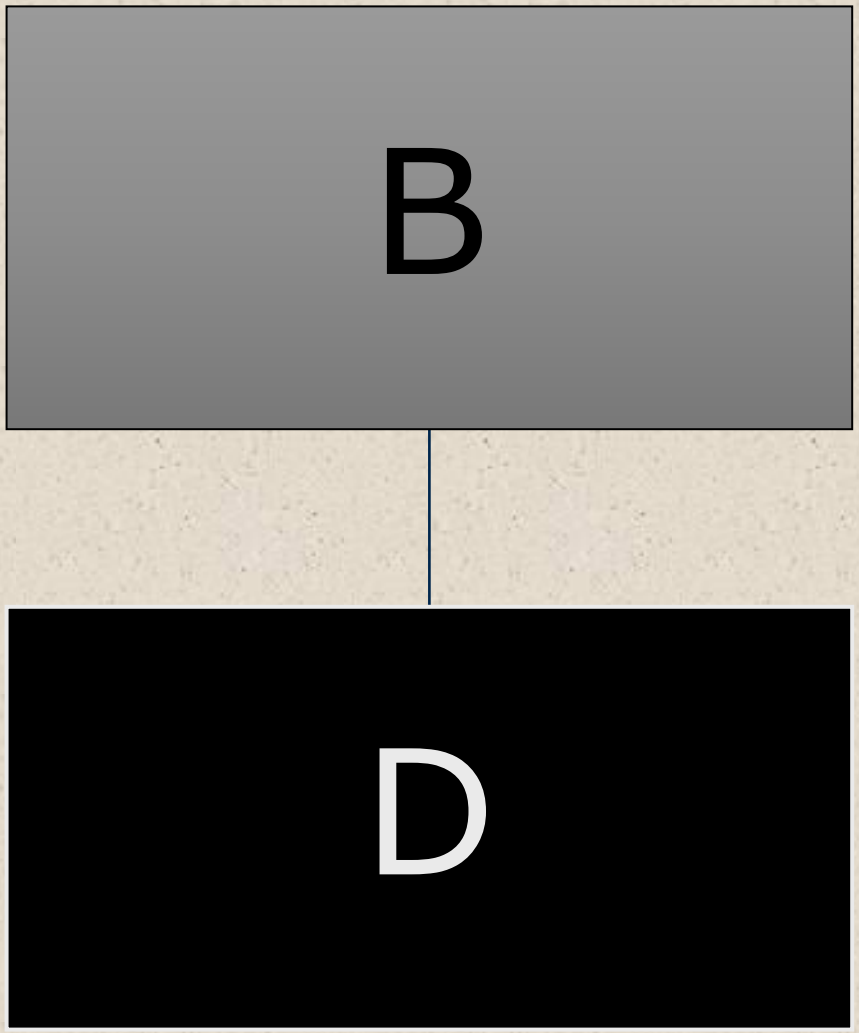
Приклад перетворення, що *підвищує* (upcast)

```
class B {/ * * /};
class D: public B {/ * * /};
D* d = new D;

B* b = dynamic_cast <B *> (d);
// Еквівалентно B* b = d;
```



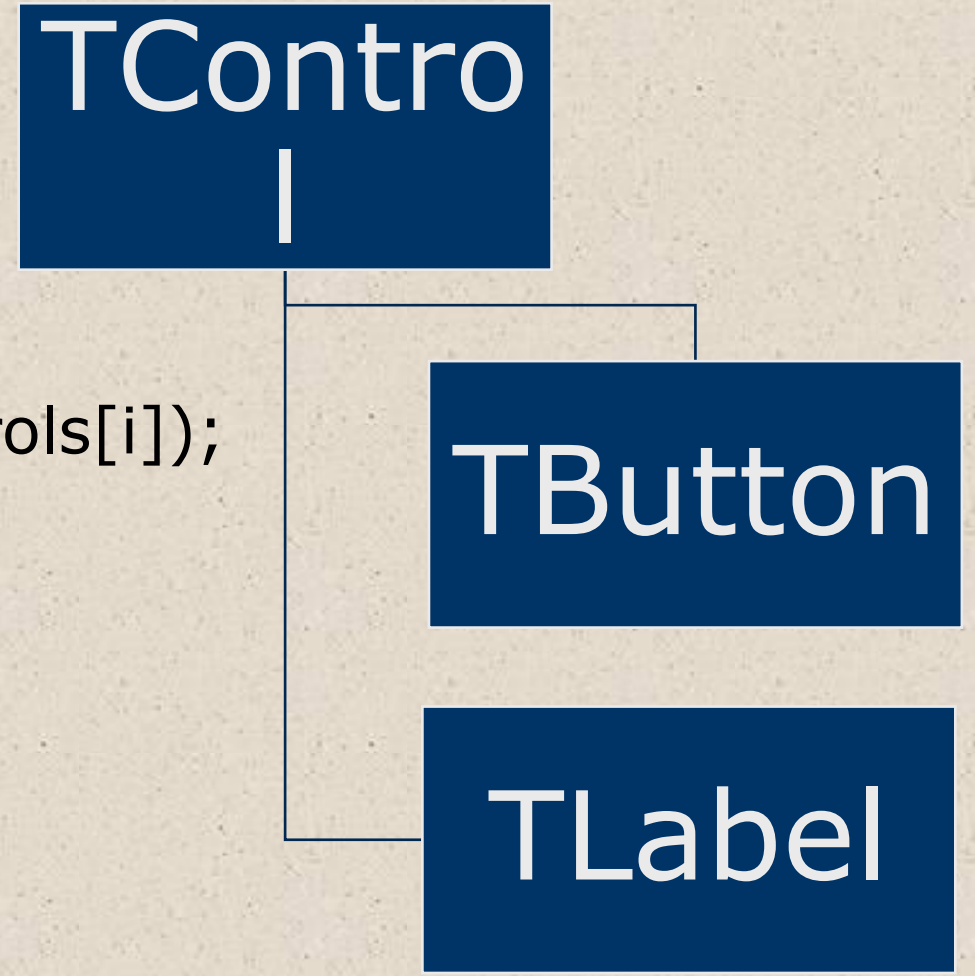
Приклад перетворення, що *понижує* (downcast)

```
class B {…};
class D: public B
 {public: void f(){cout << "f\n";}; };

void demo(B* p){
    D* d = dynamic_cast<D*>(p);
    if (d) d->f();
    else cout << "Передано не клас D";
    }
 int main(){
    B* b = new B;
    demo(b);// друк: "Передано не клас D"

    D* d = new D;
    demo(d);  // друк: "f" (правильно)
    return 0;
}
```



Приклад перетворення, що *понижує* (downcast)

```
for(int i=0;i<ControlCount;++i)
{
TButton *p =dynamic_cast<TButton*> (Controls[i]);
    if(p){
                p->Width=200;
        p->Height=100;
        . . .
             }
}
```



Операція static_cast

- Операція використовується для перетворення типу між:
  - цілими типами;
  - цілими і дійсними типами;
  - цілими і типами перерахування;
  - вказівниками і посиланнями на об'єкти однієї ієрархії
- Операція static_cast дозволяє виконувати перетворення з похідного класу в базовий і навпаки без обмежень.
- Перетворення виконується при компіляції, при цьому об'єкти можуть не бути поліморфними.
- Програміст повинен сам відслідковувати допустимість подальших дій з перетвореними величинами.

Формат операції: **static_cast <тип> (вираз)**

Результат операції має вказаний тип, який може бути посиланням, покажчиком, арифметичним або перераховуваних типом.

Приклад перетворення static_cast і dynamic_cast

```
for(int i=0;i<MDIChildCount;++i)
{
//TForm1 *pF =dynamic_cast<TForm1*> (MDIChildren[0]);
TForm1 *pF =static_cast<TForm1*> (MDIChildren[0]);
(pF)pF->Release();
 }
```

- static_cast знищує всі форми-нащадки TForm
- dynamic_cast знищує тільки форми типу TForm1

- TForm
  - TForm1
  - TForm2
  - TForm3

Операція reinterpret_cast

- Застосовується для перетворення не пов'язаних між собою типів, наприклад, покажчиків в цілі або навпаки, а також покажчиків типу void * в конкретний тип.
- Внутрішнє представлення даних залишається незмінним.
- **reinterpret_cast <тип> (вираз)**
- Результат операції має вказаний тип, який може бути посиланням, покажчиком, цілим або речовим типом.
- char * p = reinterpret_cast <char *> (malloc (100));
- long l = reinterpret_cast <long> (p);
- Різниця між static_cast і reinterpret_cast дозволяє компілятору виробляти мінімальну перевірку при використанні static_cast, а програмісту - позначати небезпечні перетворення за допомогою reinterpret_cast. Результат перетворення залишається на совісті програміста.

4. Асоціювання документів з MDI-програмами, що їх обробляють

4.1.Схема роботи Mdi-програми та її реалізація

Схема:

- Перевірка існування в пам’яті вже запущена копія цієї програми;
- Якщо програми в пам’яті не існує, то продовжити роботу, і обробити рядок вхідних параметрів;
- Якщо програма в пам’яті існує – передати повідомлення запущеній копії програми про необхідність створення нового дочірнього вікна (причому так, щоб Mdi-програма знала про необхідність обробки вхідних параметрів) і завершити своє виконання.

Реалізація

А) створення унікального м’ютекса

М’ютекс – це спеціальний синхронізуючий об’єкт в міжпроцесній взаємодії, що подає сигнал, коли він не захоплений будь-яким процесом. В нашому випадку достатньо знати, що після створення м’ютекса першою копією нашої програми знову створити такий же м’ютекс буде неможливо. Цей факт і використовується для перевірки того, чи працює в даний момент ще одна копія програми. Формат команди:

```
HANDLE CreateMutex(
   // pointer to security attributes
    LPSECURITY_ATTRIBUTES lpMutexAttributes,
    BOOL bInitialOwner,  // flag for initial ownership
    LPCTSTR lpName  // pointer to mutex-object name
   );
```

CreateMutex(NULL, false, "MySpl");

Якщо м’ютекс створено, то функція GetLastError() поверне значення *ERROR_ALREADY_EXISTS*.

Б) Пошук запущеної копії програми

- Вказівник на головне вікно програми шукається за допомогою функції **FindWindow()**, яка шукає вікна за належністю до певного класу або за іменем цього вікна.

```
HWND FindWindow(
    LPCTSTR lpClassName, // pointer to class name
    LPCTSTR lpWindowName  // pointer to window name
   );
```

- У випадку вдалого виконання ця функція повертає вказівник на знайдене вікно. В протилежному випадку **NULL**.

```
int d=FindWindow("TFrameForm",NULL)
```

- В) Створення повідомлення і його реєстрація

```
const int WM_MY=
RegisterWindowMessage("MySoob");
```

- Г) Відправлення повідомлення програмі А (разом з параметрами)
  - **I спосіб – PostMessage і буфер**
  - **II спосіб – SendMessage і** повідомлення WM_COPYDATA

I спосіб – PostMessage і буфер або 1-2 параметри цілого типу

```
BOOL PostMessage(
  HWND hWnd,// handle of destination window дескр. Вікна d
  UINT Msg, // message to post            WM_MY
  WPARAM wParam,// first message parameter  0
  LPARAM lParam     // second message parameter  0
 );
```

- Якщо функція PostMessage відпрацювала неуспішно, то причину помилки можна дізнатися, викликавши функцію GetLastError().

```
#include <Clipbrd.hpp>
  TClipboard* cb = Clipboard();
  cb->SetTextBuf(ParamStr(1).c_str());
  PostMessage(d, WM_MY,0,0);
```

II спосіб – SendMessage
і повідомлення WM_COPYDATA

LPRESULT SendMessage(HWND hwnd, // дескриптор вікна,
d**UINT Msg, //** повідомлення WM_COPYDATA
WPARAM wParam, // перший параметр WM_MY
LPARAM lParam);//другий параметр (LPARAM)&cds

```
COPYDATASTRUCT cds;

cds.cbData = ParamStr(1).Length() +1;
cds.lpData = ParamStr(1).c_str();

SendMessage(FindWindow("TFrameForm",0),
            WM_COPYDATA,
            0,
            (LPARAM)&cds);
```

http://www.sergeev.sebastopol.ua/oslr04.html

Робота з повідомленнями WM_COPYDATA

- WM_COPYDATA - це повідомлення спеціально створено для того, щоб дозволити одному додатку відправляти дані іншому додатку. При відправці повідомлення WM_COPYDATA, передається дескриптор вікна і покажчик на структуру COPYDATASTRUCT в значенні параметра LPARAM.
- Структура COPYDATASTRUCT :

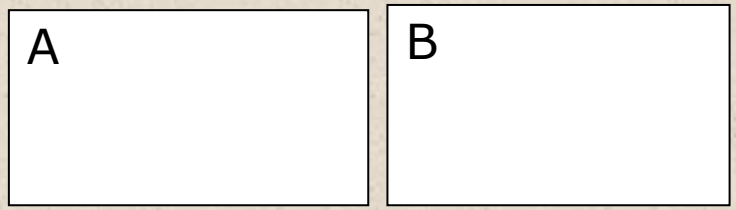
```
typedef struct tagCOPYDATASTRUCT
   {
    DWORD dwData;
    DWORD cbData;
    PVOID lpData;
   } COPYDATASTRUCT, * PCOPYDATASTRUCT;
```

- Значення члена dwData може бути використано, якщо треба передати 32 біта даних у другій екземпляр. Якщо потрібно передати блок пам'яті в другій екземпляр – треба встановити значення члена cbData в розмір переданого блоку, а значення члена lpData - в початкову адресу блоку пам'яті.
- Windows гарантує, що дані, що відправляються в структурі COPYDATASTRUCT, існуватимуть, поки повідомлення WM_COPYDATA НЕ буде оброблено.

Д) Перехоплення повідомлення програмою А

- Далі (при використанні функції PostMessage()) достатньо перехопити це повідомлення за допомогою стандартного компонента **TApplicationEvents**, для якої необхідно написати обробник події **OnMessage**.
- Для функції SendMessage() треба утворити обробник подій для перехоплення повідомлення. Це робиться за допомогою карти повідомлень (обробник визначається за допомогою макросів, описаних у приватній секції батьківської форми).

```
private:  // User declarations
  void __fastcall OnWMCopyData(TWMCopyData &Msg);

  BEGIN_MESSAGE_MAP // карта повідомлень
        MESSAGE_HANDLER(WM_COPYDATA, TWMCopyData, OnWMCopyData);
  END_MESSAGE_MAP(TForm);
```

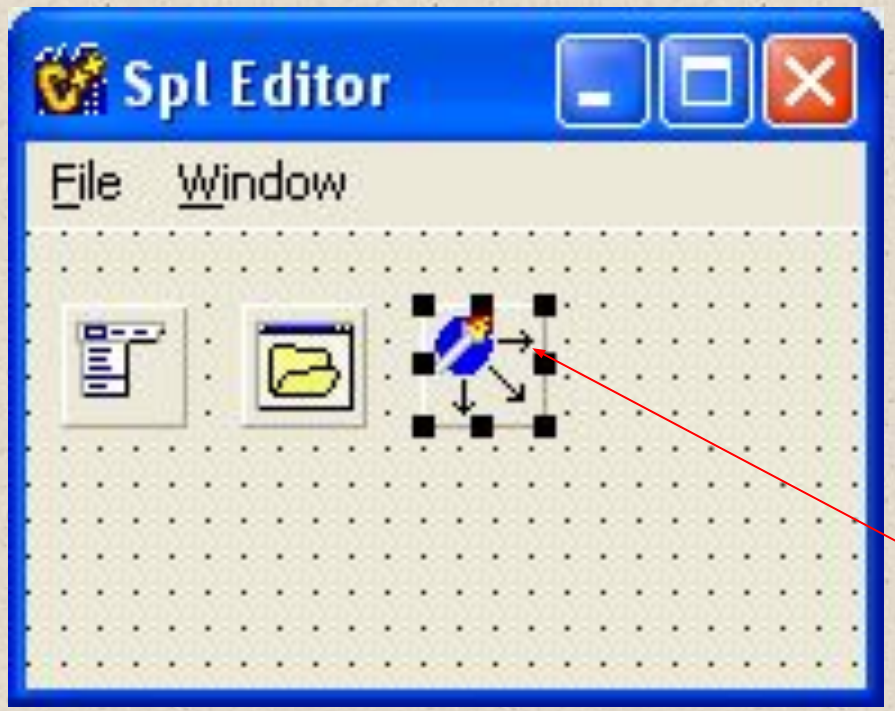


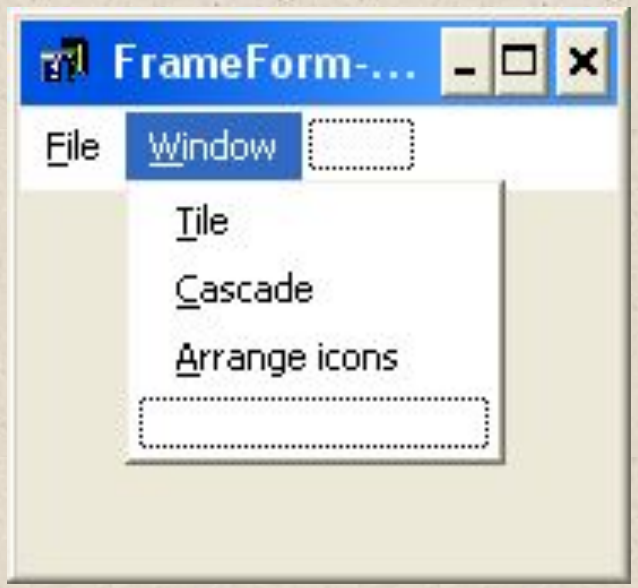
Е) Відкриття програмою А потрібних дочірніх вікон

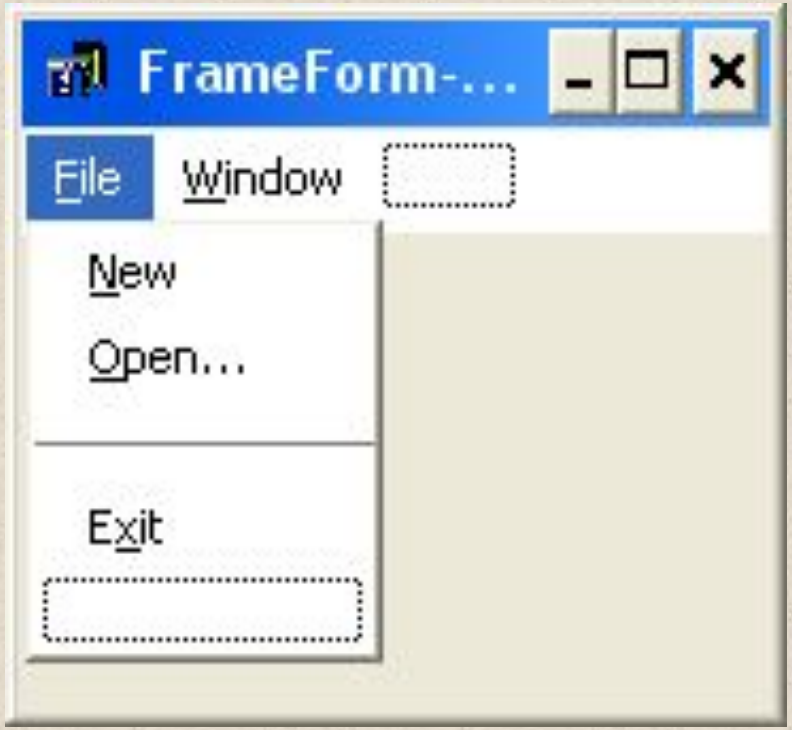
4.2 Побудова інтегрованого середовища мови програмування SPL

Задача. Створити MDI-програму для створення, зберігання, редагування і запуску програм, написаних мовою SPL

Батьківська форма

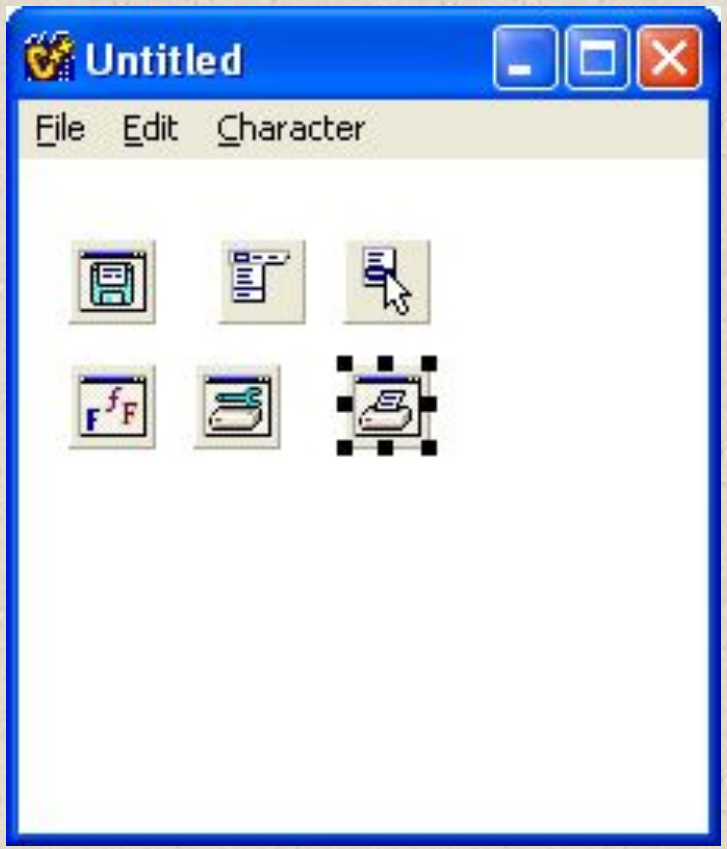


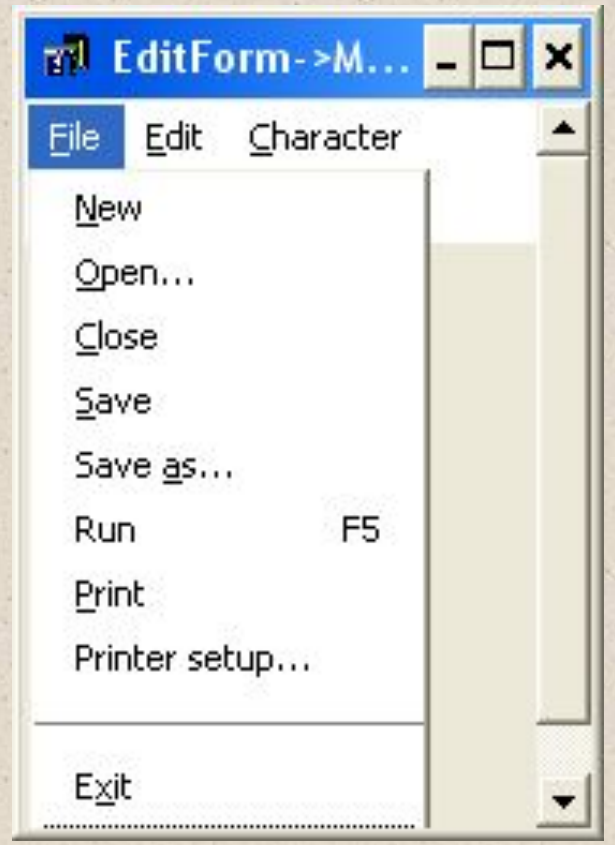




TApplicationEvents

Дочірня форма





До батьківського меню приєднано дочірнє меню

На дочірній формі ще розташований компонент типу **TRichEdit.**
Його основні змінені властивості: Name=Editor; Flat=False;

Реалізація задач асоціації

А) Створення унікального м’ютекса **MySpl**

Б) Пошук програмою В запущеної копії (програми А)

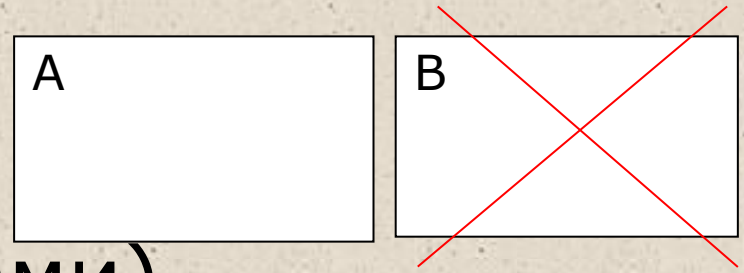
В) Створення повідомлення **MySoob** і його реєстрація

Г) Відправлення повідомлення програмі А (разом з параметрами-іменами файлів з SPL-програмами)

- I спосіб – PostMessage і буфер або 1-2 параметри цілого типу
- II спосіб – SendMessage і повідомлення WM_COPYDATA

Д) Перехоплення повідомлення програмою А і знищення програми В

Е) Відкриття програмою А потрібних дочірніх вікон (файлів з SPL-програмами)



Файл проекту project1.cpp

```
#include <vcl.h>
#include <Clipbrd.hpp> //буфер
...
USEFORM("MDIFrame.cpp", FrameForm); // батьківська форма
USERES("TextEdit.res");
USEFORM("MDIEdit.cpp", EditForm);   // дочірня форма
const int WM_MY = RegisterWindowMessage("MySoob"); // реєстрація повідомлення
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR args, int)
{       try
        {
          Application->Initialize();
          CreateMutex(NULL, false, "MySpl"); //створення унікального іменованого мютекса
          if (GetLastError()!=ERROR_ALREADY_EXISTS) //перевірка наявності копії мютекса
                {
                  Application->CreateForm(__classid(TFrameForm), &FrameForm);
                  Application->Run();
                }
```

```
else
        {
       // спосіб передачі параметрів через буфер і PostMessage
        /*    TClipboard* cb = Clipboard();
                    cb->SetTextBuf(ParamStr(1).c_str()); // перетворення AnsiString в char*
                    PostMessage(FindWindow("TFrameForm", NULL),WM_MY,0,0); */
       // спосіб передачі параметрів через SendMessage і WM_COPYDATA
          if(strlen(args) != 0)   // рядок параметрів в лапочках
                    {
                    COPYDATASTRUCT cds;
        //    cds.dwData = WM_MY;// 32 біти можно передати тут
        /*  cds.cbData = strlen(args);   cds.lpData = args+1;  args[strlen(args)-1]='\0';  */

                    cds.cbData = ParamStr(1).Length() +1;
                    cds.lpData = ParamStr(1).c_str();
          SendMessage(FindWindow("TFrameForm",0),WM_COPYDATA, 0, (LPARAM)&cds);
          }
    }
...
```