МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Івано-Франківський національний технічний університет нафти і газу
Кафедра програмного забезпечення автоматизованих систем
Кафедра технології нафтогазового машинобудування

Відкритий міжнародний університет розвитку людини "Україна"
Івано-Франківська філія
Кафедра інформаційних технологій та програмування

В.Б.Копей, І.З.Лютак, Я.Б.Сторож

ПРОГРАМУВАННЯ НА С++. ПРИКЛАДИ ПРОГРАМ З КОМЕНТАРІЯМИ

НАВЧАЛЬНИЙ ПОСІБНИК

Для студентів спеціальностей
"Програмне забезпечення автоматизованих систем"
"Технологія машинобудування"

Івано-Франківськ
2008

МВ 02070855-2071-2008
Копей В.Б., Лютак І.З., Сторож Я.Б. Програмування на С++. Приклади програм з коментаріями: Навчальний посібник. – Івано-Франківськ: Факел, 2008. – 170 с.

Навчальний посібник містить довідкові матеріали по мові програмування С++: термінологічний словник термінів програмування та приклади програм на мові С++ з детальними коментаріями. Пропонується для самостійного опрацювання студентами спеціальностей "Програмне забезпечення автоматизованих систем" та "Технологія машинобудування". Може бути корисним усім, хто вивчає та програмує на С++.

Рецензенти:

Канд. техн. наук, доцент кафедри комп'ютеризованого машинобудівного виробництва ІФНТУНГ О.Р. Онисько

Доктор техн. наук, зав. кафедри програмного забезпечення автоматизованих систем ІФНТУНГ В.М. Юрчишин

Канд. техн. наук, проректор з наукової роботи Відкритого міжнародного університету розвитку людини "Україна" К.О. Кольченко

Канд. техн. наук, доцент кафедри комп'ютерних інформаційних технологій Тернопільського національного економічного університету М.Я. Шпінталь

Канд. техн. наук, доцент кафедри інформатики Прикарпатського національного університету ім. Василя Стефаника В.О. Горелов

Зміст

Передмова

Термінологічний словник містить основні терміни програмування на алгоритмічній і об'єктно-орієнтованій мові C++, а також терміни сучасних об'єктно-орієнтованих технологій у програмуванні. Приклади програм підібрані прості, загальні і завершені, які охоплюють усі основні сторони програмування на C++: імперативне, об'єктно-орієнтоване і узагальнене програмування на C++, використання стандартних бібліотек C++, використання бібліотеки VCL Borland C++ Builder 6, основи технологій DLL, COM, CORBA, MFC, .NET. Вихідні коди програм містять детальні і чіткі коментарії без будь-яких додаткових зовнішніх пояснень. Тому, посібник має довідковий характер, і бажано його використовувати паралельно з рекомендованою авторами літературою по мові C++ (с.163). Порядок подання термінів і прикладів програм відповідає класичному порядку вивчення мови C++ та сучасних об'єктно-орієнтованих технологій у програмуванні. Приклади опрацьовані в середовищі Borland C++ Builder 6 (для програмування прикладів MFC і .NET використовували Microsoft Visual C++ 8).

Пропонується для самостійного опрацювання студентами спеціальності "Програмне забезпечення автоматизованих систем" при вивченні навчальної дисципліни "Об'єктно-орієнтоване програмування" та студентами спеціальності "Технологія машинобудування" при вивченні навчальних дисциплін: "Сучасні мови програмування" (для спеціалізації "Комп'ютеризоване машинобудівне виробництво"), "Основи наукових досліджень", "Засоби програмного керування технологічними системами", "Моделювання виробничих систем в машинобудуванні" при виконанні курсового і дипломного проектування.

Може бути корисним усім, хто вивчає та програмує на C++.

1 Програмування: термінологічний словник

1.1 Алгоритм, програма, мови програмування

Комп'ютер (електронна обчислювальна машина (ЕОМ)) – цифрова обчислювальна машина, побудована на електронних компонентах, призначена для автоматизації процесу обробки інформації і обчислень, які виконуються у відповідності з заданим алгоритмом. Складається з центрального процесору, пристроїв оперативної і зовнішньої пам'яті, пристроїв вводу-виводу.

Центральний процесор ЕОМ - центральний пристрій (або комплекс пристроїв) ЕОМ, який виконує арифметичні і логічні операції, задані програмою перетворення інформації, управляє обчислювальним процесом і координує роботу периферійних пристроїв системи. Процесор виконує команди на машинній мові, які зберігаються в оперативній пам'яті. Складається з пристрою керування, арифметико-логічного пристрою і набору регістрів (комірок швидкодіючої пам'яті). Процесору характерна певна система команд.

Пам'ять ЕОМ - частина ЕОМ, призначена для запису, зберігання і видачі інформації, представленій в кодовій формі; утворюється з одного або декількох запам'ятовуючих пристроїв (ЗП). Поділяється на оперативну (або основну), зовнішню (допоміжну) і буферну.

Оперативна пам'ять ЕОМ призначена для тимчасового збереження команд і даних. Кожен елемент пам'яті, називається словом пам'яті, має унікальну адресу, а кожне слово складається з фіксованого числа бітів, зазвичай з 16, 32 або 64 бітів.

Команда ЕОМ (процесора) - вказівка, що записана на машинній мові конкретної обчислювальної машини і визначає її дії при виконанні окремої операції або частини обчислювального процесу.

Програма обчислювальної машини, опис алгоритму рішення задачі, заданий на мові програмування (на машинну мову конкретної ЕОМ переводиться автоматично за допомогою транслятора). Процес складання програм називають програмуванням.

Програмне забезпечення ЕОМ (математичне забезпечення ЕОМ) - комплекс програм, описів і інструкцій, що забезпечують автоматичне функціонування ЕОМ. Розрізняють **загальне програмне забезпечення** (для організації обчислювального процесу на даній ЕОМ) і **спеціальне програмне забезпечення** (для вирішення конкретних завдань).

Програмування - процес підготовки завдань для вирішення їх на ЕОМ, що складається з наступних етапів: складання алгоритму; складання

програми на мові програмування; трансляція програми з мови програмування на машинну мову.

Системне програмування – програмування базового (загального) програмного забезпечення ЕОМ (операційних систем, трансляторів, сервісних засобів і т.д.)

Прикладне програмування – програмування спеціального програмного забезпечення прикладного характеру, яке призначене для рішення конкретних задач з різних областей (наука, техніка, виробництво, сфера обслуговування, навчання і т.д.)

Алгоритм - спосіб (програма) розв'язування обчислювальних і інших задач, який точно описує, як і в якій послідовності отримати результат, який однозначно визначається початковими даними. Алгоритм - одне з основних понять математики і кібернетики. У обчислювальній техніці для опису алгоритму використовуються мови програмування. Основними властивостями алгоритму є: зрозумілість для виконавця, дискретність, визначеність, результативність і масовість. Алгоритм може бути представлений словесно, графічно і програмно.

Мови програмування - формальні мови для опису даних (інформації) і алгоритму (програми) їх обробки на ЕОМ. Мови програмування можна розділяти на машинні, низького рівня (мови асемблера) і високого рівня (наприклад, алгоритмічні мови). За рівнем спеціалізації розрізняють спеціальні, спеціалізовані і універсальні мови програмування.

Машинна мова - мова програмування, призначена для представлення програм у формі, яка дозволяє виконувати їх на конкретній ЕОМ. Машинною мовою, також, називають систему команд ЕОМ. Наприклад, система команд процесорів Intel 8086, 386, Pentium.

Асемблер - допоміжна програма у складі операційної системи для автоматичного перекладу програми, що підлягає виконанню на ЕОМ, на машинну мову. Один з видів транслятора.

Середовище програмування - набір програм-інструментів, які використовуються для створення програм. Може містити редактор, компілятор або інтерпретатор, бібліотекар, компонувач, завантажувач, налагоджувач, засоби тестування та ін.

Транслятор - в інформатиці, програма ЕОМ, призначена для автоматичного перекладу опису алгоритму з однієї мови програмування на іншу, зокрема на машинну мову. Процес перекладу називається трансляцією. Транслятори поділяються на компілятори і інтерпретатори.

Компілятор – транслятор, який перекладає вихідний код програми на машинну (або іншу) мову у вигляді об’єктного коду. Компілятор також знаходить в програмі помилки компіляції і оптимізує команди машинної мови, тому скомпільовані програми працюють швидше інтерпретованих.

Інтерпретатор – транслятор, який почергово виконує команди інтерпретованої програми. Як інтерпретатори працюють мови Basic, LISP, Perl, Python.

Мови високого рівня – машинно-незалежні мови, які імітують природні, використовуючи зручні для сприйняття конструкції, деякі слова розмовної мови і загальноприйняті математичні символи. Поділяються на алгоритмічні (наприклад, Pascal), логічні (наприклад, Prolog) та об'єктно-орієнтовані (наприклад, C++).

Алгоритмічна мова – формалізована мова високого рівня для однозначного запису алгоритмів. Складається з набору символів (алфавіт), синтаксичних правил і семантичних визначень. Є основою мов програмування. Наприклад: Basic, C, Pascal, Fortran, C++, Java.

Еволюція мов програмування відбувалась в такому напрямку: машинні мови і асемблери; непроцедурні алгоритмічні мови; процедурні алгоритмічні мови для структурованого програмування; мови об'єктного і об'єктно-орієнтованого програмування.

1.2 Елементи мови програмування

Рекомендована література: [1-14]

Мова програмування C++ - об'єктно-орієнтована мова програмування високого рівня, яка включає описи класів, контроль типів, перевантаження функцій, керування пам'яттю, постійні типи, посилання, вбудовуванні функції, похідні класи і віртуальні функції. Розроблена на основі мови C, і на даний час є найбільш поширеною універсальною мовою системного і прикладного програмування.

Алфавіт мови – система нероздільних символів, які використовуються для побудови мови програмування. В алфавіт C++ входять великі і малі латинські літери, арабські цифри, та спеціальні символи: `"' () [] {} <> . , ; : ? ! ~ * + - = / \ | # % $ & ^ @ _`

Синтаксис мови – правила побудови конструкцій мови.

Семантика мови – визначає змістове значення конструкцій мови.

Команди мови (оператори, вказівки, речення, інструкції) – синтаксичні конструкції мови, які будуються з лексем. Наприклад, команди присвоювання, вводу-виводу, циклу, розгалуження.

Вираз – послідовність даних, операторів і розділювачів, яка конкретизує обчислення, наприклад, арифметичні (с.33), логічні вирази. Результатом виконання виразу є значення. Приклад:

```
y=(x*x+1)/z;//змінній y присвоїти значення виразу (x*x+1)/z
```

Лексеми (tokens) – неподільні елементи мови, первинні вирази: слова, число, символ операції. У C++ є шість класів лексем:

ідентифікатори, ключові слова, константи, рядки, оператори і інші розділювачі.

Ключові слова – зарезервовані ідентифікатори, які використовуються для написання команд, наприклад: `double`, `class`, `for`, `if`, `return`.

Директиви препроцесора – команди компілятора, які виконуються на першому етапі компіляції програми (с.31). Приклад:

```
#include <iostream.h>//директива підстановки файлу iostream.h
```

Коментар – пояснення вихідного коду програми, який ігнорується компілятором. Приклади:

```
/*багаторядковий коментар*/
//однорядковий коментар
```

Дане – поіменована область оперативної пам'яті, в якій зберігається значення певного типу. Даному характерні ім'я, адреса, тип і значення. Приклад:

```
int x=2;//даному з іменем x і типом int присвоїти значення 2
cout<<&x;//вивести на екран адресу цього даного
```

Ім'я (ідентифікатор) позначає дане унікальним іменем (змінну, об'єкт, функцію, тип, значення або мітку) (с.32).

Визначення імені (definition) – визначає для імені сутність, до якої воно відноситься, і деколи значення, виділяє пам'ять для об'єкту.

Оголошення імені (declaration) називає тип і ім'я об'єкта, але не виділяє пам'ять для нього, а тільки описує його абстрактні властивості. Оголошення є визначенням, за винятком тих випадків, коли воно описує функції, не задаючи тіла функції (с.41), коли воно містить специфікатор `extern` і в ньому немає ініціалізації або тіла функції, або коли воно є описом класу. Будь-яка сутність перед використанням повинна бути оголошена. В програмі може бути тільки одне визначення кожного імені і безліч оголошень. Приклади оголошень:

```
int i;
float x=1.17;
char* p="String";
struct s1{int n; float y;};
s1 obj1;
int f (int x){return x*x;};
typedef s1 s2;
```

Пов'язування (binding) – процес, в результаті якого ім'я або вираз асоціюється з певним атрибутом. Розрізняють **раннє** (статичне, на етапі компіляції) і **пізнє** (динамічне, на етапі виконання) **пов'язування**.

Ініціалізація – указання початкового значення об'єкту, що оголошується. Приклад:

```
int x=2;//присвоїти змінній x значення 2
```

Адреса даного – індекс (адреса) комірки оперативної пам'яті, в якій зберігається дане, або з якої воно починається (с.38). Приклад адреси змінної x: `&x`

Вказівник – це змінна, значенням якої є адреса даного (с.38). Якщо вказівник ні на що не вказує, то він має пусте значення. Приклад:

```
int *px=&x//створити вказівник px і присвоїти йому адресу змінної x
```

Область видимості імені – частина програми, де оголошене ім'я та допустиме його використання. Область видимості буває глобальна і локальна (с.42).

Час життя об'єктів (змінних). Якщо тільки програміст не втрутиться явно, об'єкт буде створений при появі його визначення і знищений, коли зникне з області видимості. Об'єкти з глобальними іменами створюються, ініціалізуються (причому тільки один раз) і існують до кінця програми. Якщо локальні об'єкти описані із службовим словом `static`, то вони також існують до кінця програми (с.42).

Тип даних - визначає допустимі значення даного, а також операції, які можуть виконуватись над ним (с.32). Для кожного типу даних резервується певний обсяг пам'яті (с.32). Тип є конкретне представлення деякої концепції (поняття). Наприклад, наявний в C++ тип `float` з його операціями +, -, * і т.д. забезпечує обмежену, але конкретну версію математичного поняття дійсного числа. Новий тип створюється для того, щоб дати спеціальне і конкретне визначення поняття, якому ніщо прямо і очевидно серед вбудованих типів не відповідає:

```
typedef <опис типу> <назва типу>;
```

Основні типи C++ (с.32):

`char` - символьний;
`int` – цілий;
`float` – дійсний;
`double` – дійсний подвійної точності;
`bool` - булевий (логічний);
`void` – використовується для визначення функції, яка не повертає ніяких значень, або для узагальненого вказівника.

Модифікації основних типів C++ (с.32) утворюються за допомогою слів:

`short` – для цілих типів,
`long` - для цілих типів і `double`,
`signed, unsigned` - для цілих і символьних типів.

Похідні типи C++:

масиви (с.39);
функції (с.41);
вказівники на об'єкти даного типу (с.38);

посилання на об'єкти даного типу;
константи, що є значеннями даного типу;
класи (с.46);
структури (с.47);
об'єднання (с.47).

Значення – вміст області пам’яті відведеної під дане певного типу.

Літерал – константа, конкретне значення, задане в програмі «буквально», у вигляді послідовності символів. Літерал може бути цілочисельний, символьний, дійсний, рядковий і булевий. Наприклад:

```
cout<<"x="<<125<<'\n';//"x=", 125, '\n' – літерали
```

Змінна – дане, значення якого може змінюватися протягом виконання програми. Приклад оголошення змінної:

```
float a;
```

Стала (константа) – дане, значення якого не може змінюватися протягом виконання програми. Приклад:

```
const float a=1.5;
```

Контроль відповідності типів - це перевірка того, що при присвоюванні змінній, тип виразу сумісний з типом змінної. Строгий контроль характеризується відмовою від виконання, якщо типи відрізняються.

Перетворення (приведення) типів – це операція перетворення значення одного типу до значення іншого типу (с.33). Відбувається тоді коли у виразі зустрічаються дані різного типу. Перетворення може бути **явне** і **неявне**. При неявному перетворенні результат за замовчуванням перетворюється до старшого типу. При явному перетворенні програміст вказує до якого типу перетворити змінну. Приклад:

```
i=(int)c;//явне приведення с до int
```

Динамічна ідентифікація типів RTTI (Run-Time Type Identification) – механізм, який дозволяє ідентифікувати тип об’єкта при виконанні програми і безпечно виконувати перетворення типів (с.61).

Посилання – змінна, оголошена за допомогою модифікатора "&" і використовується для надання ще одного імені (синоніму) даному. Приклад:

```
int &x2=x;//посилання x2
```

Логічний вираз – вираз, значення якого належить до логічного типу. Приклад:

```
bool  x=(2>1)&&(1<=5);//змінній x присвоїти значення логічного виразу
```

Перерахування (enum) – це унікальний тип даних, значення якого покриваються множиною іменованих констант, які називаються перечислювачами (с.41). Приклад:

```
enum days {sun,mon,tues,wed,thur,fri,sat};// sun=0, mon=1...
```

Оператори C++ і їх пріоритети (порядок виконання) (таблиця 1.1)

Усі оператори таблиці, відділені жирною горизонтальною лінією, мають однаковий пріоритет. Пріоритет зменшується "зверху вниз". Наприклад, `a+b*c` означає `a+(b*c)`, оскільки `*` має пріоритет вище, ніж `+`; а вираз `a+b-c` означає `(a+b)-c`, оскільки `+` і `-` мають однаковий пріоритет, і операції `+` і `-` застосовуються "зліва направо".

Таблиця 1.1 - Оператори C++ і їх пріоритети

Оператор	Опис	Синтаксис
`::`	Дозвіл області видимості	`class_name::member`
`::`	Глобальне	`::name`
`.`	Вибір члена	`object.member`
`->`	Вибір члена	`pointer->member`
`[]`	Індексування	`pointer[expr]`
`()`	Виклик функції	`expr(expr_list)`
`()`	Структурне значення	`expr(expr_list)`
`sizeof`	Розмір об'єкту	`sizeof expr`
`sizeof()`	Розмір об'єкту	`sizeof(expr)`
`++`	Постфіксний інкремент	`lvalue++`
`++`	Префіксний інкремент	`++lvalue`
`--`	Постфіксний декремент	`lvalue--`
`--`	Префіксний декремент	`--lvalue`
`~`	Доповнення	`~expr`
`!`	Логічне НЕ	`!expr`
`-`	Унарний мінус	`-expr`
`+`	Унарний плюс	`+expr`
`&`	Отримання адреси	`&lvalue`
`*`	Розіменування	`*expr`
`new`	Створення (розміщення)	`new type`
`delete`	Знищення (вивільнення)	`delete pointer`
`delete[]`	Знищення масиву	`delete[] pointer`
`()`	Приведення(перетворення) типу	`(type)expr`
`.*`	Вибір члена непрямий	`object.pointer-to-member`
`->*`	Вибір члена непрямий	`pointer->pointer-to-member`
`*`	Множення	`expr*expr`

Продовження табл. 1.1

Оператор	Опис	Синтаксис
/	Ділення	expr/expr
%	Остача від ділення	expr%expr
+	Додавання (плюс)	expr+expr
-	Віднімання (мінус)	expr-expr
<<	Зсув вліво	expr<<expr
>>	Зсув вправо	expr>>expr
<	Менше	expr<expr
<=	Менше або рівно	expr<=expr
>	Більше	expr>expr
>=	Більше або рівно	expr<=expr
==	Рівно	expr==expr
!=	Не рівно	expr!=expr
&	Порозрядне І	expr&expr
^	Порозрядне що виключає АБО	expr^expr
|	Порозрядне що включає АБО	expr|expr
&&	Логічне І	expr&&expr
||	Логічне АБО	expr||expr
? :	Операція умови	expr?expr:expr
=	Просте присвоювання	lvalue=expr
*=	Присвоювання з множенням	lvalue*=expr
/=	Присвоювання з діленням	lvalue/=expr
%=	Присвоювання з узяттям залишку від ділення	lvalue%=expr
+=	Присвоювання з додаванням	lvalue+=expr
-=	Присвоювання з відніманням	lvalue-=expr
<<=	Присвоювання з зсувом вліво	lvalue<<=expr
>>=	Присвоювання з зсувом вправо	lvalue>>=expr
&=	Присвоювання з порозрядним І	lvalue&=expr
|=	Присвоювання з порозрядним що включає АБО	lvalue|=expr
^=	Присвоювання з порозрядним що виключає АБО	lvalue^=expr

Команда присвоєння – використовується для присвоєння змінним (об'єктам) значень. Приклади:

```
a=1.25;
x=a+2;
```

Команди вводу-виводу – команди, які використовуються для здійснення операцій вводу-виводу на пристрої вводу-виводу (клавіатура, монітор, файл, периферійні пристрої) (с.34). Приклади:

```
cin>>x;//ввід з клавіатури
cout<<x;//вивід на монітор
```

Складена команда – використовується для об'єднання кількох команд в одну. Приклад:

```
{int x=1; cout<<x;}
```

Керуючі команди - призначені для зміни порядку виконання команд програми. Це команди безумовного переходу, розгалуження, циклу.

Команда безумовного переходу (goto) – використовується для переходу до виконання позначеної команди і наступних за нею команд (с.36). Приклад:

```
goto a;//перейти на мітку a
a:y=3;//команда під міткою a
```

Команди розгалуження (if-else, switch) – використовуються для переходу до виконання команд в залежності від значення певного логічного виразу (с.36). Приклад:

```
if (x>1) x=0;else x=1;//якщо x>1 то x=0, інакше x=1
```

Команди циклу (for, while, do-while) – використовуються для повтору команд задану кількість раз в залежності від значення певного логічного виразу (с.37). Приклад:

```
for (x=1;x<5;x++) cout<<x;//вивести на екран: 1234
```

Статична пам'ять – пам'ять, яка виділяється для збереження даних на початку роботи програми і зберігається до її завершення. Приклад:

```
int x=1;
```

Динамічна пам'ять - пам'ять, яка виділяється для збереження даних у динамічних змінних і вивільняється під час роботи програми. Це дозволяє економно використовувати оперативну пам'ять (с.39). Приклад:

```
int *pi=new int(2);
```

Файл – це сукупність даних, які розміщені у постійному запам'ятовуючому пристрої. Для роботи з файлом необхідно відкрити для доступу фізичний файл і пов'язати з ним файлову змінну (об'єкт), виконати операції вводу-виводу і закрити файл. Файли бувають послідовного доступу і довільного (прямого), текстові (містять символи форматування тексту) і бінарні.

Файл послідовного доступу – файл, доступ до записів якого виконується послідовно з першого по останній (с.67).

Файл довільного доступу – файл, доступ до кожного запису якого може виконуватись напряму (с.68).

Підпрограма - це пойменована частина програми, яка містить групу команд і може викликатися з інших частин програми потрібну кількість раз. У підпрограму можуть передаватись дані для обробки (аргументи, фактичні параметри). Підпрограми поділяються на **процедури** і **функції**. Результатом виконання функції, на відміну від процедури, є якесь значення. В мові С++ усі підпрограми є функціями (с.41). Приклад:

```
int sum1(int a,int b)//визначення функції sum1
{return a+b;}//функція повертає значення a+b цілого типу
```

Прототип функції – опис функції з параметрами і типом результату, але без її реалізації. Приклад:

```
int sum1(int a,int b=0);
```

Формальні параметри підпрограми – її локальні змінні, які приймають значення аргументів підпрограми. Приклад:

```
int sum1(int a,int b)//a,b - формальні параметри
```

Фактичні параметри – значення даних, або їх адреси, які передаються в підпрограму. Приклад:

```
cout<<sum1(5,6);//5,6 - фактичні параметри
```

Передача параметрів по значенню – формальному параметру присвоюється копія значення аргументу (фактичного параметра). Приклад:

```
cout<<sum1(a=5,b=6);//передача параметрів по значенню
```

Передача параметрів по посиланню – формальному параметру присвоюється адреса аргументу. У цьому випадку зміна формального параметру впливає на аргумент. Приклад:

```
cout<<sum2(&a,&b);//передача параметрів по посиланню
```

Локальна змінна – володіє областю видимості тільки в тій функції або блоці команд де вона оголошена (с.42).

Глобальна змінна - володіє областю видимості в усій програмі (с.42).

Статична змінна (static) – зберігає своє значення після виходу з функції де вона оголошена (с.42). Приклад:

```
static int k=0;
```

Простір імен (namespace) – область видимості імен ідентифікаторів, яка використовується для запобігання конфлікту між однаковими іменами. Приклад:

```
namespace My {int x=1;}//простір імен My
void main()
{cout<<My::x;//вказати область видимості змінної x
using namespace My;//використовувати простір імен My
cout<<x;}
```

Головна функція (main) – функція, з якої починається виконання програми (с.31). Приклад пустої функції:

```
void main() {}
```

Вбудовувана функція (inline) – функція, код якої вкладається в кожній точці виклику. Приклад:

```
inline int s(int x){return x+2;}
```

Рекурсія – алгоритмічна конструкція, в якій підпрограма викликає саму себе. Приклад:

```
long int fact(int x)//рекурсивна функція визначення факторіалу
{if (x>1) return x*fact(x-1);else return 1;}
```

Масив – іменований впорядкований набір даних одного типу, які зберігаються в послідовно розташованих комірках пам'яті (с.39). Масив складається з елементів, кожен з яких має унікальний індекс. Масиви можуть бути одновимірні і багатовимірні. Приклад:

```
float A[3];//масив А дійсних чисел розміром 3
A[0]=2.5;//присвоїти елементу масиву А з індексом 0 число 2.5
```

Рядки – дані або набір даних, які містять сукупність символів. Наприклад, масиви символів (с.43) або рядкові контейнерні класи. Наприклад:

```
char s[20]="cpp";//масив символів
#include <string>
string s("hello");//об'єкт s класу string
```

Помилки програмування поділяються на помилки компіляції, помилки виконання та логічні помилки. Засобом опрацювання помилок виконання і логічних помилок є: налагоджувач, засоби обробки виключних ситуацій мови програмування, зміна методики програмування, тестування програм.

Помилки компіляції виникають на етапі компіляції програми і виявляються компілятором. Приклади:

```
inte x=1;//невизначений символ inte
int x:=1;//синтаксична помилка
```

Помилки виконання, або виключні ситуації (exception) виникають на етапі виконання програми та призводять до її аварійного завершення або неправильного функціонування. В мові С++ обробляються за допомогою конструкції `try-catch-throw` (с.45).

Логічні помилки виникають на етапі виконання програми та не призводять до її аварійного завершення, але до неправильного функціонування. Виявляються важко, за допомогою засобів налагодження і тестування.

1.3 Основи об’єктно-орієнтованого програмування

Рекомендована література: [15-61]

Процедурне програмування – програмування основане на використанні підпрограм, що дозволяє підвищити ефективність програмування відносно складних програмних систем (с.41).

Модульне програмування – програмування основане на використанні модулів – множини підпрограм з даними, які ці підпрограми обробляють.

Структурне проектування - методологія проектування, основана на алгоритмічній декомпозиції.

Об’єктно-орієнтоване проектування – це методологія проектування, яка поєднує в собі процес об’єктної декомпозиції і прийоми представлення логічної і фізичної, а також статичної і динамічної моделей системи, що проектується.

Об’ктно-орієнтоване програмування (ООП) –програмування основане на використанні об’єктів, які є абстрактними моделями реальних об’єктів. Основні принципи ООП: абстрагування, інкапсуляція, успадкування, поліморфізм. Об’єктно-орієнтований підхід у програмуванні реалізується за допомогою спеціальних типів даних – класів (с.46) і ефективно використовується для розробки складних програмних систем.

Основні переваги ООП:

- можливість створення абстрактних програмних моделей предметів і понять і ієрархій їх систем.
- зменшення трудомісткості програмування складних програм, забезпечення можливості повторного використання окремих програмних компонентів, спрощення модифікації програмних компонентів.

Об’єктно-орієнтовані мови програмування – мови програмування високого рівня, побудовані на принципах ООП: Simula, Smalltalk, Ada, CLOS, C++, Object Pascal, Java, C#, Visual Basic .NET, Python та інші.

Абстрагування полягає в тому, що модель (клас) містить не всі ознаки і властивості предмету (поняття), що представляється нею, а тільки ті, які істотні для програмної системи, що розробляється.

Інкапсуляція – об’єднання даних і алгоритмів їх опрацювання в класі (с.46).

Композиція – включення в поля класу об’єктів інших класів (с.50).

Успадкування (inheritance) – успадкування дочірнім класом доступних (не закритих (**private**)) полів і методів батьківського класу таким чином, що вони стають членами цього дочірнього класу (с.54).

Закрите успадкування (private inheritance) – усі відкриті і захищені члени базового класу стають закритими членами похідного класу. Використовується по замовчуванню. Приклад:

```
class B:private A {};
```

Відкрите успадкування (public inheritance) – не змінює права доступу у похідному класі. Приклад:

```
class B:public A {};
```

Захищене успадкування (protected inheritance) - усі відкриті члени базового класу стають захищеними членами похідного класу. Приклад:

```
class B:protected A {};
```

Множинне успадкування – безпосереднє успадкування дочірним класом кількох базових (с.58). Приклад:

```
class C:public A, public B {};
```

Поліморфізм – можливість функцій з однаковим ім'ям або методів з однаковим ім'ям батьківського і дочірнього класів виконувати різні команди. В основі **статичного поліморфізму** лежить механізм перевантаження функцій і операторів (с.55). В основі **динамічного поліморфізму** лежить принцип успадкування та механізм віртуальних функцій, які викликаються через вказівник або посилання (с.56).

Структура (struct) - похідний тип даних, який використовується для опису об'єктів, які можуть складатися з даних різного типу (с.47). Подібним на структуру типом є клас. Приклад структури A:

```
struct A
{int x;
float y;};
```

Об'єднання (union) - похідний тип даних, який використовується для опису об'єктів, які в кожен момент часу містять дані різного типу (с.47).

Клас (class) – похідний тип даних, який використовується для опису об'єктів, в яких дані різного типу об'єднані з алгоритмами їх опрацювання (функціями) (с.46). Клас може описувати також набір обмежень на доступ до цих даних і функцій:

```
class <назва класу>: <специфікатор доступу успадкування> <назва батьківського класу>
{
<специфікатор доступу>:
<тип поля> <назва поля>;
...;
<тип методу> <назва методу(<формальні параметри>)>
{
<реалізація алгоритму методу>;
}
```

```
...;
...;
}
```

Члени класу - поля і методи класу (с.46).

Поля класу – дані, які є членами класу.

Методи класу – функції, які є членами класу, використовуються для операцій над об'єктами. Метод класу може бути визначений в класі, або зовні класу. В останньому випадку в класі повинен бути оголошений прототип методу. Приклад визначення методу `print` класу `A` поза класом:

```
void A::print()
{cout<<b;}
```

Об'єкт (екземпляр класу) – дане певного класового типу (с.46). Об'єкту властиві стан, поведінка і ідентичність. Приклад:

```
A obj;//об'єкт obj класу A
```

Конструктор – метод класу (функція), який має ім'я класу і використовується для створення екземплярів класу (с.47). Викликається при створенні об'єкту. Приклад визначення конструктора класу A:

```
A::A(int x)
{a=x;}//ініціалізація поля a параметром x
```

Конструктор копії – конструктор, який використовує в якості аргументу екземпляр класу, для якого цей конструктор визначений (с.49). Використовується для створення копій об'єкта. Приклад:

```
A::A(A &obj)
{a=obj.a;}//ініціалізація поля a параметром obj.a
```

Деструктор - метод класу (функція), який має ім'я класу зі символом ~ і використовується для знищення екземплярів класу (с.48). Викликається автоматично при знищенні об'єкту. Приклад:

```
A::~A()
{cout<<"destructor";}
```

Специфікатори доступу членів класу (public, private, protected) використовуються для визначення доступності членів класу (с.54):

`public` (відкритий) - доступність для будь-якої зовнішньої функції;

`private` (закритий) - доступність тільки для методів класу, використовується по замовчуванню;

`protected` (захищений) - доступність тільки для методів класу і похідних класів.

Абстрактні класи - це класи, які містять хоча б одну чисто віртуальну функцію і, тому, не можуть мати екземплярів. Використовуються тільки для породження похідних класів. Приклад:

```
class A//абстрактний клас A
{
public: int a;
virtual void f()=0;//чисто віртуальний метод
```

```
};
```

Інтерфейс – зовнішній вигляд класу, об'єкту або модуля, що виділяє його істотні риси і не показує внутрішнього устрою і деталей поведінки. У C++ це абстрактний клас, який містить тільки оголошення методів і статичні константні поля.

Реалізація – на відміну від інтерфейсу - внутрішнє представлення класу, об'єкту або модуля, включаючи деталі його поведінки.

Ієрархія класів – сукупність класів побудована на основі принципу успадкування (с.54). Клас, який лежить в основі ієрархії називається **базовим (батьківським)**, а клас, який його успадковує називається **похідним (дочірнім)**. Приклад ієрархії:

```
class A {};
class B:public A {};
class C:public B {};
```

Віртуальні (узагальнені) методи (функції) – методи, які оголошуються в базовому класі за допомогою слова `virtual` та використовуються для реалізації принципу динамічного поліморфізму (с.56). Віртуальна функція може бути перевизначена в похідних класах, отже, її реалізація визначається множиною методів, оголошених у всіх класах дерева успадкування. Приклад:

```
virtual void f()
{cout<<a;}
```

Таблиця віртуальних методів – таблиця вказівників на методи, яка існує для кожного класу. Всі екземпляри класу містять вказівник на цю таблицю.

Чисто віртуальна функція – метод класу, який не має реалізації. Приклад:

```
virtual void f()=0;
```

Перевантаження функцій-членів класу полягає в тому, що декілька різних функцій-членів використовують одне і теж ім'я, при цьому, кожне перевизначення функції повинно використовувати інші типи параметрів або їх кількість (с.55). Приклад перевантаження функції `s`:

```
void s(int x){cout<<x;}
void s(int x,int y){cout<<x+y;}
```

Дружні функції – функції, які не є членами класу але володіють усіма правами члена класу і оголошуються в класі за допомогою слова `friend` (с.53). Мають доступ до закритих і захищених полів класу. Приклад:

```
friend void f();
```

Статичні поля – змінні члени класу, які оголошуються за допомогою слова `static` і належать цьому класу, а не його об'єктам. Їх визначення повинно знаходитись поза класом. Приклад:

```
class A
```

```
{static int a;};//оголошення статичного поля a
int A::a=0;//визначення поля a
```

Статичні методи – функції члени класу, які оголошуються за допомогою слова static, і можуть звертатись тільки до статичних полів і методів. Можуть викликатись як звичайні функції: <клас>::<метод>. Приклад:

```
class A
{public: static int a;//статичне поле
 static void f(){cout<<a;};//статичний метод
};
```

Константні методи - функції члени класу, які оголошуються за допомогою слова const, і не можуть змінювати значення полів класу. Приклад:

```
class A
{public: int a;
 void f() const {cout<<a;};//константний метод
};
```

Вказівник this – псевдозмінна-вказівник, який використовується в методі класу і вказує на об'єкт, який викликав цей метод. Приклад:

```
void A::f(){this->a=1;}
```

Перевантаження операторів основане на перевантаженні операторних функцій, які створюються за допомогою слова operator і визначають дії перевантажених операторів стосовно певного класу (с.56). Приклад:

```
A A::operator+(const A& obj)
{return A(a+obj.a);}//повернути об'єкт, у якого поле
a є сумою полів доданків
```

Шаблони (template) – механізм реалізації статичного поліморфізму, який дозволяє створювати узагальнені функції і класи, які працюють з типом даних, який заданий як параметр (с.62). Таким чином, одну і ту ж функцію, або клас можна застосовувати до різних типів даних, не використовуючи окремі варіанти для кожного типу. Конкретна версія шаблонної функції або класу створюється на етапі компіляції і називається **спеціалізацією**. Шаблони оголошуються за допомогою слова template. Приклад шаблону функції:

```
template <typename T>//список шаблонних параметрів
T sum(T a, T b){return a+b;};
```

Узагальнене програмування основане на використанні шаблонів і призначене для створення програмних компонентів, які не залежать від типу даних.

1.4 Сучасні об'єктно-орієнтовані технології

Рекомендована література: [62-112]

Стандартна бібліотека C++ – об'єктно-орієнтована бібліотека програмних компонентів загального призначення для виконання операцій вводу-виводу (бібліотека потокового вводу-виводу IOStream), роботи з рядками, контейнерами (структурами даних), алгоритмами, які обробляють вміст контейнерів (бібліотека STL), для математичних обчислень, підтримки інтернаціоналізації [62-65].

Стандартна бібліотека шаблонів STL (Standard Template Library) – це набір узагальнених програмних компонентів (шаблонних класів і функцій), які реалізують велику кількість широко розповсюджених алгоритмів і структур даних [62-65] (с.70). STL основана на використанні таких компонентів: алгоритм, контейнер, ітератор, функціональний об'єкт, адаптер.

Алгоритм STL (algorithm) - шаблон функції, який описує універсальні обчислювальні процедури над контейнерами. Приклад алгоритму `fill`:

```
#include <algorithm>//бібліотека функцій-алгоритмів
fill(v.begin(),v.end(),7); //заповнити контейнер v
числами 7
```

Контейнер STL (container) - структура даних, яка описана у вигляді шаблонного класу. Приклад:

```
#include <vector>//бібліотека контейнера вектора
vector<int> v(5);//оголосити контейнер-вектор цілих
чисел розміром 5
```

Ітератор STL (iterator) - узагальнений вказівник, який забезпечує доступ до елементів контейнера. Приклад:

```
#include <vector>//бібліотека контейнера вектора
vector<int>::iterator it;//оголосити ітератор
```

Функціональний об'єкт STL (function object) - шаблонний клас, який інкапсулює перевантажений оператор виклику функції.

Адаптер STL (adaptor) - шаблонний клас, який забезпечує відображення інтерфейсу (адаптує компонент для забезпечення різного інтерфейсу).

Бібліотека потокового вводу-виводу IOStream – бібліотека шаблонних класів для операцій вводу-виводу, файлового вводу-виводу і рядкового вводу-виводу (с.64). Класи містять конструктори і деструктори, поля і функції для форматування, стану і режиму використання потоку, функції і оператори вводу-виводу, функції визначення і установки позиції в потоці та інші члени.

Потік вводу-виводу – абстракція фізичного пристрою вводу-виводу, яка забезпечує однотипний інтерфейс роботи з ним. Наприклад, `cin` - стандартний потоковий об'єкт вводу (клавіатура), `cout` – виводу (монітор). Робота з фізичним пристроєм за допомогою потоків є буферизованою.

Оператори потокового вводу-виводу – перевантажені в бібліотеці IOStream оператори вводу даних в потік << і виводу даних з потоку >> (с.66). Приклади:

```
cin>>x;//вивід з стандартного потоку вводу
cout<<x;//ввід у стандартний потік виводу
```

Флагі форматування – в бібліотеці IOStream члени дані класу ios, які призначені для форматування потоку вводу-виводу (с.64). Наприклад, установка в потоці виводу флагу `hex`:

```
cout.flags(ios::hex);//виводити в шістнадцятковому
форматі
```

Маніпулятор – в бібліотеці IOStream спеціальна функція, яка керує станом потоку вводу-виводу, отримує і повертає посилання на потоковий об'єкт (с.65). Наприклад маніпулятор `hex`:

```
#include <iomanip.h>//бібліотека маніпуляторів
cout<<hex<<16<<endl;//вивести в шістнадцятковому
вигляді 16
```

Середовище розробки (framework) - набір програмних компонентів (часто класів), що надають деякі базові послуги в певній області. Таким чином, середовище розробки експортує класи і механізми, які клієнти можуть використовувати або адаптувати.

Borland C++ Builder - інтегроване середовище розробки основане на візуальному об'єктно-орієнтованому програмуванні мовою C++ і призначене для розробки програмного забезпечення для Windows і Linux, розподіленого програмного забезпечення з використанням технологій COM і CORBA, систем роботи з базами даних, Internet і т.п. Однією з основних частин C++ Builder є бібліотека візуальних компонентів VCL [66-83].

Бібліотека візуальних компонентів VCL (Visual Component Library) – об'єктно-орієнтована бібліотека програмних компонентів (класів) загального призначення (framework), яка використовується в Borland C++ Builder і Delphi [66-83] (с.73). В бібліотеку входять класи для створення графічного інтерфейсу програми, класи для доступу до баз даних, класи, в яких інкапсульовані різноманітні функції Win32API, службові та інші класи. Класи VCL утворюють ієрархію зі спільним базовим класом `TObject`.

Компонент VCL – програмний компонент (клас бібліотеки VCL, базовий від `TComponent`), який використовується для розробки додатків

Windows. Компоненти можна умовно поділити на службові та елементи керування. Компонент VCL володіє властивостями, методами і подіями.

Елементи керування VCL – видимі компоненти VCL (базовий клас - `TControl`), які використовуються для побудови графічного інтерфейсу програми з користувачем (с.87). Бувають віконні (походять від `TWinControl`) і графічні (походять від `TGraphicControl`). Віконні елементи являють собою вікно ОС Windows, можуть отримувати фокус вводу і служити контейнерами для інших елементів, але використовують більше системних ресурсів. Приклад динамічного створення компоненту класу `TButton`:

```
TButton *Btn=new TButton(Panel1);//динамічно створити кнопку
```

Властивості компоненту VCL (property) – члени класу компоненту, які визначають певні характеристики об'єкту. На відміну від полів (членів даних) класу властивості самі не зберігають дані, а можуть використовувати методи читання і запису у відповідні захищені поля класу, чим забезпечується їх перевага (с.126). Приклад використання властивості `Text` об'єкту `Edit1`:

```
Edit1->Text="Hello";//вивести рядок в текстове поле
```

Методи компоненту VCL (method) – функції-члени класу компоненту, які виконують певні обчислювальні процедури над об'єктом (с.126). Приклад використання методу `Close` об'єкту `Form1`:

```
Form1->Close();//закрити форму
```

Подія компоненту VCL (event) – типізований вказівник на певний метод в екземплярі класу (с.126). Подія дозволяє компоненту реагувати на певний вплив (наприклад, натиск кнопкою миші на ньому) і пов'язує цей вплив з певним кодом (функцією обробки події). Для визначення подій використовуються властивості. Більшість стандартних подій (`OnKeyDown`, `OnMouseMove`, `OnMouseDown`) відповідають повідомленням Windows (Windows Message). Приклад:

```
//вказати функцію-обробник події OnClick об'єкту Button1
 Button1->OnClick=Button1Click;
```

Інтерфейс програмування Win32 API (Application Programming Interface) – інтерфейс програмування додатків для 32-х розрядних версій Windows, який описує сукупність низькорівневих системних функцій і структур загального призначення, розміщених у файлах dll ОС Windows.

Паралелізм - означає одночасне виконання декількох програм на одній ЕОМ.

Паралельна програма складається з одного або декількох програмних компонентів (процесів), які можуть виконуватися паралельно.

Виконання паралельної програми визначається як будь-яке чергування атомарних (нероздільних) команд завдань.

Потік керування (thread) – окремий процес. Запуск потоку керування приводить до виникнення незалежної динамічної діяльності в системі; дана система може мати декілька одночасно виконуваних потоків, деякі з яких можуть динамічно виникати і знищуватися. У VCL для створення потоків використовується клас `TThread`.

Синхронізація використовується для контролю виконання паралельних процесів, наприклад, два процеси не можуть звертатись до одного ресурсу одночасно.

Бібліотека динамічної компоновки DLL (Dynamic-link Library) - набір програмних компонентів (функцій, класів) і ресурсів, які упаковані в один пов'язуваний бінарний модуль, і які можуть динамічно (на етапі виконання) використовуватись кількома програмами (с.129). DLL основані на механізмі підтримки повторного використання програмних компонентів на рівні бінарних файлів у Windows.

Модель компонентних об'єктів COM (Component Object Model) об'єктно-орієнтована технологія об'єднання універсальних бінарних програмних компонентів (об'єктів) для Windows [84-86] (с.130). Підтримує повторене використання і можливість взаємодії об'єктів незалежно від мови програмування на якій вони були розроблені (на відміну від DLL) і включає сервери, клієнти, інтерфейси, елементи керування ActiveX, технологію зв'язування і упровадження об'єктів OLE і автоматизацію.

Об'єкт COM – екземпляр класу COM в сервері COM.

Сервер COM - програма-сервер, яка організовує доступ до створеного в ній об'єкту COM, реалізуючи інтерфейси.

Клієнт COM - програма-клієнт, яка, використовуючи інтерфейс, отримує доступ до об'єкту COM.

Інтерфейс COM – інтерфейс, який визначає відкриті методи, які використовуються для доступу до об'єкту COM.

Клас COM (CoClas) – реалізація інтерфейсу COM в сервері COM.

Бібліотека типу (type library) COM – бінарний файл, який використовується для визначення типів: інтерфейсів, диспінтерфейсів, класів COM і інших елементів COM. Вміст бібліотеки типу аналогічний вмісту заголовного файлу C++, але до неї можливий доступ незалежно від мови програмування, оскільки бібліотека типу - це відкомпільована версія файлу IDL.

Елемент керування ActiveX (OLE) – внутрішньопроцесний сервер COM у вигляді бінарного програмного модуля (DLL), який являє собою універсальний програмний компонент, який може інтегруватися в існуючі додатки (клієнти COM) для розширення їх можливостей (с.132). Елемент ActiveX володіє властивостями, методами і подіями.

OLE Automation (OLE автоматизація) – механізм, оснований на COM, який забезпечує програмовану модель функціональних можливостей додатку незалежну від мови реалізації, автоматично визначає стандартні типи даних і забезпечує перетворення типів і транспортування всіх необхідних параметрів, забезпечує стандартний механізм, за допомогою якого керуюча програма може проаналізувати і ідентифікувати методи і властивості інтерфейсу. Автоматизація передбачає диспетчерські інтерфейси (`IDispatch`), сервери і клієнти (контролери) автоматизації, якими можуть бути мови програмування Visual Basic, VBA, VBScript, Java.

Бібліотека активних шаблонів ATL (Active Template Library) – бібліотека програмних компонентів, яка використовується для полегшення розробки компонентів COM.

Модель розподілених компонентних об'єктів DCOM (Distributed COM) - це розвиток моделі COM для розподілених програмних компонентів, у якій підтримується об'єктний протокол, який дає можливість прямого доступу об'єктів COM один до одного за допомогою мережі.

Архітектура посередника запитів типового об'єкта CORBA (Common Object Request Broker Architecture) об'єктно-орієнтована технологія об'єднання програмних компонентів для побудови розподілених додатків (с.135). Підтримує можливість взаємодії об'єктів незалежно від мови програмування, на якій вони були розроблені, і архітектури, на якій вони використовуються, і включає сервери з реалізацією об'єкту (Skeleton), клієнти з замісником об'єкту (Stub), інтерфейси і брокери об'єктних запитів (ORB) [41].

Брокер об'єктних запитів (ORB) - архітектурний елемент CORBA, який визначає правила взаємодії об'єктних компонентів один з одним, а також дозволяє змішувати компоненти. ORB - це типовий серверний додаток, що опитує певний порт і чекає доступу в процесі з'єднання каналу TCP/IP. Клієнтські додатки з'єднуються з ORB, посилають повідомлення про запити і отримують у відповідь повідомлення від ORB. Основна функція ORB - ініціалізувати посередницькі запити для виклику методу.

Мова опису інтерфейсу IDL (Interface Definition Language) – використовується для визначення інтерфейсів з метою уникнути його залежності від мови програмування, яка визначає реалізацію (с.135).

Бібліотека основних класів Microsoft MFC (Microsoft Foundation Classes) – об'єктно-орієнтована бібліотека програмних компонентів (класів) загального призначення (framework), яка використовується в середовищі розробки Microsoft Visual C++ (с.139). В бібліотеку входять класи C++ для створення графічного інтерфейсу програми, класи для доступу до баз даних, класи, в яких інкапсульовані різноманітні функції Win32API, службові та інші класи.

Java – об'єктно-орієнтована мова програмування, призначена в основному для створення архітектурно-незалежного і розподіленого програмного забезпечення. Архітектурна незалежність досягається шляхом інтерпретації скомпільованого байт-коду на віртуальній машині Java.

JavaBeans – програмні компоненти у вигляді Java-класів. Компоненти володіють властивостями, методами і подіями.

.NET – об'єктно-орієнтована технологія об'єднання програмних компонентів, яка забезпечує загальномовне середовище виконання CLR (common language runtime), загальну систему типів CTS (common type system), бібліотеку класів .NET Framework і призначена, в основному, для створення архітектурно-незалежного, мовно-незалежного і розподіленого програмного забезпечення [98-112] (с.149). Загальномовне середовище виконання основане на використанні проміжної мови IL (intermediate language) і її компіляції безпосередньо перед виконанням.

.NET Framework – бібліотека класів загального призначення, яка використовується для платформи .NET (с.149).

C# - об'єктно-орієнтована мова програмування, призначена в основному для створення програмного забезпечення в рамках платформи .NET [98-112].

Сбір сміття – технологія розподілу пам'яті, яка в процесі виконання програми автоматично визначає втрачені блоки пам'яті і робить їх вільними. Використовується в сучасних мовах програмування, наприклад, у Java і C#.

Уніфікована мова моделювання UML (Unified Modeling Language) є графічною мовою для візуалізації, специфікації, конструювання і документування систем, в яких велика роль належить, програмному забезпеченню. За допомогою UML можна розробити детальний план створюваної системи, що містить не тільки її концептуальні елементи, такі як системні функції і бізнес-процеси, але і конкретні особливості, наприклад класи, написані на спеціальних мовах програмування, схеми баз даних і програмні компоненти багатократного використання.

Перспективи розвитку об'єктно-орієнтованих технологій у програмуванні – створення програмних компонентів високого рівня абстракції, незалежних від архітектури і мови програмування для побудови розподілених додатків.

2 Основи програмування мовою С++

2.1 Найпростіша програма

```
//директива підстановки файлу iostream.h,
//в якому описаний об'єкт cout і оператор <<
#include <iostream.h>
void main()//головна функція
{
cout<<"Hello world!"; //команда виводу рядка на екран
}
```

2.2 Директиви препроцесора

```
//директива підстановки файлу iostream.h
#include <iostream.h>
//директива визначення макросу K
#define K 15
//директива визначення макросу F
#define F(x) x*x+K
//директива підстановки файлу File1.h
#include "File1.h"
//головна функція
void main()
{
x=2;
prn;//макропідстановка
//директива підстановки файлу File2.h
#include "File2.h"
prn;//макропідстановка
}


// File1.h
//директива визначення макросу prn
#define prn cout<<F(x)<<endl
float x;//оголошення глобальної змінної


// File2.h
//директива видалення визначення макросу F
#undef F
//директива визначення макросу F(x)
#define F(x) x*x+2
```

2.3 Основні типи даних С++

```
#include <iostream.h>
void main()
{
//оголошення і ініціалізація констант
const k1=7,k2=9;
const int j=3;
const float pi=3.14;
//оголошення змінних
int i=-2,l;//цілого типу
short int n;//короткого цілого типу
unsigned int m;//беззнакового цілого типу
float x;//дійсного типу
double y,z=3.25e-2;//дійсного типу подвійної точності
char c1,c2='B';//символьного типу
bool b1=true,b2;//булевого (логічного) типу
typedef unsigned short int uint;//визначення типу
uint
uint u;//оголошення змінної типу uint
//присвоєння змінним значень
i=5;
l=-12;
n=3;
m=17;
x=25.751;
y=383.321;
c1='A';
c1=65;
c1=0x41;
cout<<c1;
b2=false;
u=13;
}
```

2.4 Граничні значення і розмір змінних різних типів

```
#include <iostream.h>
#include <values.h>
void main()
{
//вивід граничних значень і розміру змінних різних
типів
```

```
cout<<MAXINT<<" "<<sizeof(int)<<endl;
cout<<MAXSHORT<<" "<<sizeof(short)<<endl;
cout<<MAXLONG<<" "<<sizeof(long)<<endl;
cout<<MAXFLOAT<<" "<<sizeof(float)<<endl;
cout<<MINFLOAT<<endl;
cout<<MAXDOUBLE<<" "<<sizeof(double)<<endl;
cout<<MINDOUBLE<<endl;
}
```

2.5 Приведення (перетворення) типів

```
#include <iostream.h>
void main()
{
int i=1,k=2;//оголошення цілих змінних
double x,y=1.2;//оголошення дійсних змінних
x=i/k;//x=0
x=(double)i/k;//приведення i до double, x=0.5
x=static_cast<double>(i)/k;//приведення  i до double,
x=0.5
i=k*y;//i=2
char c='A';
i=c;
i=(int)c;//приведення c до int
}
```

2.6 Арифметичні оператори С++

```
#include <iostream.h>
void main()
{
//оголошення  і ініціалізація констант
const float a=1.5,b=0.7;
//оголошення і ініціалізація змінних
int i=1;
float x=2,y;
y=-i*(a+b)/x+b*(x+1);//присвоїти значення виразу
y=9%4;//остача від ділення 9 на 4
y=x*++i;//збільшити і на 1 і обчислити вираз
y=x*i++;//обчислити вираз і збільшити і на 1
x+=0.5;//додати до x 0.5
x*=2;//домножити x на 2
```

```
y=x=a+1;//множинне присвоювання
}
```

2.7 Стандартні математичні функції

```
#include <iostream.h>
#include <math.h>//заголовочний файл математичних
функцій
void main()
{
int i=-3;
double x=0.35,y;
y=abs(i);//модуль
y=fabs(x);//модуль
y=cos(x);//косинус
y=sin(x);//синус
y=tan(x);//тангенс
y=log(x);//логарифм натуральний
y=log(x)/log(2);//логарифм з основою 2
y=pow(x,3);//x в степені 3
y=sqrt(x);//корінь квадратний
y=exp(x);//e в степені x
y=pow10(i);//10 в степені i
y=log10(x);//логарифм з основою 10
y=acos(x);//арккосинус
y=asin(x);//арксинус
y=atan(x);//арктангенс
y=ceil(x);//заокруглення до більшого цілого
y=floor(x);//відкидає дробову частину
y=fmod(1,x);//остача від ділення 1 на x
y=M_E;//основа натурального логарифму
y=M_PI;//число Пі
}
```

2.8 Команди потокового вводу-виводу

```
#include <iostream.h>//підключити класи вводу-виводу
void main()
{
char c;
char s[20];
int i;
```

```
float x;
double y;
cin>>i;//ввести значення і
cout<<i<<"\n";//вивести значення і та перейти на
новий рядок
cin>>c>>s>>x>>y;//ввести c,s,x,y
cout<<c<<s<<x<<y<<endl;//вивести c,s,x,y та перейти
на новий рядок
cout<<"s="<<s<<endl;//вивести s=, s та перейти на
новий рядок
cin.get();//або cin>>ws;
cin.getline(s,20);//ввести s з символами пропуску
cout<<s;//вивести s
}
```

2.9 Стандартні функції вводу-виводу мови С

```
#include <stdio.h>//стандартна бібліотека вводу-
виводу
#include <conio.h>//бібліотека для getch() і getche()
void main()
{
char c;
c=getchar();//ввести символ
putchar(c);//вивести символ
c=getch();//ввести символ без відображення
c=getche();//ввести символ з відображенням
char s[20];
gets(s);//ввести рядок
puts(s);//вивести рядок
int i;
float x;
double y;
//форматований ввід c,s,i,x,y
scanf("%c%s%d%f%lf",&c,&s,&i,&x,&y);
//форматований вивід c,s,i,x,y з переходом на новий
рядок
printf("%c %s %d %f %lf\n",c,s,i,x,y);
printf("x=%5.1f\n",x);//вивід x у 5 цифрах з 1 після
коми
printf("y=%lf%s\n",y," mm");//вивід значення y і
рядку " mm"
}
```

2.10 Оператор безумовного переходу goto

```
#include <iostream.h>
void main()
{
float x,y=0;
cin>>x;
//якщо x>0 перейти на мітку a, інакше перейти на
мітку b
if (x>0)goto a;else goto b;
//мітка a
a:y=3;
//мітка b
b:cout<<y;
}
```

2.11 Оператор умовного переходу if-else

```
#include <iostream.h>
void main()
{
float a,x,y;
cin>>x>>a;
//якщо x>2 і x<3 то y=x*a
if (x>2&&x<3)y=x*a;
//інакше, якщо x>=3 то a=3;y=x+a
else if (x>=3){a=3;y=x+a;}
//інакше y=a
else y=a;
cout<<y;
}
```

2.12 Оператор switch

```
#include <iostream.h>
void main()
{
int x;
float y;
cin>>x;
switch (x)
{
```

```
case 1:y=x;break;//якщо x=1 то y=x
case 2:y=x*x;break;//якщо x=2 то y=x*x
case 3:y=x*x*x;break;//якщо x=3 то y=x*x*x
default: y=0;//у інших випадках y=0
}
cout<<y;
}
```

2.13 Тернарний оператор ?:

```
#include <iostream.h>
void main()
{
float x,y;
cin>>x;
//якщо x>2 або x==0 то y=x*x, інакше y=x*x+2
y=(x>2||x==0)?x*x:x*x+2;
cout<<y;
}
```

2.14 Цикл з лічильником for

```
#include <iostream.h>
void main()
{
float x,y;
//повторювати від 0 з кроком 0.1 поки x<2
for(x=0;x<2;x+=0.1)
{
y=x*x;
cout<<y<<endl;
}
}
```

2.15 Оператор циклу з передумовою while

```
#include <iostream.h>
void main()
{
float x=0,y;
while (x<2)//поки x<2 повторювати цикл
{
```

```
y=x*x;
cout<<y<<endl;
x+=0.1;
}
}
```

2.16 Оператор циклу з післяумовою do-while

```
#include <iostream.h>
void main()
{
float x=0,y;
do //виконувати цикл
{
y=x*x;
cout<<y<<endl;
x+=0.1;
}while (x<2);//поки x<2
}
```

2.17 Оператори break і continue

```
#include <iostream.h>
void main()
{
float x,y;
//цикл
for(x=0;x<2;x+=0.1)
{
y=x*x;
//якщо y>=2 то вийти з циклу, інакше - продовжити
if (y>=2)break;else continue;
}
cout<<x<<endl;
}
```

2.18 Вказівники і адреси даних

```
#include <iostream.h>
void main()
{
```

```
int i=2;//оголошення і ініціалізація змінної і цілого
типу
int *pi;//оголошення вказівника pi цілого типу
pi=&i;//вказівник указує на адресу змінної і
*pi=3;//записати за адресою pi число 3
cout<<pi<<endl;//адреса змінної і
cout<<*pi<<endl;//значення по адресі pi, дорівнює 3
cout<<i<<endl;//значення змінної і, дорівнює 3
}
```

2.19 Оператори динамічного розподілу пам'яті new і delete

```
#include <iostream.h>
void main()
{
//виділити пам'ять для збереження цілого числа
//вказівник pi вказує на цю пам'ять
int *pi=new int;
*pi=2;//записати в цю пам'ять число 2
//виділити пам'ять для збереження дійсного числа
//і записати в неї 3.1
float *px=new float(3.1);
//вивести вміст пам'яті, на яку вказує pi і px
cout<<*pi<<endl;
cout<<*px<<endl;
delete pi;//вивільнити пам'ять, на яку вказує pi
delete px;//вивільнити пам'ять, на яку вказує px
}
```

2.20 Масиви

```
#include <iostream.h>
void main()
{
//оголошення масиву дійсних чисел A розміром 3
float A[3];
//оголошення і ініціалізація масиву цілих чисел B
розміром 3
int B[3]={1,3,5};
//оголошення і ініціалізація масиву дійсних чисел C
float C[]={1.1,0.2,3.5};
```

```
//оголошення і ініціалізація масиву цілих чисел D
розміром 2x2
int D[2][2]={4,5,1,3};
A[0]=1.2;//присвоєння першому елементу масиву A 1.2
A[1]=1.7;//присвоєння другому елементу масиву A 1.7
A[2]=0.5;
D[0][1]=2;
//вивід вмісту масивів A,B,C
for(int i=0;i<3;i++)
cout<<A[i]<<" "<<B[i]<<" "<<C[i]<<endl;
//вивід вмісту масиву D
for(int i=0;i<2;i++)
{
for(int j=0;j<2;j++)cout<<D[i][j]<<" ";
cout<<endl;
}
}
```

2.21 Динамічні масиви

```
#include <iostream.h>
void main()
{
//динамічно виділити пам'ять
//для масиву цілих чисел A розміром 3
int *A=new int[3];
//заповнити масив числами 0,1,2 і вивести його
for(int i=0;i<3;i++)
{
*(A+i)=i;//присвоїти елементам масиву і
cout<<A[i]<<endl;
}
//динамічно виділити пам'ять
//для масиву *B розміром 2
int **B=new int*[2];
int k=0;
//заповнити двовимірний динамічний масив 2x3
for(int i=0;i<2;i++)
{
//динамічно виділити пам'ять
//для масиву B[i] розміром 3
B[i]=new int[3];
for(int j=0;j<3;j++)
```

```
{
*(B[i]+j)=k++;//присвоїти елементам масиву k
cout<<*(B[i]+j)<<" ";//вивести елемент
//або cout<<*(*(B+i)+j)<<" ";
//або cout<<B[i][j]<<" ";
}
delete[]B[i];//вивільнити пам'ять для масиву B[i]
cout<<endl;
}
delete[]A;//вивільнити пам'ять для масиву A
delete[]*B;//вивільнити пам'ять для масиву *B
}
```

2.22 Перерахований тип enum

```
#include <iostream.h>
void main()
{
//перерахований тип days
enum days {sun=1,mon,tues,wed,thur,fri,sat};
int day;
//вивести 1234567
cout<<sun<<mon<<tues<<wed<<thur<<fri<<sat<<endl;
cin>>day;
if (day==sun||day==sat) cout<<"weekend";
else cout<<"not weekend";
}
```

2.23 Створення функцій C++

```
#include <iostream.h>
int sum1(int a,int b=0);//оголошення функції sum1
int sum2(int *a,int *b) //оголошення функції sum2
//параметри функції - вказівники
{
int c=*a+*b;//оголошення локальної змінної c
*a=2;*b=2;//зміна аргументів
return c;//функція повертає значення с цілого типу
}
int sum3(int &a,int &b) //оголошення функції sum3
```

```
//параметри функції - посилання
{
int c=a+b;//оголошення локальної змінної c
a=3;b=3;//зміна аргументів
return c;//функція повертає значення c цілого типу
}
void main() //головна функція
{
cout<<sum1(5,6)<<endl;//передача аргументів по
значенню, результат=11
cout<<sum1(5)<<endl;//передача аргументів по
значенню, результат=5
int a=1,b=2; //оголошення локальних змінних
int *pa=&a,*pb=&b;//оголошення вказівників на ці
змінні
cout<<sum2(pa,pb)<<endl;//передача аргументів по
посиланню, результат=3
cout<<sum2(&a,&b)<<endl;//передача аргументів по
посиланню, результат=4
cout<<sum3(a,b)<<endl;//передача аргументів по
посиланню, результат=4
cout<<a<<","<<b<<endl;//результат=3,3
}
int sum1(int a,int b) //визначення функції sum1
{return a+b;}//функція повертає значення a+b цілого
типу
```

2.24 Глобальні, локальні і статичні змінні

```
#include <iostream.h>
float x;//глобальна змінна
float sum(float a,float b)//функція sum
{
//a, b - локальні змінні
static int k=0;//статична змінна
float x;//локальна змінна для sum
x=a+b;
cout<<++k<<endl;
return x;
}
void main()
{
float x;//локальна змінна для main
```

```
x=5;
::x=6;//доступ до глобальної змінної
cout<<sum(x,::x)<<endl;
cout<<sum(1,3)<<endl;
}
```

2.25 Масив як параметр функції

```
#include <iostream.h>
int sum1(int A[5])//оголошення функції sum1
{
//тут можлива зміна масиву V
int s=0;
for(int i=0;i<5;i++) s=s+A[i];
return s;//повернути значення суми елементів масиву s
}
int sum2(int *A)//оголошення функції sum2
{
//тут можлива зміна масиву V
int s=0;
for(int i=0;i<5;i++) s=s+A[i];
return s;//повернути значення суми елементів масиву s
}
void main()
{
int V[5]={1,2,3,4,5};//оголошення локального масиву
cout<<sum1(V)<<endl;//передача аргументів по
посиланню
cout<<sum2(V)<<endl;//передача аргументів по
посиланню
}
```

2.26 Рядки і функції обробки рядків

```
#include <iostream.h>
#include <string.h>//заголовочний файл з функціями
обробки рядків
void main()
{
//оголошення і ініціалізація рядків (масиву символів)
char s[20]="cpp";
char s2[20]="";
```

```
char s3[]={'h','e','l','l','o','\0'};
*s='C';//присвоїти першому символу рядка 'C'
*(s+1)='+';//другому символу
*(s+2)='+';//третьому символу
*(s+3)='!';//четвертому символу
//вивести довжину рядків
cout<<s<<" "<<strlen(s)<<endl;
cout<<s3<<" "<<strlen(s3)<<endl;
strcpy(s2,"plus");//копіювати в рядок
cout<<s2<<endl;//вивести рядок
strncpy(s2,"C plus plus",6);//копіювати в рядок 6
символів
cout<<s2<<endl;
char *p=strstr(s,"++");//знайти підрядок в рядку
cout<<p<<endl;
strcat(s,s3);//об'єднати рядки
cout<<s<<endl;
float x=strtod("5.37",&p);//конвертувати рядок в
дійсне число
gcvt(x,3,s);//конвертувати дійсне число в рядок
cout<<s<<endl;
}
```

2.27 Структури

```
#include <iostream.h>
struct A//структура A (тип даних)
{
char Name[10];//поле типу масив символів
int Year;//поле цілого типу
bool Sex;//поле логічного типу
};
//функція для виводу вмісту об'єкта
void prn(A *ps)
{
cout<<ps->Name<<endl;
cout<<ps->Year<<endl;
cout<<ps->Sex<<endl;
}
void main()
{
//оголошення і ініціалізація об'єкта s1
A s1={"Ivan",1980,true};
```

```
A s2;//оголошення об'єкта s2
//присвоєння полям значень
strcpy(s2.Name,"Vasyl");
s2.Year=1981;
s2.Sex=true;
//динамічне створення структури
//ps - вказівник на структуру
A *ps=new A;
//присвоєння полям значень
strcpy(ps->Name,"Inna");
ps->Year=1983;
ps->Sex=false;
//вивід вмісту об'єктів
prn(&s1);
prn(&s2);
prn(ps);
delete ps;//вивільнення ps
}
```

2.28 Обробка виключних ситуацій

```
//слова try, throw, catch
#include <iostream.h>
#include <stdio.h>
//клас виключної ситуації
class Ex
{
public:
float x;//поле
char s[20];//поле
Ex(float a,char *str)//конструктор
{x=a; strcpy(s,str);}
};
void main()
{
float a=1,b,c;
do//початок циклу
{
cin>>b;//ввести b
//блок, в якому можливе виникнення виключної ситуації
try {
if (b<0) throw b;//генерувати виключну ситуацію float
```

```
if (b==0) throw "b=0";//генерувати виключну ситуацію
char*
if (b>1000) throw Ex(b," b>1000");//генерувати
виключну ситуацію Ex
if (b==1) throw true;//генерувати виключну ситуацію
bool
c=a/b;
cout<<c<<endl;//вивести c
}
//перехоплення виключної ситуації float
catch (float x){cout<<"b="<<x<<endl;}
//перехоплення виключної ситуації char*
catch (const char *s){cout<<s<<endl;}
//перехоплення виключної ситуації Ex
catch (Ex obj) {cout<<obj.x<<obj.s<<endl;}
//перехоплення інших виключних ситуацій
catch (...){cout<<"warning! exception!\n";}
}while(getchar()!='q');//поки не натиснуто q
}
```

3 Об’єктно-орієнтоване програмування мовою C++

3.1 Простий клас

```
#include <iostream.h>
//клас A
class A
{
//відкриті члени класу
public:
int a;//поле цілого типу a
void print()//метод print
{cout<<a<<endl;}
};
void main()
{
A obj;//оголошення об’єкту obj типу A
obj.a=2;//присвоїти полю a об’єкта obj 2
obj.print();//виклик методу print об’єкта obj
}
```

3.2 Структура

```
#include <iostream.h>
//структура A
struct A
{
//відкриті члени
int a;//поле цілого типу a
void print()//метод print
{cout<<a<<endl;}
};
void main()
{
A obj;//оголошення об’єкту obj типу A
obj.a=2;//присвоїти полю a об’єкта obj 2
obj.print();//виклик методу print об’єкта obj
}
```

3.3 Об’єднання

```
//об'єднання
#include <iostream.h>
union A//об'єднання A
{int x;//поле x цілого типу
float y;//поле y дійсного типу
};
int main()
{
A u;//створити екземпляр u об'єднання A
u.x=1;//присвоїти x 1
u.y=2;//присвоїти y 2, x не доступне
cout<<u.x<<endl;//помилка!
cout<<u.y<<endl;//об’єднання містить тільки дане y
return 0;
}
```

3.4 Конструктор

```
#include <iostream.h>
class A //клас A
{
int a,b;//закриті поля класу
```

```
public://відкриті члени класу
//конструктор (метод) зі списком ініціалізації
закритих полів
A(int inita,int initb):a(inita),b(initb){};
void print()//метод
{cout<<a<<" "<<b<<endl;}
};
void main()
{
A obj(2,3);//створення об'єкту і виклик конструктора
obj.print();//виклик методу
}
```

3.5 Деструктор

```
#include <iostream.h>
class A
{
char *a;
int len;
public:
A(int ln)//конструктор A
{
len=ln;
//динамічно виділити пам'ять для масиву символів
кількістю len
//вказівник a вказує на перший елемент масиву
a=new char[len];
}
~A()//деструктор ~A
{delete a;}//вивільнити пам'ять для a
void input(char *s)//метод для вводу рядка
{a=s;}
void print()//метод для виводу рядка
{cout<<a<<endl;}
};
void main()
{
A obj(4);//створити об'єкт і викликати конструктор
obj.input("hello");//викликати метод
obj.print();//викликати метод
//викликати деструктор
}
```

3.6 Конструктор копіювання

```
#include <iostream.h>
class A//клас A
{
int a;
public:
//перевантажені конструктори
A(int inita);//конструктор
A(A&);//конструктор копіювання
void print()
{cout<<a<<endl;}
};
A::A(int inita)//визначення конструктора
{a=inita;}
A::A(A &obj)//визначення конструктора копіювання
{a=obj.a;}
void main()
{
A obj1(3);//створення об'єкту obj1 і виклик
конструктора
A obj2=obj1;//створення об'єкту obj2 і виклик
конструктора копіювання
obj2.print();
}
```

3.7 Динамічне створення об'єктів типу класу

```
#include <iostream.h>
//клас A
class A
{
//закриті члени
int a;
int b;
//відкриті члени
public:
A(int inita, int initb);//конструктор (метод)
int sum()//метод
{
int c=a+b;
return c;
}
```

```
};
//визначення конструктора A класу A
A::A(int inita, int initb)
{
//присвоїти закритим полям значення параметрів
a=inita;
b=initb;
}
void main()
{
//статично створити об'єкт obj1, викликати
конструктор
A obj1(2,3);
cout<<obj1.sum()<<endl;//виклик методу sum
//динамічно виділити пам'ять для об'єкту типу A,
викликати конструктор
//вказівник pobj вказує на цю пам'ять
A *pobj =new A(4,5);
cout<<pobj->sum()<<endl;//виклик методу sum
delete pobj;//вивільнити пам'ять для pobj
cout<<pobj->sum()<<endl;//помилка!
}
```

3.8 Вкладені класи (1)

```
#include <stdio.h>
class A//клас A
{
class B//вкладений у клас A клас B
{
int b;
public:
B(int initb):b(initb){};
void print();//метод класу B
};
public://відкриті члени класу A
B objb;//поле objb типу B
A(B initobjb):objb(initobjb){};
void print()//метод класу A
{printf("%d\n",objb);}
};
//визначення методу вкладеного класу B
void A::B::print()
```

```
{printf("%d\n",b);}
void main()
{
A obj(5);//створення об'єкту obj і ініціалізація поля
b
obj.print();//виклик методу print() класу A
obj.objb.print();//виклик методу print() вкладеного
класу B
}
```

3.9 Вкладені класи (2)

```
#include <stdio.h>
class B//клас B
{
int b;
public:
B(int initb):b(initb){};
};
class A//клас A
{
B objb;//поле objb типу B
public:
A(B initobjb):objb(initobjb){};
void print()
{printf("%d\n",objb);}
};
void main()
{
A obj(5);//створення об'єкту obj і ініціалізація поля
b
obj.print();//виклик методу print() класу A
}
```

3.10 Вкладені класи (3)

```
#include <stdio.h>
class B//клас B
{
int b;//поле b типу int
public:
B(int initb):b(initb){};
```

```
void print(){printf("%d\n",b);}//метод класу B
};
class A//клас A
{
public:
B objb;//поле objb типу B
A(B initobjb):objb(initobjb){};
void print(){printf("%d\n",objb);}//метод класу A
};
void main()
{
B obj1(5);//створення об’єкту obj1 і ініціалізація
поля b
A obj(obj1);//створення об’єкту obj і ініціалізація
поля objb
obj.print();//виклик методу print() класу A
obj.objb.print();//виклик методу print() класу B
}
```

3.11 Вкладені класи (4)

```
#include <iostream.h>
class B//клас B
{
int b,c;//поля b і c типу int
public:
//метод для ініціалізації закритих полів b і c
void set(int sb,int sc){b=sb; c=sc;}
void print(){cout<<b<<" "<<c<<endl;}
};
class A//клас A
{
public:
B objb;//поле objb типу B
A(int ib, int ic);
};
A::A(int ib, int ic)//визначення конструктора класу A
{objb.set(ib,ic);}
void main()
{
A obj(5,7);//створення об’єкту obj і ініціалізація
полів b,c поля objb
obj.objb.print();
```

```
}
```

3.12 Вкладені класи (5)

```
#include <iostream.h>
class B//клас B
{
int b,c;//поля b і c типу int
public:
B(int initb,int
initc):b(initb),c(initc){};//конструктор класу B
void print(){cout<<b<<" "<<c<<endl;}
};
class A//клас A
{
public:
B objb;//поле objb типу B
A(int ib, int ic):objb(ib,ic){};//конструктор класу A
};
void main()
{
A obj(5,7);//створення об’єкту obj і ініціалізація
полів b,c поля objb
obj.objb.print();
}
```

3.13 Дружні функції (1)

```
#include <iostream.h>
class A//клас A
{
int a,b;//закриті поля
public:
A(int ia, int ib):a(ia),b(ib){};
friend int sum(int pa,int pb);//оголошення дружньої
функції (не члену класу A)
};
int sum(int pa, int pb)//визначення дружньої функції
{
A* obj=new A(pa,pb);
int s=obj->a+obj->b;//можливий доступ до закритих
полів класу A
```

```
delete obj;
return s;
}
void main()
{
cout<<sum(2,3);//виклик дружньої функції
}
```

3.14 Дружні функції (2)

```
#include <iostream.h>
class A//клас A
{
int a,b;//закриті поля
public:
A(int ia, int ib):a(ia),b(ib){};
friend int sum(A);//оголошення дружньої функції (не
члену класу A)
};
int sum(A pobj)//визначення дружньої функції
{
return pobj.a+pobj.b;//можливий доступ до закритих
полів класу A
}
void main()
{
A obj(2,3);//створити об'єкт obj
cout<<sum(obj);//викликати дружню функцію з
аргументом obj
}
```

3.15 Успадкування

```
#include <iostream.h>
class A//базовий клас A
{
protected://захищені члени
int a;//поле
public://відкриті члени
A(int ia):a(ia){};//конструктор
};
//клас B, він успадковує клас A
```

```
class B:public A
{
//закриті члени
int b;//поле
public://відкриті члени
B(int ia, int ib):A(ia),b(ib){};//конструктор
int sum(){return a+b;}//метод
};
void main()
{
B obj(2,3);//створити об'єкт obj класу B,
ініціалізувати a=2,b=3
cout<<obj.sum();
}
```

3.16 Статичний поліморфізм, перевантаження методів

```
#include <iostream.h>
class A//клас A
{
int c;
double cd;
public:
//перевантажені методи sum
int sum(int a, int b);
double sum(double ad, double bd);
void print()
{cout<<c<<" "<<cd<<endl;}
};
//визначення перевантаженого методу sum,
//який отримує і повертає дані цілого типу
int A::sum(int a, int b)
{return c=a+b;}
//визначення перевантаженого методу sum,
//який отримує і повертає дані дійсного типу
double A::sum(double ad, double bd)
{return cd=ad+bd;}
void main()
{
A obj;
cout<<obj.sum(2,3)<<endl;//виклик методу sum(int a,
int b)
```

```
cout<<obj.sum(2.1,3.05)<<endl;//виклик методу
sum(double ad, double bd)
obj.print();
}
```

3.17 Перевантаження операторів

```
#include <iostream.h>
class A//клас A
{
int a;
public:
A(int inita):a(inita){};//конструктор
A(A &obj){a=obj.a;}//конструктор копіювання
//перевантажена операторна функція-член operator+,
//яка отримує посилання і повертає об'єкт класу A
A operator+(const A& obj)
{
A obj_sum(0);//створити об'єкт класу A
//присвоїти результату суму поля a та поля a об'єкта
obj (аргументу)
obj_sum.a=a+obj.a;
return obj_sum;//повернути результат obj_sum
}
void print(){cout<<a;}
};
void main()
{
A obj1(2),obj2(3),obj3(0);//створити об'єкти класу A
obj3=obj1+obj2;//застосування перевантаженого
оператора +
//або obj3=obj1.operator+(obj2)
obj3.print();
}
```

3.18 Динамічний поліморфізм, віртуальні функції (1)

```
#include <iostream.h>
class A//базовий клас A
{
protected:
int a;
```

```
public:
A(int ia):a(ia){};
//віртуальна функція
virtual void print() //спробуйте без слова virtual
{cout<<a<<endl;}
};
class B: public A//клас B, він успадковує клас A
{
protected:
int b;
public:
B(int ia, int ib):A(ia),b(ib){};
void print()//віртуальна функція
{cout<<a<<" "<<b<<endl;}
};
class C: public B//клас C, він успадковує клас B
{
protected:
int c;
public:
C(int ia,int ib, int ic):B(ia,ib),c(ic){};
void print()//віртуальна функція
{cout<<a<<" "<<b<<" "<<c<<endl;}
};
void main()
{
A obj1(1); obj1.print();
B obj2(2,3); obj2.print();
C obj3(4,5,6); obj3.print(); //віртуальність не
потрібна
A* pobj;
pobj=&obj1; pobj->print();
pobj=&obj2; pobj->print();
pobj=&obj3; pobj->print(); //необхідна віртуальність
A* pobj1=new A(1); pobj1->print();
A* pobj2=new B(2,3); pobj2->print();
A* pobj3=new C(4,5,6); pobj3->print(); //необхідна
віртуальність
}
```

3.19 Динамічний поліморфізм, віртуальні функції (2)

```
#include <iostream.h>
```

```
class A//базовий клас A
{
protected:
int a;
public:
A(int ia):a(ia){};
//віртуальна функція
virtual int sum() //спробуйте без слова virtual
{return a;}
void print()//метод
{cout<<sum()<<endl;}
};
class B: public A//клас B, він успадковує клас A
{
protected:
int b;
public:
B(int ia, int ib):A(ia),b(ib){};
int sum()//віртуальна функція
{return a+b;}
};
void main()
{
B obj(2,3);//створити об'єкт класу B
//при виклику методу print буде викликаний метод sum
класу B,
//а не A, оскільки sum віртуальний
obj.print();
}
```

3.20 Успадкування множинне

```
#include <iostream.h>
class A//базовий клас A
{
protected://захищені члени
int a;
public://відкриті члени
A(int ia):a(ia){};//конструктор
};
class B//базовий клас B
{
protected://захищені члени
```

```
int b;
public://відкриті члени
B(int ib):b(ib){};//конструктор
};
//клас C, він успадковує клас A і B
class C:public A, public B
{
//закриті члени
int c;
public://відкриті члени
C(int ia, int ib, int
ic):A(ia),B(ib),c(ic){};//конструктор
int sum(){return a+b+c;}
};
void main()
{
//створити об'єкт класу C; a=2, b=3, c=4
C obj(2,3,4);
cout<<obj.sum();
}
```

3.21 Діамантове успадкування

```
#include <iostream.h>
class A//базовий клас A
{
protected:
int a;
public:
A(int ia):a(ia){};
};
//клас B, він успадковує клас A
class B: public A
{
protected:
int b;
public:
B(int ia, int ib):A(ia),b(ib){};
};
//клас C, він успадковує клас A
class C: public A
{
protected:
```

```
int c;
public:
C(int ia, int ic):A(ia),c(ic){};
};
//клас D, він успадковує клас B і C
class D:public B, public C
{
int d;
public:
D(int ia1, int ia2, int ib, int ic, int
id):B(ia1,ib),C(ia2,ic),d(id){};
void print()
{cout<<B::a<<C::a<<b<<c<<d;}
};
void main()
{
//створити об’єкт класу D; B::a=1, C::a=2, b=3, c=4,
d=5
D obj(1,2,3,4,5);
obj.print();
}
```

3.22 Діамантове віртуальне успадкування

```
#include <iostream.h>
class A//базовий клас A
{
protected:
int a;
public:
A():a(0){}; //конструктор по замовчуванню
A(int ia):a(ia){};
};
//клас B, він віртуально успадковує клас A
class B: virtual public A
{
protected:
int b;
public:
B(int ia, int ib):A(ia),b(ib){};
};
//клас C, він віртуально успадковує клас A
class C: virtual public A
```

```
{
protected:
int c;
public:
C(int ia, int ic):A(ia),c(ic){};
};
//клас D, він успадковує клас B і C
class D:public B, public C
{
int d;
public:
D(int ia, int ib, int ic, int
id):B(ia,ib),C(ia,ic),d(id){};
void print()
{cout<<a<<b<<c<<d;}
};
void main()
{
//створити об'єкт класу D; a=2, b=3, c=4, d=5
D obj(2,3,4,5);
obj.print();
}
```

3.23 Динамічне приведення і ідентифікація типів (dynamic_cast і typeid)

```
#include <iostream.h>
#include <typeinfo.h>//заголовочний файл для
ідентифікації типів
class A//поліморфний клас A
{virtual void f(void){}};//віртуальна функція
class B//пустий клас B
{ };
class C : public A, public B //множинне успадкування
A і B класом C
{ };
void main()
{
C objc, *pobjc;//оголосити об'єкт і вказівник класу C
A *pobja = &objc;//оголосити вказівник класу A, який
вказує на об'єкт objc
if ((pobjc = dynamic_cast<C *>(pobja)) != 0)//якщо
приведення pobja до типу C* успішне
```

```
{cout<<typeid(pobjc).name()<<endl;} //то інформація
про тип pobjc
B *pobjb;//оголосити вказівник класу B
if ((pobjb = dynamic_cast<B *>(pobja)) != 0)//якщо
приведення pobja до типу B* успішне
{cout << typeid(pobjb).name() << endl;} //то
інформація про тип pobjb
}
```

3.24 Шаблони функцій (1)

```
#include <iostream.h>
//шаблон функції sum
template <typename T>//список шаблонних параметрів
T sum(T a, T b)
{return a+b;};
void main()
{
int x1=2, x2=3;
cout<<sum(x1,x2)<<endl;//спеціалізація функції sum (T
є int)
double y1=2.1, y2=3.5;
cout<<sum(y1,y2)<<endl;//спеціалізація функції sum (T
є double)
cout<<sum<int>(4,6);//явна спеціалізація функції sum
(T є int)
}
```

3.25 Шаблони функцій (2)

```
#include <iostream.h>
//шаблон функції sum
template <typename T>
T sum(T a, T b)
{cout<<"I-"; return a+b;};
//явна спеціалізація шаблонної функції sum
template <>
int sum<int>(int a, int b)
{cout<<"II-"; return a+b;};
//перевантаження шаблонної функції sum
template <typename T>
T sum(T a)
```

```
{cout<<"III-"; return a+a;};
bool sum(bool a, bool b)//перевантаження функції sum
{cout<<"IV-"; return a+b;};
void main()
{
double y1=2.1, y2=3.5;
cout<<sum(y1,y2)<<endl;//результат: I-5.6
int x1=2, x2=3;
cout<<sum(x1,x2)<<endl;//результат: II-5
cout<<sum(2)<<endl;//результат: III-4
bool b1=true, b2=false;
cout<<sum(b1,b2)<<endl;//результат: IV-1
}
```

3.26 Шаблони класів

```
#include <iostream.h>
//шаблон класу A
template <typename T=int>//список шаблонних
параметрів
class A
{
T a;//поле a типу T
public:
A(T ia):a(ia){};
void print();
};
//шаблон методу print шаблонного класу A
template <typename T>
void A<T>::print()
{cout<<a<<endl;}
void main()
{
A<> obj(2);//створити об'єкт класу A<int>
obj.print();
A<double> obj2(2.1);//створити об'єкт класу A<double>
obj2.print();
}
```

3.27 Шаблони класів і віртуальні методи

```
#include <iostream.h>
```

```
//шаблон класу A
template <class T>
class A
{
protected:
T a;//поле a типу T
public:
A(T ia):a(ia){};//конструктор
//віртуальний метод sum для динамічного поліморфізму
virtual int sum() //спробуйте без слова virtual
{return a;}
void print()
{cout<<sum()<<endl;}
};
//шаблон класу B, успадкований від шаблону класу A
template <class T>
class B: public A<T>
{
protected:
T b;//поле b типу T
public:
B(T ia, T ib):A<T>(ia),b(ib){};//конструктор
int sum()//віртуальний метод
{return a+b;}
};
void main()
{
B<int> obj(2,3);//створення об'єкту класу B<int>
//у методі print буде викликаний метод sum класу B,
//а не A, оскільки sum - віртуальний
obj.print();
}
```

3.28 Застосування флагів форматування при потоковому вводу-виводі

```
#include <iostream.h>//бібліотека потокового вводу-
виводу
void main()
{
//установити флагі: шістнадцятковий, показувати
основу системи числення
cout.flags(ios::hex|ios::showbase);
cout<<16<<endl;
```

```
//зняти флаг: шістнадцятковий
cout.unsetf(ios::showbase);
cout<<16<<endl;
//зняти установлені флагі
cout.unsetf(cout.flags());
cout<<16<<endl;
//установити флагі: науковий, показувати основу
системи числення
cout.setf(ios::scientific|ios::showpos);
cout.precision(5);//установити кількість цифр після
крапки
cout.width(20);//установити ширину поля виводу
cout.fill('*');//установити символ-заповнювач
cout<<135.59017605<<endl;
}
```

3.29 Установка флагів за допомогою функцій-маніпуляторів

```
#include <iostream.h>//бібліотека потокового вводу-
виводу
#include <iomanip.h>//бібліотека маніпуляторів
void main()
{
cout<<hex<<showbase<<16<<endl;//шістнадцятковий,
показувати основу
cout.unsetf(cout.flags());//зняти установлені флагі
cout<<showpoint<<16.0<<endl;//виводити десяткову
крапку
cout<<noshowpoint<<16.0<<endl;//не виводити десяткову
крапку
cout.unsetf(cout.flags());//зняти установлені флагі
cout<<setprecision(2)<<1.0541<<endl;//виводити 2
знаки після крапки
}
```

3.30 Створення функції-маніпулятора

```
#include <iostream.h>
#include <iomanip.h>
//функція-маніпулятор my_manip,
//яка отримує і повертає посилання на потоковий
об’єкт класу ostream
```

```
ostream& my_manip(ostream& s)
{
s.width(10);//установити ширину поля
s.precision(4);//установити точність
s.setf(ios::showpos);//установити флаг: показувати
основу
return s;//повернути s
}
void main()
{
cout<<my_manip<<16.0124<<endl;//виклик маніпулятора
my_manip
}
```

3.31 Перевантаження операторів потокового вводу-виводу

```
#include <iostream.h>
class A//клас A
{
double a;
public:
A(double ia):a(ia){};
//оголошення дружньої операторної функції operator<<
friend ostream& operator<<(ostream &os, A obj);
};
//перевантаження операторної функції operator<<,
//яка отримує посилання на потоковий об'єкт класу
ostream, об'єкт класу A
//і повертає посилання на потоковий об'єкт класу
ostream
ostream& operator<<(ostream &os, A obj)
{
os.precision(2);//установити точність
os.setf(ios::showpos);//установити флаг форматування
os<<obj.a;//помістити a в потоковий об'єкт os
return os;//повернути os
}
void main()
{
A obj(2.125);//створити об'єкт класу A
//помістити obj в потік виводу, використовуючи
перевантажений оператор <<
cout<<obj;
```

```
}
```

3.32 Клас рядкових потоків

```
#include <iostream.h>
#include <strstream.h>//бібліотека класу рядкових
потоків
#include <string.h>
void main()
{
const char s[]="hello";
strstream out;//створити рядковий потоковий об'єкт
out класу strstream
out<<s;//помістити в потік out рядок s
cout<<out.rdbuf()<<endl;//вивести вміст рядкового
буферу
}
```

3.33 Бінарні файли

```
#include <iostream.h>
#include <fstream.h>//бібліотека класів файлового
потокового вводу-виводу
void main()
{
//створити файловий об'єкт бінарного вводу f1,
зв'язати з файлом 1.txt
ifstream f1("1.txt", ios_base::binary);
//створити файловий об'єкт бінарного виводу f2,
зв'язати з файлом 2.txt
ofstream f2("2.txt", ios_base::binary);
char c;
while (!f1.eof())//поки не кінець файлу f1
повторювати цикл
{
f1.get(c);//читати символ з f1
if (f1) f2.put(c);//якщо f1, записати символ у f2
}
f1.close();//закрити файл f1
f2.close();//закрити файл f2
}
```

3.34 Бінарні файли прямого доступу

```
#include <fstream.h>//бібліотека класів файлового
потокового вводу-виводу
#include <conio.h>
#include <iostream.h>
//структура для запису у бінарний файл
struct My
{
char C[20];
int N;
};
//перевантажений оператор << для виводу вмісту
структури
ostream& operator<< (ostream &s, const My &x)
{
return s << x.C << ";" << x.N;
}
void main()
{
My s1={"string 1",1};//перший екземпляр
My s2={"string 2",2};//другий екземпляр
My s3={"string 3",3};//третій екземпляр
My s4={"string 4",4};//четвертий екземпляр
//створити файловий об'єкт для потокового виводу
ofstream ofile("file.dat", ios::binary);
//записати у файл екземпляри у вигляді рядків
розміром My
ofile.write((char*)&s1, sizeof(My));
ofile.write((char*)&s2, sizeof(My));
ofile.write((char*)&s3, sizeof(My));
ofile.write((char*)&s4, sizeof(My));
ofile.close();//закрити файл ofile
//створити файловий об'єкт для потокового вводу
ifstream ifile("file.dat", ios::binary);
//читати з файлу екземпляри у вигляді рядків розміром
My поки не кінець файлу
while (ifile.read((char*)&s1, sizeof(My)),
!ifile.eof())
{
//якщо s1.C="string 2" вивести вміст екземпляра
if (!strcmp(s1.C,"string 2")) cout<<s1<<endl;//виклик
перевантаженого оператора <<
```

```
}
ifile.close();//закрити файл ifile
//створити файловий об’єкт для потокового вводу
ifstream ifile2("file.dat", ios::binary);
//установити позицію у файлі на третій запис від
початку
ifile2.seekg(sizeof(My)*2,ios::beg);
ifile2.read((char*)&s1, sizeof(My));//читати дані
//вивести вміст екземпляра
cout<<s1<<endl;//виклик перевантаженого оператора <<
ifile2.close();//закрити файл ifile2
getch();//очікувати вводу з клавіатури
}
```

3.35 Шаблон структури "пара" бібліотеки STL

```
#include <stdio.h>
//шаблон структури pair
template <class T1, class T2>//список шаблонних
параметрів
struct pair
{
T1 first;//поле типу T1
T2 second;//поле типу T2
//конструктори
pair(){};
pair(const T1& x,const T2& y):first(x), second(y){};
};
//шаблон перевантаженої операторної функції
operator==
template <class T1, class T2>//список шаблонних
параметрів
inline bool operator==(const pair<T1,T2>& x,const
pair<T1,T2>& y)
{
//повернути true, якщо поля рівні
return x.first==y.first && x.second==y.second;
};
void main()
{
//спеціалізації структур (T1 є double, T2 є int)
pair<double,int> obj1(2.1,3);//створити об’єкт obj1
pair<double,int> obj2(2.1,3);//створити об’єкт obj2
```

```
//використання перевантаженого оператора ==
if (obj1==obj2) printf("obj1=obj2");
}
```

3.36 Використання бібліотеки STL

```
#include <iostream.h>
#include <vector>//бібліотека контейнера вектора
#include <algorithm>//бібліотека функцій-алгоритмів
#include <functional>//бібліотека функціональних
об'єктів (функторів)
//функціональний шаблонний клас add
template <typename T>
class add
{
private:
T x;
public:
add(T ix) : x(ix) { }//конструктор
void operator()(T i)//перевантажений оператор ()
{cout<<i+x<<" ";}
};
//функціональний шаблонний клас add2,
//успадкований від функтора binary_function
template <typename T>
class add2 : binary_function<T, T, T>
{public:
T operator()(T x, T y)//перевантажений оператор ()
{return x+y;}
};
//функція для виводу вмісту вектора
void print(vector<int> &v)
{
static int k=0;
cout<<++k<<": ";
//оголошено ітератор (вказівник на елементи вектора)
vector<int>::iterator it;
//в циклі ітератор указує з першого по останній
елемент вектора
//вивести значення елементу
for(it=v.begin(); it!=v.end(); it++) cout<<*it<<" ";
cout<<endl;
}
void main()
```

```
{
vector<int> v(5);//оголосити контейнер-вектор
розміром 5
vector<int>::iterator it;//оголосити ітератор
//приклад заповнення вектора
int i;
for(it=v.begin(),i=0; it!=v.end(); it++,i++) v[i]=1;
print(v);//1-вивід вектора
//приклад заповнення вектора
for(it=v.begin(); it!=v.end(); it++) *it=2;
print(v);//2
v.erase(v.begin(),v.end());//видалення усіх елементів
print(v);//3-вивід пустого вектора
//приклад заповнення вектора
for(int i=0; i<5; i++) v.push_back(i);//добавляє в
вектор i
print(v);//4
v.erase(v.begin()+4);//видалення 5-го елемента
print(v);//5
v.erase(v.begin()+1,v.begin()+3);//видалення 2 і 3-го
елементів
print(v);//6
cout<<v.max_size()<<endl;//максимальний розмір
вектора
cout<<v.size()<<endl;//розмір вектора
v.insert(v.begin()+1,3,7);//вставка 7 в діапазон з
індексами [1,3]
print(v);//7
v.pop_back();//видалення останнього елемента
print(v);//8
//створити вектор v2 - копію v
vector<int> v2(v.begin(), v.end());
cout<<v2.empty()<<endl;//вивід логічного 0 (вектор не
пустий)
print(v2);//9
//дописуємо в v вміст v2 з першого по передостанній
елемент
v.assign(v2.begin(),v2.end()-1);
print(v);//10
//порівнюємо вектори
if (v==v2) cout<<"v=v2";else cout<<"v<>v2"<<endl;
v2=v;//присвоюємо v2 вміст v
//створення вектора з масиву
```

```
int array [] = { 1, 3, 5, 7 , 9};
vector<int> v3 (array, array + 5);
print(v3);//11
//приклади застосування алгоритмів:
//кількість елементів менших 100
cout<<count_if(v.begin(),v.end(),bind2nd(less<int>(),
100))<<endl;
add<int> f(2); //функтор (функціональний об'єкт)
//для кожного елементу з діапазону застосувати f
for_each(v.begin(),v.end(),f);
cout<<endl;
//пошук першого числа < 3
//застосування адаптера bind2nd і предиката less
cout<<*find_if(v.begin(),v.end(),bind2nd(less<int>(),
3));
cout<<endl;
//копіювати діапазон v3 в v
copy(v3.begin(),v3.end(),v.begin());
print(v);//12
//замінити в діапазоні числа 5 на 13
replace(v3.begin(),v3.end(),5,13);
print(v3);//13
//перемножити елементи v3[i]=v[i]*v2[i]
//використовується стандартний функтор multiplies
transform(v.begin(),v.end(),v2.begin(),v3.begin(),mul
tiplies<int>());
print(v3);//14
//додати елементи v3[i]=v[i]+v2[i]
//використовується функтор користувача add2
transform(v.begin(),v.end(),v2.begin(),v3.begin(),add
2<int>());
print(v3);//15
//сортувати v3 по зростанню
sort(v3.begin(),v3.end());
print(v3);//16
//сортувати v3 по спаданню
sort(v3.begin(),v3.end(),greater<int>());
print(v3);//17
//заповнити v3 числами 7
fill(v3.begin(),v3.end(),7);
print(v3);//18
v2.clear();//очистити вектор v2
}
```

4 Приклади використання компонентів VCL

4.1 Поради для вивчення компонентів:

1 Переглядайте вміст Object Inspector.
2 Користуйтесь довідкою F1.
3 Користуйтесь підказкою коду Ctrl+Space.
4 Користуйтесь довідковою літературою і прикладами.
5 Перед копіюванням прикладів в код програми розмістіть на формі усі необхідні компоненти (наприклад, `Label1, Label2, Button1` і т.д.) та створіть усі необхідні функції-обробники подій (наприклад, `TForm1::Button1Click, TForm1::FormCreate` і т.д.), які використовуються в прикладі.

4.2 Простий проект типу Win32 Application у C++ Builder

Постановка задачі: створити додаток Win32 у C++ Builder, вікно якого містить кнопку і поле вводу. При натиску на кнопці в полі повинен з'явитись текст "Hello World!"

Послідовність виконання:

1 Створіть проект, якщо він не створений (*File*|*New*| *Application*)
2 Добавте на `Form1` компоненти Edit і Button з панелі компонентів
3 Двічі клацніть на кнопці `Button1` і у функцію `TForm1::Button1Click` вставте код:

```
Edit1->Text="Hello World!";
```

4 В конструктор форми `TForm1::TForm1` вставте код:

```
Caption="First program";
Button1->Caption="Click Me!";
```

5 Відкомпілюйте проект (*Run*|*Run*)
6 Збережіть проект (*File*|*Save Project As...*).
7 Закрийте проект (*File*|*Close All*)

Код програми

```
//файл Unit1.h з описом класу форми
#ifndef Unit1H//директива умовної компіляції "якщо не визначено"
#define Unit1H//директива визначення макросу
//директиви підстановки файлів
#include <Classes.hpp>//бібліотека службових класів
```

```
#include <Controls.hpp>//бібліотека класів
компонентів
#include <StdCtrls.hpp>//бібліотека класів
стандартних компонентів
#include <Forms.hpp>//бібліотека класів форм
//клас форми TForm1, який успадкований від TForm
class TForm1 : public TForm
{
__published://компоненти, які обслуговуються IDE
        TButton *Button1;//компонент VCL кнопка класу
TButton
        TEdit *Edit1;//компонент VCL поле вводу класу
TEdit
        //метод обробки події OnClick
        void __fastcall Button1Click(TObject
*Sender);
private:        //закриті члени класу
public:         //відкриті члени класу
        __fastcall TForm1(TComponent*
Owner);//конструктор
};
//специфікатор вказує, що Form1 визначена в окремому
файлі
extern PACKAGE TForm1 *Form1;
#endif//кінець директиви умовної компіляції


//файл Unit1.cpp з визначенням методів форми
#include <vcl.h>//бібліотека VCL
//директива закінчує список заголовочних файлів,
//придатних для попередньої компіляції
#pragma hdrstop
#include "Unit1.h"//під'єднати файл Unit1.h
//вказує, що упаковані модулі ініціалізовані
//в порядку, визначеному їх залежностями
#pragma package(smart_init)
//помічає файли *.dfm як модулі форми
#pragma resource "*.dfm"
TForm1 *Form1;//створити об'єкт форми класу TForm1
//визначення конструктора форми
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
```

```
//присвоїти властивості форми Form1 рядок
//властивість Caption відповідає надпису на
компоненті
Caption="First program";
//присвоїти властивості кнопки Button1 рядок
Button1->Caption="Click Me!";
}
//визначення методу обробника події OnClick
void __fastcall TForm1::Button1Click(TObject *Sender)
{
//присвоїти властивості поля вводу Edit1 рядок
//властивість Text відповідає тексту на компоненті
Edit1->Text="Hello World!";
}


//файл Project1.cpp з головною функцією WinMain
#include <vcl.h>
#pragma hdrstop
//використовувати файл форми Unit1.cpp
USEFORM("Unit1.cpp", Form1);
//головна функція
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
try//блок обробки виключних ситуацій
{
//Application - об'єкт, який описує додаток
Application->Initialize();//ініціалізувати
//створити форму Form1
Application->CreateForm(__classid(TForm1), &Form1);
Application->Run();//виконати
}
//перехоплення виключної ситуації Exception
catch (Exception &exception)
{Application->ShowException(&exception);}//показати
помилку
//перехоплення інших виключних ситуацій
catch (...)
{
  try
  {throw Exception("");}//генерувати виключну
ситуацію
  //перехоплення виключної ситуації Exception
```

```
  catch (Exception &exception)
  {Application->ShowException(&exception);}//показати
помилку
}
return 0;
}
```

4.3 Класи

AnsiString (String) – тип рядків

```
String s; //або тип AnsiString - створити рядок s
s="Hello"; //оператор присвоєння
String s2(s); //створити рядок s2 і викликати
конструктор з параметром s
s=s+" User"; //оператор конкатенації
s+="!"; //оператор присвоєння (додати до рядка "!")
bool cmp=s>s2; //оператор порівняння
s[6]='_'; //оператор []
char *c; //вказівник на масив символів
c=s.c_str(); //присвоїти с вказівник на масив
символів рядка s
Application->MessageBoxA(c,"",0); //вивести
повідомлення
s=AnsiString(c); //присвоїти s рядок з масиву
символів с
int i=s.Pos("User"); //позиція входження рядка "User"
в s
s.Delete(7,4); //видалити 4 символи починаючи з
позиції 7
s.Insert("User",7); //вставити рядок "User" в позицію
7
int l=s.Length(); //довжина рядка
s2=s.SubString(7,4); //підрядок з позиції 7 довжиною
4 символи
bool d=s.IsDelimiter("_,.",6); //true, якщо символ 6
належить до розділювачів з рядка "_,."
s=s.LowerCase(); //перетворює рядок до нижнього
регістру
double x=12.356; //змінна дійсного типу
int y=12; //змінна цілого типу
s=FloatToStr(x); //перетворити дійсне число в рядок
x=s.ToDouble(); //перетворити рядок в дійсне число
s=IntToStr(y); //перетворити ціле число в рядок
```

```
y=s.ToInt(); //перетворити рядок в ціле число
TVarRec args[4] = {" x=",x,"; y=",y}; //масив
аргументів
s=Format("%s%8.3f%s%d", args, 3); //присвоїти
форматований рядок
s.printf("x=%f",x); //установити значення
форматованого рядка
s=s.Trim(); //видалити пробіли на початку і в кінці
Edit1->Text=s; //присвоїти властивості Text поля
Edit1
```

TApplication - додаток
(див. також TApplicationEvents)

1.Змініть модуль Project1.cpp так:

```
USEFORM("Unit1.cpp", Form1);
//точка входу в додаток Windows
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
try //перехоплювати виключення
{
Application->Initialize(); //ініціалізувати
Application->CreateForm(__classid(TForm1), &Form1);
//створити форму
Application->Title=Application->ExeName; //змінити
підпис на ім'я exe-файлу
Application->MainForm->Caption="1"; //надпис головної
форми
Application->Icon->LoadFromFile("C:/Program
Files/Borland/CBuilder6/Examples/Icons/internet.ico")
; //іконка
Application->ShowMainForm=false; //не показувати
головну форму
Application->Run(); //виконати додаток
}
catch (Exception &exception) //при перехопленні
виключення
{Application->ShowException(&exception);} //показати
виключення
return 0;
}
```

2.Змініть модуль Unit1.cpp так:

```
//конструктор форми
```

```
__fastcall TForm1::TForm1(TComponent* Owner):
TForm(Owner)
//зареєструвати гарячу клавішу Ctrl+Alt+A
{RegisterHotKey(Handle,25,MOD_ALT|MOD_CONTROL,int('A'
));}
//при натиску кнопки Button1
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Application->Minimize(); //звернути додаток
Sleep(2000); //зупинити виконання на 2 секунди
Application->Restore(); //відновити додаток
Form1->Refresh(); //обновити форму
Application->MessageBoxA("Terminate!","!",0);
//вивести повідомлення
Application->Terminate(); //знищити додаток
}
//при перехопленні повідомлення WM_HOTKEY
//цей метод створюється за допомогою інструмента Add
Method
//з вказівкою Message Handler - WM_HOTKEY
void __fastcall TForm1::WMHotKey(TWMHotKey& key)
{//якщо id гарячої клавіші=25
if (key.HotKey=25)
{
Form1->Show(); //показати форму
Application->Restore(); //відновити додаток
Application->BringToFront(); //помістити на передній
план
}
}
//при знищенні форми
void __fastcall TForm1::FormDestroy(TObject *Sender)
{UnregisterHotKey(Handle,25);} //відмінити реєстрацію
гарячої клавіші
//при мінімізації (звертанні) додатку
void __fastcall
TForm1::ApplicationEvents1Minimize(TObject *Sender)
{
Form1->Hide(); //сховати форму
static int i=0;
//відрахувати 10 секунд зі звуковим сигналом
for (i=1;i<=10;i++)
{
```

```
Beep();
Sleep(1000);
}
}
```

TScreen – стан екрану

```
Height=Screen->Height;//висота форми=висоті екрану у
пікселях
Memo1->Lines->Add(Height);
Width=Screen->Width;//ширина форми=ширині екрану у
пікселях
Memo1->Lines->Add(Width);
//або
Screen->Forms[0]->Height=Screen->Height;
//або
Screen->ActiveForm->Height=Screen->Height;
Memo1->Lines->Add(Screen->Fonts->Strings[0]);//перший
екранний шрифт
Screen->Cursor=crCross;//змінює курсор
Memo1->Lines->Add(Screen->ActiveControl-
>Name);//активний елемент керування
Memo1->Lines->Add(Screen->DesktopHeight);//висота
робочого столу
Memo1->Lines->Add(Screen->WorkAreaHeight);//висота
робочої зони
```

Set – шаблон класу множина

```
Set <int, 1, 10> S,S2;//множина S і S2 цілих від 1 до
10
S<<1<<2;//добавити в множину S 1,2
S2<<1<<2<<3<<4;//добавити в множину S2 1,2,3,4
S=S+S2;//множина S містить 1,2,3,4
S>>4;//видалити з множини 4
Caption="";
for (int i=1;i<=10;i++)
//якщо множина S мітить і то вивести його
if (S.Contains(i)) Caption=Caption+i;
S.Clear();//очистити множину
```

TList – список вказівників, TObject – базовий клас VCL

```
TList *List = new TList;//список
TClass C=Button1->ClassType();//тип об'єкту Button1
//в циклі від першого по останній компонент форми
```

```
for (int i=0;i<ComponentCount;i++)
{
List->Add((TObject *)Components[i]);//добавити у
список компонент
TObject *obj=(TObject *)List->Items[i];//створити
об'єкт obj з елементу списку
Memo1->Lines->Add(obj->ClassName());//вивести ім'я
класу
//або так:
//TClass C2=obj->ClassType();
//Memo1->Lines->Add(C2->ClassName());
//якщо клас об'єкту obj успадкований від базового
класу класу TButton
if (obj->InheritsFrom(C->ClassParent()))
//вивести ім'я базового класу об'єкту obj
Memo1->Lines->Add(obj->ClassParent()->ClassName());
}
```

TComponent – базовий клас компонентів, TControl – базовий клас візуальних компонентів

```
//створити компонент Btn, власник - форма Form1
TButton *Btn=new TButton(Form1);
Btn->Parent=Panel1;//батько - Panel1
//видалити компонент зі списку володіння (Components)
Form1
Form1->RemoveComponent(Btn);
Panel1->InsertComponent(Btn);//добавити у список
володіння Panel1
//в циклі від першого по останній компонент форми
for (int i=0;i<ComponentCount;i++)
{
Memo1->Lines->Add(Form1->Components[i]-
>ComponentIndex);//індекс компоненту
Memo1->Lines->Add(Form1->Components[i]->Name);//ім'я
компоненту
//визначення стилю компонента csInheritable
if (Form1->Components[i]-
>ComponentStyle.Contains(csInheritable))
Memo1->Lines->Add("csInheritable");
Memo1->Lines->Add(Form1->Components[i]->Owner-
>Name);//ім'я власника
//якщо компонент має батька
if (Form1->Components[i]->HasParent())
```

```
Memo1->Lines->Add("HasParent");//вивести "HasParent"
TControl *C;//вказівник на елемент керування
//якщо можливе приведення компоненту до типу TControl
//(якщо компонент є елементом керування)
if (C=dynamic_cast<TControl *>(Components[i]))
{
C->Top=0;//змінити розташування
Memo1->Lines->Add(C->Parent->Name); //вивести ім'я
батька
C->Hide();//зробити невидимим
//якщо невидимий то відобразити
if (C->Visible==false) C->Show();
C->Enabled=false;//зробити недоступним
}
}
```

TWinControl – базовий клас віконних компонентів, TGraphicControl – базовий клас графічних компонентів

```
for (int i=0;i<ComponentCount;i++)
{
TWinControl *C;//вказівник на віконний елемент
керування
//якщо можливе приведення компоненту до типу
TWinControl
//(якщо компонент є віконним елементом керування)
if (C=dynamic_cast<TWinControl *>(Components[i]))
{
Memo1->Lines->Add(C->Name+"(WinControl)");//вивести
ім'я
bool cf=C->CanFocus();//чи може елемент прийняти
фокус вводу
bool f=C->Focused();//чи елемент у фокусі вводу
C->SetFocus();//установити фокус вводу
C->TabStop=true;//дозволяє клавішу табуляції для
перебору елементів
C->ScaleBy(1,2);//зменшити у 2 рази
void *p=C->Handle;//віконний дескриптор елементу
}
TGraphicControl *C2;//вказівник на графічний елемент
керування
//якщо можливе приведення компоненту до типу
TGraphicControl
//(якщо компонент є графічним елементом керування)
```

```
if (C2=dynamic_cast<TGraphicControl
*>(Components[i]))
Memo1->Lines->Add(C2-
>Name+"(GraphicControl)");//вивести ім'я
}
```

TStrings – базовий клас списку рядків, TStringList – список рядків

```
TStringList *S=new TStringList;//створити список
рядків
S->Add("String3");//добавити в список рядок
S->Insert(1,"String2");//вставити рядок в позицію 1
S->Append("String1");//добавити рядок в кінець
S->Sort();//сортувати список
S->AddStrings(S);//добавити копію себе
int i;
S->Find("String2",i);//знайти індекс рядка
S->Delete(i);//видалити рядок з індексом i
S->Delete(S->IndexOf("String3"));//видалити рядок
S->Move(1,2);//змінює позицію рядка 1 на 2
Memo1->Lines=S;//помістити список в Memo1
Memo1->Lines->Assign(S);//або так
Memo1->Lines->Add(S->Strings[3]);//добавити рядок з
індексом 3
Memo1->Lines->Add(S->Text);//добавити весь текст
списку
S->SaveToFile("file.txt");//зберегти у файл
delete S;//знищити S
```

TStream – базовий клас потокових об'єктів, TMemoryStream – потік в динамічній пам'яті

```
TMemoryStream* pms = new TMemoryStream();//потік
динамічної пямяті
TStringList *S=new TStringList;//список рядків
char Buffer[10];//буфер
memset(Buffer, 0, 10);//установити буфер
S->DelimitedText="1,2,3,4,5";//заповнити список
рядків
S->SaveToStream(pms);//записати список в потік
pms->Position = 0;//установити позицію на початок
pms->Read(Buffer, 4);//прочитати 4 байти з потоку в
буфер
pms->Position = 0;
pms->Write(Buffer, strlen(Buffer) + 1);//записати
буфер в потік
```

```
pms->Position = 0;
Memo1->Lines->LoadFromStream(pms);//завантажити рядки
з потоку
Memo1->Lines->Add(AnsiString(Buffer));//додати вміст
буферу
delete pms,S;
```

TThread – потік керування

Постановка задачі: використати компонент класу `TThread` для створення багатопотокового додатку.

Послідовність виконання:

1 Створіть проект (*File*|*New*| *Application*)

2 Добавте на форму три компонента Button

3 Створіть новий клас потоку (*File*|*New*|*Other...*|*Thread Object*)

4 Назвіть клас `TThread1`

5 Добавте в файл Unit2.cpp директиви

```
#include "Unit1.h"
#include <stdlib.h>
```

6 Добавте в файл Unit1.cpp директиву

```
#include "Unit2.h"
```

7 Виберіть в Class Explorer клас `TThread1` і в контекстному меню виберіть *New Method...*

8 Вкажіть ім'я методу: `Rand`, тип результату: `void`, доступ: `Private`, директиви: `__fastcall`

9 Запрограмуйте метод `Rand()` :

```
void __fastcall TThread1::Rand()
{
 //Змінити випадковому пікселю на канві колір на
чорний
 Form1->Canvas->Pixels[rand()%1000][rand()%1000]
=clBlack;
 Sleep(1);//Зупинити потік на 1мс
 }
```

10 Запрограмуйте метод `Execute ()` :

```
void __fastcall TThread1::Execute()
{
 do
 Synchronize(Rand);//виконати метод Rand
синхронізовано
 while(!Terminated);//поки потік не знищений
}
```

11 Вставте в Unit1.cpp код:

```
TThread1 *Thread1;//оголосити вказівник на
потоковий об'єкт
```
12 Запрограмуйте методи `TForm1::Button1Click`, `TForm1::Button2Click`, `TForm1::Button3Click`:
```
void __fastcall TForm1::Button1Click(TObject
*Sender)
{
Thread1=new TThread1(false);//динамічно створити і
виконати потік
Thread1->Priority=tpIdle;//встановити пріоритет на
найнижчий
Thread1->FreeOnTerminate=true;//знищити об’єкт
при завершенні
}
//натиснута кнопка 2
void __fastcall TForm1::Button2Click(TObject
*Sender)
{
Thread1->Terminate();//завершити роботу потоку
}
//натиснута кнопка 3
void __fastcall TForm1::Button3Click(TObject
*Sender)
{
if (!Thread1->Suspended)//якщо потік не
призупинено
{Thread1->Suspend();}//зупинити виконання
else
{Thread1->Resume();}//продовжити виконання
}
```
13 Запустіть програму: *Run*.

TForm - форма

Створення модальних форм

Постановка задачі: створити додаток, в якому використовуються модальні форми, створити і використати модальні форми, які розміщені у dll.

Послідовність виконання:

1 Створюємо додаток: *File/New/Application*

2 Добавте кнопку на форму `Form1`.

3 Добавляємо нову форму: *File/New/Form*

4 Добавте кнопку на форму `Form2`.

5 У файл Unit1.cpp дописуємо код:

```
#include "Unit2.cpp"
```
6 У конструктор форми `TForm1::TForm1` вставте:
```
Form2=new TForm2(Application);
```
7 У реалізацію методу `TForm1::Button1Click` вставте:
```
Form2->ShowModal();
```
8 У реалізацію методу `TForm2::Button1Click` вставте:
```
Form2->Close();
```
9 Зберігаємо усе: *File/Save All*
10 Компілюємо: *Run*.

Створення модальних форм розміщених у DLL

Послідовність виконання:
1 *File/New/Other/DLL Wizard*
2 *Source Type: C++, Use VCL, Ok*
3 Добавляємо нову форму*: File/New/Form*
4 Добавляємо новий заголовочний файл *File/New/Other/Header File* і зберігаємо його під іменем Unit1.h, дописуємо у нього код:
```
extern "C" void
__declspec(dllexport)ShowMyForm(void);
```
5 У файл Unit1.cpp дописуємо код:
```
#include "Unit1.h"
#include "Unit2.h"
```
6 Добавляємо у файл Unit1.cpp функцію:
```
void ShowMyForm(void)
{
    Form2 = new TForm2(NULL);
    Form2->ShowModal();
    delete Form2;
}
```
7 Зберігаємо усе: *File/Save All.*
8 Компілюємо: *Project/Build Project1*.

Створюємо додаток, який динамічно використовує DLL:
9 Створюємо додаток: *File/New/Application*
10 Добавте кнопку на форму.
11 У розділ private класу форми `TForm1` вставте код:
```
HINSTANCE Dll;
```
12 У конструктор форми `TForm1::TForm1` вставте код:
```
Dll = NULL;
```
13 У реалізацію методу `TForm1::Button1Click` вставте код:
```
typedef void __declspec(dllimport)SHOWMYFORM(void);
SHOWMYFORM *ShowMyForm;
Dll = LoadLibrary("Project1.dll");
```

```
ShowMyForm = (SHOWMYFORM *)GetProcAddress(Dll,
"_ShowMyForm");
ShowMyForm();
FreeLibrary(Dll);
```
14 Зберігаємо: *File/Project As...* Називаємо модуль Unit3, а проект Project2.
15 Компілюємо: *Run*.

Створюємо додаток, який статично використовує DLL:
9 Створюємо додаток: *File/New/Application*
10 Добавте кнопку на форму.
11 Добавте до проекту бібліотеку Project1.lib
12 У файл модуля дописуємо код:
```
#include "Unit1.h"
```
13 У реалізацію методу `TForm1::Button1Click` вставте код:
```
  ShowMyForm();
```
14 Зберігаємо: *File/Project As...* Називаємо модуль Unit4, а проект Project3.
15 Компілюємо: *Run*.

Створення MDI форм

Постановка задачі: створити додаток з багатодокументним інтерфейсом (MDI), створити і використати MDI форми, які розміщені у dll.
Послідовність виконання:
1 Створюємо додаток: *File/New/Application*
2 Добавте кнопку на форму `Form1`.
3 Добавляємо нову форму*: File/New/Form*
4 Добавте кнопку на форму `Form2`.
5 У файл Unit1.cpp дописуємо код:
```
#include "Unit2.h"
```
6 У файл Unit2.cpp дописуємо код:
```
#include "Unit1.h"
```
7 У реалізацію методу `TForm1::Button1Click` вставте:
```
Form2=new TForm2(this);
Form2->Show();
```
8 У реалізацію методу `TForm2::Button1Click` вставте:
```
Button1->Caption="2";
Form1->Button1->Caption="1";
```
9 Зберігаємо усе: *File/Save All*
10 Компілюємо: *Run*.

Створення MDI форм розміщених у DLL

Послідовність виконання:
1 *File/New/Other/DLL Wizard*

2 *Source Type: C++, Use VCL, Ok*
3 Добавляємо нову форму*: File/New/Form*
4 Добавляємо новий заголовочний файл *File/New/Other*/*Header File* і зберігаємо його під іменем Unit1.h, дописуємо у нього код:

```
extern "C" void
__declspec(dllexport)ShowMyForm(TComponent*);
```

5 У файл Unit1.cpp дописуємо код:

```
#include "Unit1.h"
#include "Unit2.h"
```

6 Добавляємо у файл Unit1.cpp функцію:

```
void ShowMyForm(TComponent* Owner)
{
    Form2 = new TForm2(Owner);
    Form2->Show();
}
```

7 Зберігаємо усе: *File/Save All.*
8 Компілюємо: *Project/Build Project1*.

Створюємо додаток, який динамічно використовує DLL:
9 Створюємо додаток: *File/New/Application*
10 Добавте кнопку на форму.
11 У розділ private класу форми `TForm1` вставте код:

```
HINSTANCE Dll;
```

12 У конструктор форми `TForm1::TForm1` вставте код:

```
Dll = NULL;
```

13 У реалізацію методу `TForm1::Button1Click` вставте код:

```
typedef void
__declspec(dllimport)SHOWMYFORM(TComponent*);
SHOWMYFORM *ShowMyForm;
Dll = LoadLibrary("Project1.dll");
ShowMyForm = (SHOWMYFORM *)GetProcAddress(Dll,
"_ShowMyForm");
ShowMyForm(this);
```

14 Зберігаємо: *File/Project As...* Називаємо модуль Unit3, а проект Project2.
15 Компілюємо: *Run*.

4.4 Компоненти Standard

TFrame – кадр, контейнер компонентів

Постановка задачі: створити і використати компонент класу `TFrame`
Послідовність виконання:

1 Закрийте проект (*File*|*Close All*)

2 Добавте фрейм (*File*|*New*|*Frame*)
3 Добавте на *Frame1* компонент Button
4 Збережіть проект (*File*|*Save Project As..*) Назвіть модуль Unit_frame.cpp а проект Project_frame
5 Закрийте проект (*File*|*Close All*)
6 Створіть проект, який використовуватиме фрейм (*File*|*New*|*Application*)
7 Добавте до проекту (*Project*|*Add to Project..*) файл Unit_frame.cpp
8 Перетягніть з панелі компонентів компонент Frames на форму, виберіть `Frame1`.
9 Двічі клацніть на кнопці `Button1` і у функцію `TForm1::Frame11Button1Click` вставте код:

```
Application->MessageBox("Button1 натиснута", "Ok",
MB_OK);
```

TMainMenu – головне меню, TPopupMenu – контекстне меню, TActionList – список дій

Постановка задачі: використати компоненти `TMainMenu`, `TPopupMenu`, `TActionList` для побудови графічного інтерфейсу додатку.

Послідовність виконання:

1 Помістіть на форму компоненти: `MainMenu1`, `PopupMenu1`, `Button1`, `Button2`, `ActionList1`.
2 На компоненті `MainMenu1` в випадаючому меню виберіть *Menu Disigner*.
3 Добавте в пункти меню і пункти підменю. Надпис на кожному пункті задається його властивістю `Caption`. Пункти підменю створюються шляхом вибору *Create Submenu* в випадаючому меню.
4 Аналогічно добавте в `PopupMenu1` пункти і пункти за допомогою *Menu Disigner...*
5 На компоненті `ActionList1` в випадаючому меню виберіть *Action List Editor*.
6 Добавте дії `Action1` і `Action2` за допомогою кнопки *New Action*.
7 Створіть функції обробки подій `OnExecute` для `Action1` і `Action2` і в кожну з них вставте відповідно:

```
Application->MessageBox("дія Action1", " Action1",
MB_OK);
Application->MessageBox("дія Action2", " Action2",
MB_OK);
```

8 Змініть значення властивості Action для Button1, одного пункту MainMenu1 і PopupMenu1 на Action1.
9 Змініть значення властивості Action для Button2, одного пункту MainMenu1 і PopupMenu1 на Action2.
10 Запустіть програму: *Run*.

TLabel – мітка, надпис

```
Label1->AutoSize=false; //немає автоматичної зміни
розмірів
Label1->Alignment=taRightJustify; //вирівнювання
надпису
Label1->Font->Color=clRed; //колір шрифту
Label1->Font->Size=14; //розмір шрифту
Label1->Caption="Hello"; //текст надпису
Label1->Caption=5.1; //текст надпису
Label1->Caption="Float "+FloatToStr(2.5); //текст
надпису
```

TButton - кнопка

```
Button1->Caption="OK"; //текст надпису
Button1->Hint="Button1"; //текст підказки
Button1->ShowHint=true; //показувати підказку
Button1->Top=100; //Y-координата верхнього лівого
кута
Button1->Left=100; //X-координата верхнього лівого
кута
Button1->Height=20; //висота
Button1->Width=30; //ширина
```

Створення єдиної функції-обробника події для всіх кнопок TButton

```
//конструктор форми
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
//вказати функцію-обробник події OnClick
 Button1->OnClick=Button1Click;
 Button2->OnClick=Button1Click;
 Button3->OnClick=Button1Click;
}
//функція-обробник події OnClick
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
static int i=1; //статична змінна
//створити вказівник на об'єкт, який викликав подію
OnClick
TButton *Button=(TButton*)Sender;
Button->Caption=i++; //змінити надпис на кількість
викликів
}
```

TRadioButton - перемикач

```
RadioButton1->Checked=true; //вибраний стан
RadioButton1->Caption="check1"; //текст надпису
RadioButton1->Enabled=false; //не доступний
```

TCheckBox - прапорець

```
if (CheckBox1->Checked==true) //якщо стан вибраний
{CheckBox1->Caption="true";} //змінити надпис на
"true"
else //інакше
{CheckBox1->Caption="false";} //змінити надпис на
"false"
CheckBox1->Checked=true; //вибраний стан
CheckBox1->State=cbGrayed; // "третій" стан
```

Присвоєння властивості Checked=true всім компонентам TCheckBox

```
//для всіх компонентів на формі
for(int i=0; i<ComponentCount; i++)
{
//створити вказівник типу TCheckBox* на даний
компонент шляхом
//спроби динамічного перетворення типу
TCheckBox
*CBox=dynamic_cast<TCheckBox*>(Components[i]);
if (CBox) CBox->Checked=true; //встановити вибраний
стан
}
```

TEdit – поле редагування

```
Edit1->Text="Hello"; //текст у полі
Edit1->Text=2.1; //текст у полі
double x=StrToFloat(Edit1->Text); //перетворити текст
у дійсне число
Edit1->Text=x;
char s[128]; //масив символів
```

```
strcpy(s,Edit1->Text.c_str()); //помістити текст у
масив символів
Edit1->Text=s;
Edit1->PasswordChar='*'; //символ для вводу пароля
Edit1->Clear(); //очистити поле
```

TMemo – багаторядкове поле редагування

```
Memo1->Lines->Add("1"); //додати рядок "1"
Memo1->Lines->Strings[0]="2"; //змінити перший рядок
на "2"
Memo1->Lines->Assign(ListBox1->Items); //установити
рядки з ListBox1
Memo1->Lines->SaveToFile("Data.txt"); //зберегти
рядки у файл
Memo1->Lines->LoadFromFile("Data.txt"); //читати
рядки з файлу
//вивести весь текст Memo1 в діалоговому вікні
MessageBoxA
Application->MessageBoxA(Memo1-
>Text.c_str(),"",mbNone);
```

TListBox – список рядків

```
//добавити елементи "1", "2", "3"
ListBox1->Items->DelimitedText="1,2,3";
ListBox1->Items->Clear(); //очистити
ListBox1->Items->Add("1"); // добавити елемент "1",
ListBox1->Items->Add("2"); // добавити елемент "2",
char c[][10]={"1000","2000","3000"}; //масив символів
//додати елементи "1000", "2000", "3000"
for(int i=0;i<3;i++)ListBox1->Items->Add(c[i]);
ListBox1->Items->Insert(1,"3"); //помістити елемент
"3" на друге місце
ListBox1->Items->Delete(1); //видалити другий елемент
int k= ListBox1->Items->Count; //кількість елементів
int i=ListBox1->ItemIndex; //індекс вибраного
елементу
String s=ListBox1->Items->Strings[i]; //текст i-го
елементу
```

Заповнення ListBox об’єктами

```
//добавити елемент з рядком Button1->Name і об’єктом
Button1
ListBox1->Items->AddObject(Button1->Name,Button1);
```

```
//добавити елемент з рядком Button2->Name і об'єктом
Button2
ListBox1->Items->AddObject(Button2->Name,Button2);
int i=ListBox1->ItemIndex; //індекс вибраного
елементу
//створити вказівник типу TButton* на об'єкт з
індексом i
TButton *b=(TButton*)(ListBox1->Items->Objects[i]);
b->Caption="Selected Button"; //змінити текст надпису
```

TComboBox – випадаючий список рядків

```
ComboBox1->Text="5"; //текст
//добавити елемент з рядком ComboBox1->Text
ComboBox1->Items->Add(ComboBox1->Text);
```

TScrollBar – смуга прокручування

```
ScrollBar1->Kind=sbVertical;//тип смуги прокручування
- вертикальна
ScrollBar1->Min=1;//мінімальне значення
ScrollBar1->Max=100;//максимальне значення
ScrollBar1->LargeChange=10;//великий крок
ScrollBar1->SmallChange=1;//малий крок
int i=ScrollBar1->Position;//позиція
Edit1->Text=i;
```

TPanel - панель

Динамічне створення і видалення компонентів

```
//функція-обробник події OnClick для Button1
void __fastcall TForm1::Button1Click(TObject *Sender)
{
TButton *Btn=new TButton(Panel1); //динамічно
створити кнопку
Btn->Parent=Panel1; //власник кнопки - Panel1
//координати кнопки
Btn->Left=10;
Btn->Top=10;
Btn->Caption="Btn"; //текст надпису
Btn->OnClick=BtnClick; //функція-обробник події
OnClick
}
//функція-обробник події OnClick об'єкту Btn
void __fastcall TForm1::BtnClick(TObject *Sender)
{
//показати діалогове вікно з надписом "Remove"
```

```
 Application->MessageBoxA("Remove","",mbNone);
TControl *Btn=Panel1->Controls[0]; // Btn – перший
об’єкт на Panel1
 Panel1->RemoveControl(Btn); //видалити об’єкт Btn
}
```

4.5 Компоненти Additional

TBitBtn – кнопка з піктограмою

```
BitBtn1->Kind=bkOK; //тип кнопки
BitBtn1->Glyph->LoadFromFile("1.bmp"); //рисунок на
кнопці
BitBtn1->Caption="Hello"; //текст надпису
```

Приклад використання технології Drag&Drop

```
//натиснута кнопка миші на BitBtn1
void __fastcall TForm1::BitBtn1MouseDown(TObject
*Sender,
      TMouseButton Button, TShiftState Shift, int X,
int Y)
{
//якщо натиснута права кнопка, розпочати
перетягування
if(Button==mbRight)BitBtn1->BeginDrag(false,5);
BitBtn1->BringToFront(); //винести на передній план
}
//відбувається перетягування над Panel1
void __fastcall TForm1::Panel1DragOver(TObject
*Sender, TObject *Source,
      int X, int Y, TDragState State, bool &Accept)
{
//вивести координати у надпис форми
 Form1->Caption=IntToStr(X)+","+IntToStr(Y);
}
//завершено перетягування
void __fastcall TForm1::Panel1DragDrop(TObject
*Sender, TObject *Source,
      int X, int Y)
{
 BitBtn1->Parent=Panel1; //власник BitBtn1 - Panel1
 BitBtn1->Top=Y; //координати кнопки
 BitBtn1->Left=X;
}
```

```
//натиснута кнопка BitBtn1
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
//вивести ім'я власника BitBtn1
Application->MessageBox(BitBtn1->Parent-
>Name.c_str(),"Parent",mbNone);
}
```

TSpeedButton – кнопка з піктограмою і фіксацією

```
SpeedButton1->Glyph->LoadFromFile("1.bmp"); //рисунок
на кнопці
SpeedButton1->Caption="Ok"; //текст надпису на кнопці
```

TMaskEdit – поле редагування з шаблонами

```
MaskEdit1->Clear(); //очистити
MaskEdit1->Text="150797"; //текст до застосування
маски
MaskEdit1->EditMask="!99/99/00;1"; //маска
String S=MaskEdit1->EditText;//текст після
застосування маски
```

TStringGrid - таблиця

```
StringGrid1->ColCount=3; //кількість колонок
 StringGrid1->RowCount=3; //кількість рядків
 StringGrid1->Cells[0][1]="row1"; //текст в комірці
Cells[0][1]
 StringGrid1->Cells[1][0]="col1"; //текст в комірці
Cells[1][0]
 StringGrid1->Cells[1][1]=2; //текст в комірці
Cells[1][1]
//дозвіл і заборона редагування вмісту комірок
 StringGrid1->Options<<goEditing; //додаємо в множину
goEditing
 StringGrid1->Options>>goEditing; //видаляємо з
множини goEditing
 int c=StringGrid1->Col; //індекс колонки вибраної
комірки
 int r=StringGrid1->Row; //індекс рядка вибраної
комірки
Edit1->Text=StringGrid1->Cells[c][r]; //помістити в
поле вміст Cells[c][r]
```

TDrawGrid – таблиця зображень

```
//при необхідності перерисування зображення в
комірках DrawGrid1
void __fastcall TForm1::DrawGrid1DrawCell(TObject
*Sender, int ACol,
      int ARow, TRect &Rect, TGridDrawState State)
{
//створити графічний об'єкт bmp
Graphics::TBitmap *bmp = new Graphics::TBitmap();
bmp->LoadFromFile("1.bmp"); //завантажити з файлу
TRect b; //створити об'єкт-прямокутник
b.Left=0; //координати прямокутника
b.Top=0;
b.Right=bmp->Width;
b.Bottom=bmp->Height;
//перерисування зображення в комірці 1,1
if ((ACol==1)&&(ARow==1))
{DrawGrid1->Canvas->CopyRect(Rect,bmp->Canvas,b);}
//перерисування зображення в комірках використовуючи
//список зображень ImageList1
int i=ARow*DrawGrid1->ColCount+ACol;
if (i<=ImageList1->Count-1)
{ImageList1->Draw(DrawGrid1-
>Canvas,Rect.Left,Rect.Top,i,true);}
}
```

TImage – контейнер графічних зображень, TPicture – графічне зображення

```
Image1->Picture->LoadFromFile("1.bmp"); //завантажує
картинку з файлу
Image1->AutoSize=true; //автоматична зміна розміру
Image1
Image1->Stretch=false; //автоматична зміна розміру
картинки відключена
Image1->Center=true; // розташовує картинку по центру
Image1
int h=Image1->Height; //висота
int w=Image1->Width; //ширина
Image1->Canvas->LineTo(w,h); //рисує на канві лінію
до точки (w,h)
Image1->Picture->SaveToFile("2.bmp"); //зберегти
картинку у файл
```

TShape – геометрична форма, TBrush - пензель, TPen - перо

```
Shape1->Shape=stCircle; //форма - коло
 Shape1->Brush->Color=clGreen; //колір пензля
 Shape1->Brush->Style=bsSolid; //стиль пензля
//створити графічний об'єкт BrushBmp
 Graphics::TBitmap *BrushBmp = new Graphics::TBitmap;
 BrushBmp->LoadFromFile("1.bmp"); //завантажити з
файлу
 Shape1->Brush->Bitmap=BrushBmp; //установити рисунок
пензля
 Shape1->Pen->Style=psDot; //стиль пера
 Shape1->Pen->Color=clBlue; //колір пера
 Shape1->Pen->Mode=pmCopy; //режим рисування ліній
```

TScrollBox – контейнер прокручування

```
ScrollBox1->AutoScroll=true; //автоматично
відображати лінійки прокручування
//прокручувати вміст відразу
 ScrollBox1->HorzScrollBar->Tracking=true;
 ScrollBox1->VertScrollBar->Tracking=true;
```

TCheckListBox – список рядків з індикаторами

```
CheckListBox1->Items->Add("1"); //добавити елемент
"1"
CheckListBox1->Items->Add("2"); //добавити елемент
"2"
//i змінюється від 0 до кількості елементів
for (int i=0;i<CheckListBox1->Items->Count;i++)
{
if (CheckListBox1->Checked[i]), //якщо елемент [i]
вибраний
{CheckListBox1->Items->Strings[i]="true";} //змінити
надпис на "true"
else //інакше
{CheckListBox1->Items->Strings[i]="false";} //змінити
надпис на " false "
}
```

TSplitter - розділювач

```
ListBox1->Align=alLeft; //вирівнює розташування
ListBox1
ListBox2->Align=alClient; //вирівнює розташування
ListBox2
```

```
Splitter1->Left=ListBox2->Left; //координата
Splitter1 по горизонталі
Splitter1->Align=alLeft; //вирівнює розташування
Splitter1
```

TControlBar – контейнер інструментальних панелей

```
BitBtn1->DragMode=dmAutomatic; //автоматичний режим
перетягування
BitBtn1->DragKind=dkDock; //вид перетягування
ControlBar1->AutoDrag=true; //дозволяє перетягування
за межі компоненту
ControlBar1->AutoDock=true; //автоматичне
вбудовування при перетягуванні над компонентом
ControlBar1->RowSize=50; //висота рядка
```

TApplicationEvents – перехоплювач подій додатку

```
int i,j=0;
//при простоюванні
void __fastcall
TForm1::ApplicationEvents1Idle(TObject *Sender, bool
&Done)
{Application->Title=IntToStr(i++)+" "+IntToStr(j);}
//при активізації
void __fastcall
TForm1::ApplicationEvents1Activate(TObject *Sender)
{Application->MessageBox("Activate","",0);}
//при мінімізації
void __fastcall
TForm1::ApplicationEvents1Minimize(TObject *Sender)
{Application->MessageBox("Minimize","",0);}
//при отриманні повідомлення Windows
void __fastcall
TForm1::ApplicationEvents1Message(tagMSG &Msg,
      bool &Handled)
{j++;}
```

TValueListEditor – компонент редагування списків, які містять пару ім’я/значення

```
int r;
//при натиску на кнопку Button1
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
 ValueListEditor1->InsertRow("I","1",true);
//вставити рядок
 ValueListEditor1->InsertRow("II","2",true);
//вставити рядок
 ValueListEditor1->InsertRow("III","3",true);
//вставити рядок
//стиль редагування першого значення - простий
 ValueListEditor1->ItemProps[0]->EditStyle=esSimple;
//стиль редагування другого значення - список
 ValueListEditor1->ItemProps[1]-
>EditStyle=esPickList;
//додати елементи в список
 ValueListEditor1->ItemProps[1]->PickList->Add("1");
 ValueListEditor1->ItemProps[1]->PickList->Add("2");
//стиль редагування третього значення – через
діалогове вікно
 ValueListEditor1->ItemProps[2]-
>EditStyle=esEllipsis;
//вивести текст третього ключа
 Edit1->Text=ValueListEditor1->Keys[2];
//вивести текст значення з ключем "II"
 Edit2->Text=ValueListEditor1->Values["II"];
 }
//натиснута кнопка для редагування третього значення
void __fastcall
TForm1::ValueListEditor1EditButtonClick(TObject
*Sender)
{
//якщо вибрано третій рядок змінити текст значення з
ключем "III" за допомогою діалогового вікна
if(r==3)ValueListEditor1-
>Values["III"]=InputBox("","","");
}
//вибрана комірка
void __fastcall
TForm1::ValueListEditor1SelectCell(TObject *Sender,
      int ACol, int ARow, bool &CanSelect)
{
r=ARow; //присвоїти номер вибраного рядка
}
```

TLabeledEdit – поле редагування з надписом

```
LabeledEdit1->EditLabel->Caption="Hello"; //текст
надпису
```

```
LabeledEdit1->LabelPosition=lpLeft; //розміщення
надпису
LabeledEdit1->Text="User"; //текст
```

TColorBox – вікно вибору кольору, TColor - колір

```
ColorBox1->Style<<cbCustomColor; //стиль вікна вибору
кольору
Form1->Color=ColorBox1->Selected; //колір форми
змінити на вибраний колір
```

TChart - діаграма

Добавляємо на форму компонент Chart і викликаємо редактор діаграми Edit Chart... В редакторі діаграм добавляємо дві серії типу Line. В конструктор форми вставляємо код:

```
Chart1->Title->Text->Clear(); //очистити надпис назви
діаграми
Chart1->Title->Text->Add("My Chart"); //змінити текст
надпису назви
Chart1->BottomAxis->Title->Caption="X"; //текст
надпису осі X
Chart1->LeftAxis->Title->Caption="Y"; //текст надпису
осі Y
Series1->Clear(); //очистити серію 1
Series1->Title="x^2"; //надпис серії 1
Series2->Clear(); //очистити серію 2
Series2->Title="x^3"; //надпис серії 2
//добавити дані (X, Y) в серії 1 і 2
for (float x=0; x<5;x+=0.1)
{
 Series1->AddXY(x,x*x,"",clRed);
 Series2->AddXY(x,x*x*x,"",clBlue);
}
//змінити всі значення X серії 1 на 1
for (int i=0; i<Series1->Count();i++)
{
 Series1->XValues->Value[i]=1;
}
```

4.6 Компоненти Win32

TTabControl - вкладки

```
TabControl1->Tabs->Add("1"); //добавити вкладку "1"
```

```
TabControl1->Tabs->Add("2"); //добавити вкладку "2"
TabControl1->Tabs->Add("3"); //добавити вкладку "3"
//якщо вибрана третя вкладка, змінити її надпис на
"*"
if (TabControl1->TabIndex==2)TabControl1->Tabs-
>Strings[2]="*";
```

TPageControl - сторінки

```
PageControl1->TabPosition=tpLeft; //розміщення
закладок зліва
TTabSheet *Tab=new TTabSheet(this);//динамічне
створення сторінки
Tab->PageControl=PageControl1; //вказує PageControl
для сторінки
Tab->Caption="New"; //текст надпису
PageControl1->ActivePage=Tab; //зробити сторінку Tab
активною
int i=PageControl1->ActivePageIndex; //індекс
активної сторінки
PageControl1->Pages[i]->Caption="Sel"; //надпис і-ї
сторінки
```

TImageList – список зображень, TBitmap – зображення, бітова матриця

```
//динамічно створює графічний об'єкт
Graphics::TBitmap *Bmp = new Graphics::TBitmap;
Bmp->LoadFromFile("1.bmp"); //завантажити зображення
з файлу
ImageList1->Add(Bmp,NULL); //додати графічний об'єкт
в список
Bmp->LoadFromFile("2.bmp"); //завантажити зображення
з файлу
ImageList1->Insert(0,Bmp,NULL); //вставити об'єкт на
перше місце
ImageList1->Move(1,0); //перемістити зображення з
позиції 1 на позицію 0
ImageList1->GetBitmap(1,Bmp); //отримати об'єкт зі
списку по індексу 1
Image1->Picture->Bitmap=Bmp; //нарисувати зображення
Bmp в Image1
//нарисувати зображення з індексом 0 на формі
ImageList1->Draw(Form1->Canvas,10,10,0,true);
ImageList1->Delete(1); //видалити зображення з
індексом 1
```

TRichEdit – багаторядкове поле редагування з розширеним форматуванням

```
//конструктор форми
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
RichEdit1->Lines->LoadFromFile("1.rtf"); //зчитати
рядки з файлу
RichEdit1->WordWrap=false; //заборонити переніс
тексту
RichEdit1->DefAttributes->Color = clBlue; //колір по
замовчуванню
RichEdit1->DefAttributes->Style=RichEdit1-
>DefAttributes->Style << fsBold << fsItalic; //стиль
по замовчуванню
RichEdit1->Lines->Add("First line"); //добавити рядок
"First line"
RichEdit1->Lines->Insert(0,"Second line"); //вставити
рядок "Second line" в позицію 0
}
//натиснута кнопка Button1
void __fastcall TForm1::Button1Click(TObject *Sender)
{
RichEdit1->SelAttributes->Color = clRed; //колір
виділеного тексту
RichEdit1->SelAttributes->Height = 14; //висота
виділеного тексту
RichEdit1->Paragraph->Alignment = taCenter;
//вирівнювання поточного абзацу
String S=RichEdit1->SelText; //виділений текст
for(int i=1;i<=RichEdit1->SelLength;i++)S[i]='*';
//змінити всі символи на '*'
RichEdit1->SelText=S; //вставити рядок S замість
виділеного
RichEdit1->Lines->SaveToFile("1.rtf"); //зберегти
рядки у файл
}
```

TTrackBar - повзунок

```
//конструктор форми
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
```

```
{
TrackBar1->Orientation=trVertical; //орієнтація
вертикальна
TrackBar1->Frequency=5; //частота поділок
TrackBar1->Min=-10; //мінімальне значення
TrackBar1->Max=20; //максимальне значення
TrackBar1->SelStart=0; //мінімальне значення
виділеного
TrackBar1->SelEnd=20; //максимальне значення
виділеного
TrackBar1->Position=10; //позиція повзунка
}
//при зміні TrackBar1
void __fastcall TForm1::TrackBar1Change(TObject
*Sender)
{Label1->Caption=TrackBar1->Position;} //вивести в
надпис Label1 поточну позицію повзунка
```

TProgressBar – індикатор прогресу

```
ProgressBar1->Smooth=true; //неперервне відображення
ProgressBar1->Max=10; //максимальне значення
ProgressBar1->Step=1; //крок приросту
ProgressBar1->Position++; //збільшити позицію на 1
ProgressBar1->StepBy(2); //збільшити позицію на 2
ProgressBar1->StepIt(); //збільшити позицію на
значення кроку
```

TUpDown – спарені кнопки вверх-вниз, лічильник

```
UpDown1->Associate=Edit1; //пов'язане з UpDown1 поле
Edit1
 UpDown1->Wrap=true; //переносити при досягненні
граничного значення
 UpDown1->Min=2; //мінімальне значення
 UpDown1->Max=10; //максимальне значення
 UpDown1->Increment=2; //приріст
 int i=UpDown1->Position; //поточне значення
```

THotKey – клавіша гарячого виклику

Добавте на форму компоненти HotKey, ActionList та два компоненти Button. Створіть дію Action1. Запрограмуйте події OnExecute для Action1 та OnClick для Button1.

```
//при натиску кнопки Button1
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
//присвоїти дії Action1 клавіші гарячого виклику
Action1->ShortCut=HotKey1->HotKey;
Button2->Action=Action1; //пов'язати кнопку Button2 з
дією Action1
}
//при виконанні дії Action1
void __fastcall TForm1::Action1Execute(TObject
*Sender)
//змінити гарячі клавіші на "Ctrl+O"
{HotKey1->HotKey=ShortCut(Word('O'), TShiftState() <<
ssCtrl);}
```

TAnimate – анімація відеозаписів

```
Animate1->Active=false; //не програвати кліп
 Animate1->CommonAVI=aviFindComputer; //стандартний
кліп avi
//програвати всі кадри нескінченно
 Animate1->Play(Animate1->StartFrame,Animate1-
>StopFrame,0);
 Sleep(1000); //зупинити виконання програми на 1
секунду
 Animate1->Stop(); //зупинити програвання
 Animate1->FileName="1.avi"; //завантажити файл 1.avi
//програвати всі кадри один раз
 Animate1->Play(Animate1->StartFrame,Animate1-
>StopFrame,1);
 Sleep(3000); //зупинити виконання програми на 3
секунди
 Animate1->Seek(5); //показати п'ятий кадр
```

TDateTimePicker – вибір дати/часу, TDateTime – дата/час

```
DateTimePicker1->ShowCheckbox=true; //показувати
індикатор
if(DateTimePicker1->Checked) //якщо індикатор вибрано
{
DateTimePicker1->Kind=dtkTime; //тип - час
DateTimePicker1->Time=Time(); //присвоїти поточний
час
}
else //інакше
{
```

```
DateTimePicker1->Kind=dtkDate; //тип - дата
DateTimePicker1->Date=Date(); //присвоїти поточну
дату
DateTimePicker1->DateFormat=dfLong; //формат дати
DateTimePicker1->DateMode=dmComboBox; //спосіб зміни
дати
}
//присвоїти змінній DT значення властивості DateTime
TDateTime DT=DateTimePicker1->DateTime;
Edit1->Text=DT.DateString(); //вивести в поле Edit1
дату
Edit2->Text=DT.TimeString(); //вивести в поле Edit2
час
```

TMonthCalendar - календар, TDate - дата

```
TDate DMin=EncodeDate(2007,9,3); //присвоїти DMin
вказану дату
TDate DMax=Date()+50; //присвоїти DMax поточну
дату+50 днів
MonthCalendar1->MinDate=DMin; //встановити мінімальну
дату
MonthCalendar1->MaxDate=DMax; //встановити
максимальну дату
TDate D=MonthCalendar1->Date; //присвоїти D вибрану
дату
unsigned short y,m,d; //рік, місяць і день
D.DecodeDate(&y,&m,&d); //декодувати дату
Edit1->Text=d; //вивести в поле день
```

TTreeView – деревовидний список

```
//конструктор форми
__fastcall TForm1::TForm1(TComponent* Owner):
TForm(Owner)
{
 TTreeNode *N; //вказівник на вузол дерева
 TreeView1->Items->Clear(); //очистити вузли
 TreeView1->Items->Add(NULL,"#1"); //добавити вузол
"#1"
 N= TreeView1->Items->Item[0]; //вказівник вказує на
перший вузол
 TreeView1->Items->Add(N,"#2"); //добавити вузол "#2"
 TreeView1->Items->Add(N,"#3"); //добавити вузол "#3"
```

```
 TreeView1->Items->AddChild(N,"+1"); //добавити
дочірній вузол "+1"
 N= TreeView1->Items->Item[1]; //вказівник вказує на
другий вузол
 TreeView1->Items->AddChild(N,"1"); //добавити
дочірній вузол "1"
 TreeView1->Items->Add(N,"+2"); //добавити вузол "+1"
 Edit1->Text=TreeView1->Items->Item[4]->Text; //текст
п'ятого вузла
}
//при натиску на кнопці Button2
void __fastcall TForm1::Button2Click(TObject *Sender)
{
//вивести в поле текст вибраного вузла і його індекс
Edit1->Text=(TreeView1->Selected->Text)+"-
"+(TreeView1->Selected->Index);
//вставити на місце вибраного вузла вузол "New"
TreeView1->Items->Insert(TreeView1->Selected,"New");
//видалити вибраний вузол
TreeView1->Items->Delete(TreeView1->Selected);
}
```

TListView – табличний список елементів

```
//при створенні форми
void __fastcall TForm1::FormCreate(TObject *Sender)
{
 TListColumn  *Col; //колонка списку
 TListItem  *ListItem; //елемент списку
 ListView1->ViewStyle=vsReport; //стиль списку
 ListView1->Checkboxes=true; //показувати прапорці
 ListView1->ShowColumnHeaders=true; //показувати
заголовки колонок
//створити три колонки
 for (int i = 0; i < 3; i++)
 {
 Col = ListView1->Columns->Add(); //добавити колонку
 Col->Caption = "Col "+IntToStr(i); //змінити надпис
колонки
 }
//створити п'ять елементів з піделементами
 for (int i = 0; i < 5; i++)
 {
```

```
 ListItem = ListView1->Items->Add(); //добавити
елемент (рядок)
 ListItem->Caption = "Row "+IntToStr(i); //змінити
надпис рядку
 ListItem->SubItems->Add(i); //добавити піделемент
 ListItem->SubItems->Add(i+1); //добавити піделемент
 }
}
//при натиску на колонці
void __fastcall TForm1::ListView1ColumnClick(TObject
*Sender,
      TListColumn *Column)
//вивести в поле надпис і індекс колонки
{ Edit1->Text=Column->Caption+" "+Column->Index;}
//при виборі елемента
void __fastcall TForm1::ListView1SelectItem(TObject
*Sender,
      TListItem *Item, bool Selected)
{
//якщо прапорець елемента вибраний і індекс більше 1
if((Item->Checked)&&(Item->Index>1))
//вивести в поле надпис елемента і першого
піделемента
Edit1->Text=Item->Caption+" "+Item->SubItems-
>Strings[0];
}
```

THeaderControl - заголовки

```
//при створенні форми
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//добавити дві секції з надписами
 HeaderControl1->Sections->Add()->Text="1";
 HeaderControl1->Sections->Add()->Text="2";
 }
//при перетягуванні секції
void __fastcall TForm1::HeaderControl1SectionTrack(
      THeaderControl *HeaderControl, THeaderSection
*Section, int Width,
      TSectionTrackState State)
{
//координата Left Edit1 дорівнює координаті Left
першої  секції
```

```
 Edit1->Left=HeaderControl->Sections->Items[0]->Left;
//ширина Edit1 дорівнює ширині першої  секції
 Edit1->Width=HeaderControl->Sections->Items[0]-
>Width;
//координата Left Shape1дорівнює координаті Left
другої секції
 Shape1->Left=HeaderControl->Sections->Items[1]-
>Left;
//ширина Shape1дорівнює ширині другої секції
 Shape1->Width=HeaderControl->Sections->Items[1]-
>Width;
}
```

TStatusBar – рядок стану

```
//конструктор форми
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
Form1->Cursor=crCross; //змінити вид курсору
StatusBar1->Panels->Add(); //добавити панель
StatusBar1->Panels->Add(); //добавити панель
}
//при переміщенні курсору миші над формою
void __fastcall TForm1::FormMouseMove(TObject
*Sender, TShiftState Shift,
      int X, int Y)
{
//X,Y – координати курсору
StatusBar1->Panels->Items[0]->Text=IntToStr(X);
//змінити текст на X
StatusBar1->Panels->Items[1]->Text=IntToStr(Y);
//змінити текст на Y
StatusBar2->SimpleText=IntToStr(X)+","+IntToStr(Y);
//змінити текст на X,Y
}
```

TToolBar – панель інструментів

Для створення панелі інструментів ToolBar на етапі проектування і зв’язування її елементів керування з головним меню виконайте дії:

1 Помістіть на форму компоненти: MainMenu1, ActionList1, ImageList1, ToolBar1.

2 Створіть пункти меню MainMenu1.

3 Створіть дії ActionList1 і функції-обробники подій OnExecute.
4 Добавте рисунки в ImageList1.
5 Добавте кнопки в ToolBar1.
6 Змініть властивість Images компонентів ActionList1 і ToolBar1 на ImageList1.
8 Змініть властивість ImageIndex компонента Action1 на 0.
9 Змініть властивість Action одного пункта MainMenu1 на Action1.
10 Змініть властивість Action однієї кнопки ToolBar1 на Action1.

Для створення панелі інструментів ToolBar динамічно:
1 Добавте на форму компонент ActionList1 і добавте дію Action1.
2 Створіть для неї обробник події OnExecute.
3 Добавте в модуль директиву: #include <comctrls.hpp> і код:

```
//при створенні форми
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//створити контрольну панель Bar1
 TControlBar *Bar1= new TControlBar(Form1);
 Bar1->Parent=Form1; //батьківський компонент - форма
 Bar1->Left=0; //координата Left
 Bar1->Width=Form1->Width; //ширина дорівнює ширині
форми
//створити панель інструментів Bar2
 TToolBar *Bar2= new TToolBar(Bar1);
 Bar2->Parent=Bar1; //батьківський компонент - Bar1
 Bar2->Left=0; //координата Left
 Bar2->ShowCaptions = True; //показувати надписи
 Bar2->Height = 40; //висота
//створити кнопку Button
 TToolButton *Button = new TToolButton(Bar2);
 Button->Parent = Bar2; //батьківський компонент –
Bar2
 Button->Action=Action1; //пов’язати з дією Action1
 }
//відбулась дія Action1
void __fastcall TForm1::Action1Execute(TObject
*Sender)
//вивести повідомлення "Execute"
{Application->MessageBox("Execute","Action1",0);}
```

TCoolBar – панель зі шторками

```
//створити панель зі шторками
```

```
TCoolBar *CoolBar = new TCoolBar(this);
CoolBar->Parent = this; //батьківський компонент
CoolBar->Align = alTop; //вирівнювання
CoolBar->Bands->Add(); //добавити смугу
CoolBar->Bands->Items[0]->Text ="Band"; //змінити
надпис першої смуги
TButton *Button = new TButton(CoolBar); //створити
кнопку
Button->Caption="Button"; //надпис на кнопці
Button->Parent=CoolBar; //батьківський компонент -
CoolBar
CoolBar->Bands->Items[0]->Control=Button; //компонент
що належить до смуги - Button
```

TPageScroller – панель з прокручуванням

1 Помістіть на форму компоненти Panel1 і PageScroller1.
2 На панелі Panel1 розмістіть довільну кількість інших компонентів.
3 Змініть властивість Control компонента PageScroller1 на Panel1.

TComboBoxEx – комбінований список зі значками

```
ComboBoxEx1->Images=ImageList1; //список рисунків для
ComboBoxEx1
ComboBoxEx1->ItemsEx->AddItem("1",0,0,0,0,NULL);
//добавити елемент
ComboBoxEx1->ItemsEx->AddItem("2",1,1,1,1,NULL);
//добавити елемент
ComboBoxEx1->ItemsEx->AddItem("3",1,1,1,0,NULL);
//добавити елемент
```

4.7 Компоненти System

TTimer - таймер

```
//конструктор форми
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
Form1->Caption="0"; //надпис на формі
Timer1->Interval=2000; //інтервал таймеру – 2 секунди
Timer1->Enabled=true; //включити таймер
}
//пройшов інтервал часу
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
```

```
int t=StrToInt(Form1->Caption); //t – кількість
мілісекунд
//якщо t<10000 змінити надпис форми
if (t<10000)Form1->Caption=t+Timer1->Interval;
else Timer1->Enabled=false; //інакше виключити таймер
}
```

TPaintBox – область рисування, TCanvas - канва, TBrush - пензель, TPen - перо

```
//при необхідності перерисування зображення
void __fastcall TForm1::PaintBox1Paint(TObject
*Sender)
{
//створити графічний об’єкт bmp
Graphics::TBitmap *bmp = new Graphics::TBitmap();
bmp->LoadFromFile("1.bmp"); //завантажити з файлу
PaintBox1->Canvas->Draw(30,50,bmp); //нарисувати bmp
в заданих координатах
PaintBox1->Canvas->TextOutA(10,100,"Hello");
//вивести текст в заданих координатах
PaintBox1->Canvas->Ellipse(100,100,200,300);
//нарисувати еліпс
PaintBox1->Canvas->Brush->Color=clRed; //колір пензля
PaintBox1->Canvas-
>FloodFill(110,200,clBlack,fsBorder); //залити площу
в заданих координатах
PaintBox1->Canvas->Brush->Bitmap=bmp; //рисунок
пензля - bmp
PaintBox1->Canvas->FloodFill(20,5,clBlue,fsSurface);
//залити площу в заданих координатах
PaintBox1->Canvas->Pen->Width=5; //ширина пера
PaintBox1->Canvas->Pen->Style=psDashDotDot; //стиль
пера
PaintBox1->Canvas->MoveTo(20,20); //перемістити перо
в точку
PaintBox1->Canvas->LineTo(50,150); //рисувати лінію
до точки
for(int x=0;x<200;x+=2)
//змінити колір пікселів з координатами [x][x] на
червоний
{PaintBox1->Canvas->Pixels[x][x]=clRed;}
TPoint p[3]; //масив точок
p[0]=Point(0,0); //перша точка
```

```
p[1]=Point(100,90); //друга точка
p[2]=Point(250,50); //третя точка
PaintBox1->Canvas->Pen->Color = clBlue; //колір пера
PaintBox1->Canvas->Polygon(p,3); //нарисувати полігон
по точкам p
PaintBox1->Canvas->Polyline(p,3); //нарисувати
полілінію по точкам p
PaintBox1->Canvas->PolyBezier(p,3); //нарисувати
лінію Без'є по точкам p
}
```

TMediaPlayer – мультимедіа плейер

```
//конструктор форми
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
MediaPlayer1->FileName="1.avi"; //ім'я файлу
мультимедіа
MediaPlayer1->Open(); //відкрити
MediaPlayer1->Display=Panel1; //відображати вміст на
Panel1
MediaPlayer1->TimeFormat = tfFrames; //формат часу -
кадри
Caption=MediaPlayer1->Length; //вивести в надпис
форми загальну кількість кадрів у файлі
}
//при натиску кнопки на MediaPlayer1
void __fastcall TForm1::MediaPlayer1Click(TObject
*Sender,
      TMPBtnType Button, bool &DoDefault)
{
DoDefault=false; //заборонити використання кнопок по
замовчуванню
switch (Button) //якщо натиснута:
  {
    case btPlay: //кнопка "програвати"
      MediaPlayer1->StartPos=3; //встановити
початкову позицію
      MediaPlayer1->Play(); //програвати
      break;
    case btPause: //кнопка "пауза"
//встановити позицію на попередню
```

```
      MediaPlayer1->Position=MediaPlayer1->Position-
1;
      MediaPlayer1->Pause(); //пауза
      break;
    case btStep: //кнопка "крок"
      MediaPlayer1->Frames=1; //кількість кадрів у
кроці
      MediaPlayer1->Step(); //перемістити на задану у
Frames кількість кадрів
      break;
    case btNext: //кнопка "наступний"
//програти 5 раз фрагмент починаючи з позиції 2 до
позиції 4
    for (int i=1;i<5;i++)
      {
      MediaPlayer1->StartPos=2;
      MediaPlayer1->EndPos=4;
      MediaPlayer1->Wait=true;
      MediaPlayer1->Play();
      }
      break;
   }
}
```

TOleContainer – контейнер OLE

```
//при створенні форми
void __fastcall TForm1::FormCreate(TObject *Sender)
{
if (OleContainer1->InsertObjectDialog())//якщо
вибраний OLE об’єкт з діалогу
//вивести ім’я класу в заголовку форми
Caption=OleContainer1->OleClassName;
//інакше
else
 if (OpenDialog1->Execute())//якщо вибраний файл
//створити OLE об’єкт з файлу
 OleContainer1->CreateObjectFromFile(OpenDialog1-
>FileName,false);
//створити другий OLE об’єкт класу Equation.3
OleContainer2->CreateObject("Equation.3", false);
//завантажити з файлу
OleContainer2->LoadFromFile("1.ole");
//заповнити список ListBox1 списком команд OLE
контейнера
```

```
ListBox1->Items=OleContainer2->ObjectVerbs;
OleContainer2->DoVerb(1);//виконати другу команду
}
//при натиску кнопки Button1
void __fastcall TForm1::Button1Click(TObject *Sender)
{
OleContainer2->SaveToFile("1.ole");//зберегти об'єкт
у файл
OleContainer1->SaveAsDocument("1");//зберегти як
документ
OleContainer2->DestroyObject();//знищити об'єкт
}
```

TDdeServerConv, TDdeServerItem, TDdeClientConv, TDdeClientItem – компоненти для динамічного обміну даними

1 Створюємо DDE сервер, зберігаємо проект під назвою «Project1_s»:

```
//при створенні форми
void __fastcall TForm1::FormCreate(TObject *Sender)
{
DdeServerItem1->ServerConv=DdeServerConv1;//вказує
об'єкт для елемента
DdeServerItem1->Lines->Add("Hello");//додає рядки в
елемент
DdeServerItem1->Lines->Add("Hello2");
Edit1->Text=DdeServerItem1->Lines-
>Strings[0];//виводить рядок в Edit1
Edit2->Text=DdeServerItem1->Lines-
>Strings[1];//виводить рядок в Edit2
}
//при зміні Edit1
void __fastcall TForm1::Edit1Change(TObject *Sender)
{
DdeServerItem1->Lines->Strings[0]=Edit1-
>Text;//змінити рядок в елементі
}
//при зміні Edit2
void __fastcall TForm1::Edit2Change(TObject *Sender)
{
DdeServerItem1->Lines->Strings[1]=Edit2-
>Text;//змінити рядок в елементі
}
```

2 Створюємо DDE клієнта:

```
//при створенні форми
void __fastcall TForm1::FormCreate(TObject *Sender)
{
DdeClientConv1->ConnectMode=ddeManual;//режим
з'єднання
DdeClientItem1->DdeConv=DdeClientConv1;//вказує
об'єкт для елемента
DdeClientItem1->DdeItem="DdeServerItem1";//вказуємо
на елемент сервера
}
//натиснута кнопка Button1
void __fastcall TForm1::Button1Click(TObject *Sender)
{
//якщо установлений зв'язок з сервером
if(DdeClientConv1-
>SetLink("Project1_s","DdeServerConv1"))
{
if(DdeClientConv1->OpenLink())//якщо відкритий
зв'язок
{
//запит даних з елементу сервера і розміщення їх в
елементі клієнта
DdeClientConv1->RequestData(DdeClientItem1->DdeItem);
DdeClientConv1->FormatChars=true;//установка
форматування рядків
//вивід рядків елементу в Edit1 і Edit2
Edit1->Text=DdeClientItem1->Lines->Strings[0];
Edit2->Text=DdeClientItem1->Lines->Strings[1];
DdeClientConv1->CloseLink();//закрити зв'язок
}
else Caption="Not OpenLink";
}
else Caption="Not SetLink";
}
```

4.8 Діалогові вікна

TOpenDialog – відкриття файлу, TSaveDialog – збереження файлу

```
//при натиску кнопки "прочитати"
void __fastcall TForm1::Button1Click(TObject *Sender)
{
OpenDialog1->DefaultExt = "TXT";//розширення файлу по
замовчуванню
```

```
OpenDialog1->FileName = "*.txt";//ім'я файлу
OpenDialog1->Title="Open text";//заголовок діалогу
OpenDialog1->InitialDir="C:\\";//початковий каталог
if (OpenDialog1->Execute())//показати діалог і, якщо
вибір файлу зроблений
//прочитати рядки з вибраного файлу
Memo1->Lines->LoadFromFile(OpenDialog1->FileName);
}
//при натиску кнопки "зберегти"
void __fastcall TForm1::Button2Click(TObject *Sender)
{
SaveDialog1->Options<<ofAllowMultiSelect;//дозволити
вибір багатьох файлів
if (SaveDialog1->Execute())//показати діалог і, якщо
вибір зроблений
{
//цикл з першого по останній рядок у списку вибраних
файлів
for (int i=0;i<SaveDialog1->Files->Count;i++)
//зберегти рядки у кожен файл
Memo1->Lines->SaveToFile(SaveDialog1->Files-
>Strings[i]);
}
}
//при спробі закрити діалог без відміни
void __fastcall TForm1::SaveDialog1CanClose(TObject
*Sender,
      bool &CanClose)
{//цикл з першого по останній рядок у списку вибраних
файлів
for (int i=0;i<SaveDialog1->Files->Count;i++)
//якщо рядки зі списку файлів не містять ".txt", не
закривати діалог
if (!SaveDialog1->Files->Strings[i].Pos(".txt"))
CanClose=false;
}
```

4.9 Робота з базами даних

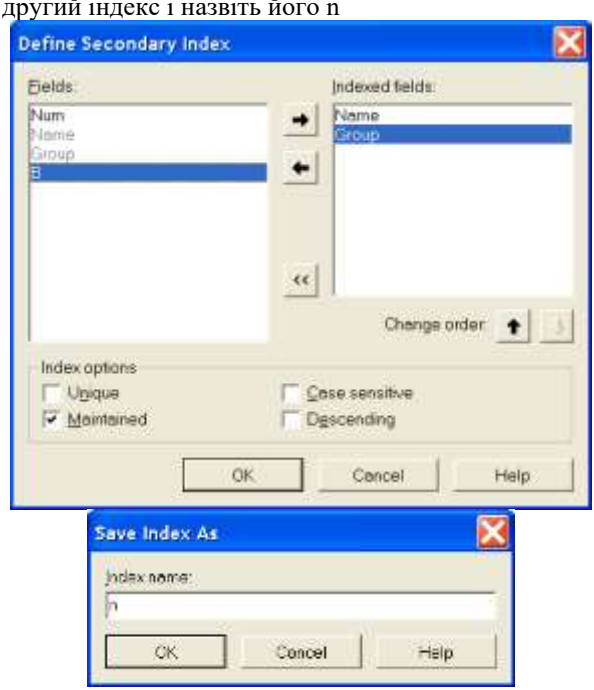
Робота з базами даних у C++ Builder за допомогою BDE (TDataSource, TDatabase, TTable, TQuery, TDBGrid, TDBNavigator)

1 Створіть базу даних Paradox за допомогою Database Desktop. Запустіть програму C:\Program Files\Common Files\Borland Shared\Database Desktop\dbd32.exe

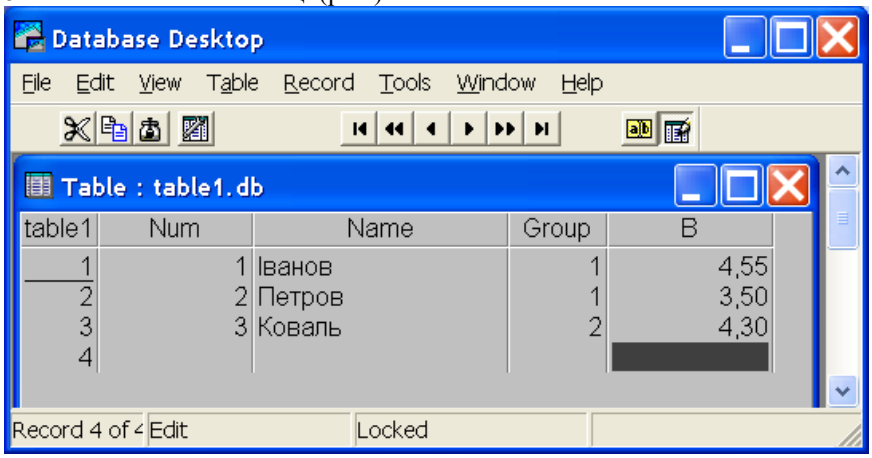
2 Створіть нову таблицю File/New/Table.../Paradox 7

3 Створіть поля таблиці (рис.) і збережіть її як C:\My_db \table1.bd

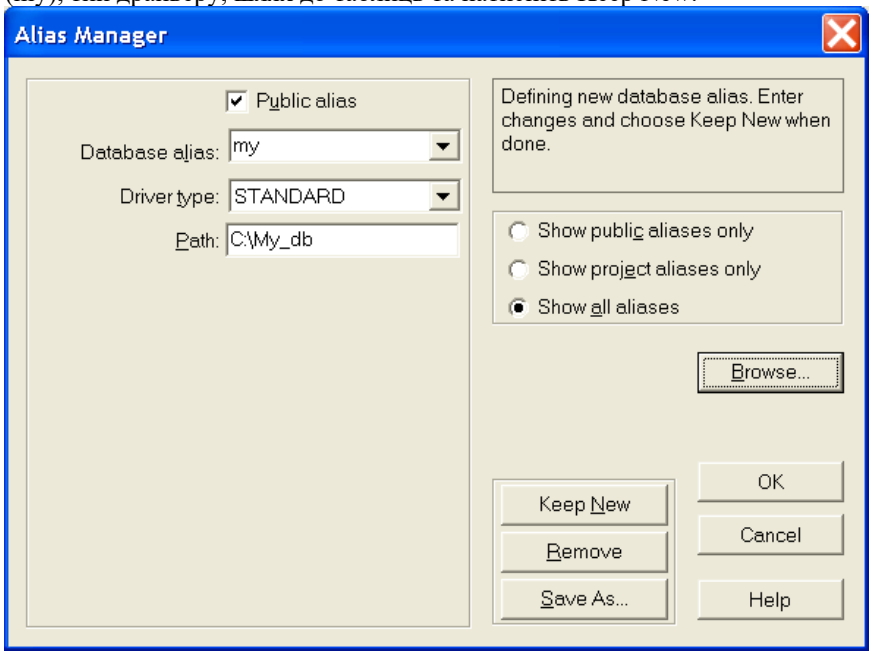
4 Визначте другий індекс і назвіть його n



5 Заповніть записи таблиці (рис.)



6 Виберіть в меню Tools/Alias Manager...(рис.) Задайте псевдонім бази (my), тип драйверу, шлях до таблиць та натисніть Keep New.



7 Створіть проект C++ Builder (*File|New| Application*)
8 Помістіть на форму такі компоненти: TDBGrid, TDBNavigator, TButton (5 штук), TDataSource, TTable, TQuery.

9 Властивості компонентів для роботи з базами даних можна виставити вже на етапі проектування і вже на етапі проектування переглядати вміст бази даних, але в даному випадку, з навчальною метою, значення властивостей задаються в коді програми.
10 Заповніть код програми:

```
//обробник події створення форми
void __fastcall TForm1::FormCreate(TObject *Sender)
{
 Table1->Active=false;//деактивувати таблицю
 DataSource1->DataSet=Table1;//вказати набір даних
для джерела даних
 DBGrid1->DataSource=DataSource1;//вказати джерело
даних для таблиці
 DBGrid1->Width=Form1->ClientWidth;//установити
ширину таблиці
 DBNavigator1->DataSource=DataSource1;//вказати
джерело даних для навігатора
 DBNavigator1->ShowHint=true;//показувати підказки
кнопок
 Table1->DatabaseName="my";//псевдонім БД
 Table1->TableName="table1.db";//таблиця БД
 Table1->Active=true;//активувати таблицю
 Table1->IndexName="n";//задати ім'я індексу
 Table1->Fields->Fields[0]->Visible=false;//сховати
поле 0
 Table1->FieldByName("Name")-
>DisplayLabel="Прізвище";//змінити надпис поля Name
 Table1->Fields->Fields[2]-
>DisplayLabel="Група";//змінити надпис поля 2
 Table1->Fields->Fields[3]-
>DisplayLabel="Бал";//змінити надпис поля 3
 Table1->EnableConstraints();//дозволити обмеження
 //установити обмеження для поля 3
 Table1->Fields->Fields[3]->CustomConstraint="x>=0
and x<=5";
 Table1->Fields->Fields[3]-
>ConstraintErrorMessage="!";
 DBGrid1->Columns->Items[2]-
>ButtonStyle=cbsAuto;//стиль колонки 2
 //список вибору колонки 2
 DBGrid1->Columns->Items[2]->PickList->Add(3);
 DBGrid1->Columns->Items[2]->PickList->Add(4);
```

```
 DBGrid1->Columns->Items[2]->PickList->Add(5);
 //змінити надписи на кнопках
 Button1->Caption="Insert";
 Button2->Caption="Filter";
 Button3->Caption="Edit";
 Button4->Caption="Find";
 Button5->Caption="Query";
 TStringList *ss=new TStringList;
 Session->GetAliasNames(ss);//записати у список імена
псевдонімів БД
 Session-
>GetTableNames("my","",true,false,ss);//записати у
список імена таблиць БД my
 }
//обробник події натиску на кнопці Button1
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 Table1->Insert();//вставити запис
 //заповнити поля
 Table1->FieldByName("Name")->AsString="Шевченко";
 Table1->FieldByName("Group")->AsInteger=2;
 Table1->FieldByName("B")->AsFloat=3.8;
 Table1->Post();//надіслати у БД
 //заборонити зміну поля Group
 Table1->FieldByName("Group")->ReadOnly=true;
}
//обробник події натиску на кнопці Button2
void __fastcall TForm1::Button2Click(TObject *Sender)
{
 if (Table1->Filtered)//якщо дані фільтровані
 {Table1->Filtered=false;//відмінити фільтрацію
 Table1->CancelRange();}//відмінити діапазони
 else//інакше
 {Table1->Filter="(Group=1)and(B<5)";//фільтр мовою
SQL
 Table1->Filtered=true;//установити фільтрацію
 Table1->IndexFieldNames="Name";//ім'я поля індексу
 Table1-
>SetRange(&TVarRec("H"),0,&TVarRec("Я"),0);//установи
ти діапазон імен від H до Я
 }
}
//обробник події натиску на кнопці Button3
```

```
void __fastcall TForm1::Button3Click(TObject *Sender)
{
 String s="Мельник";//рядок
 Table1->Edit();//редагувати
 Table1->FieldByName("Name")->AsString=s;//запис у
поле рядка
 int i=Table1->FieldByName("Group")-
>AsInteger;//читання поля
 Table1->FieldByName("B")->AsFloat=4.1;//запис у поле
дійсного числа
 Table1->Post();//надіслати у БД
}
//обробник події натиску на кнопці Button4
void __fastcall TForm1::Button4Click(TObject *Sender)
{
 //перший варіант пошуку
 String s="Мельник";//рядок
 Table1->First();//перехід до першого запису
 while (!Table1->Eof)//поки не кінець таблиці
 {
 //якщо вміст поля Name=s, видалити цей запис
 if(Table1->FieldByName("Name")->AsString==s)Table1-
>Delete();
 Table1->Next();//перехід по наступного запису
 }
 //другий варіант пошуку
 bool find=false;//булева змінна
 TLocateOptions options;//опції пошуку
 //часткове співпадання і нечутливість до регістру
 options<<loPartialKey<<loCaseInsensitive;
 //результат пошуку в полі Name присвоїти змінній
find
 find=Table1->Locate("Name","Шевченко",options);
 //або пошук по кільком полям
 find=false;
 Variant v[]={"Шевченко","1"};
 //результат пошуку в полях Name і Group присвоїти
find
 find=Table1-
>Locate("Name;Group",VarArrayOf(v,1),options);
 //третій варіант пошуку
 //присвоїти змінній b вміст поля B знайденого запису
```

```
 float b=StrToFloat(Table1-
>Lookup("Name","Шевченко","B"));
 //четвертий варіант пошуку
 Table1->IndexFieldNames="Name;Group";
 //перемістити курсор до найбільш подібного запису
 Table1-
>FindNearest(OPENARRAY(TVarRec,("Шевченко","1")));
}
 //обробник події натиску на кнопці Button5
void __fastcall TForm1::Button5Click(TObject *Sender)
{
 Table1->Active=false;//деактивувати таблицю
 Query1->DatabaseName=Table1-
>DatabaseName;//псевдонім БД для запиту
 DataSource1->DataSet=Query1;//вказати набір даних
для джерела даних
 //створити параметр P
 TParam *p = Query1->Params->CreateParam(ftFloat,
"P", ptInput);
 p->AsFloat = 4.1;//присвоїти число
 //сформувати рядок динамічного запиту SQL з
параметром
 String s="SELECT * FROM table1 WHERE (B>:P)";
 try
 {
 Query1->SQL->Clear();//очистити рядки команд SQL
 Query1->SQL->Add(s);//добавити рядок команди SQL
 Query1->Open();//виконати запит
 }
 catch (EDatabaseError&)//при помилці
 {Query1->ExecSQL();}//виконати інші команди SQL
(крім SELECT)
 delete p;//знищити p
}
//обробник події, яка виникає перед записом у БД
void __fastcall TForm1::Table1BeforePost(TDataSet
*DataSet)
{
//якщо не вибрана кнопка YES
if (Application-
>MessageBoxA("Записати?","",MB_YESNOCANCEL)!=IDYES)
{
DataSet->Cancel();//відмінити зміни
```

```
Abort();
}
}
```

Інший варіант доступу до бази даних (за допомогою TDatabase)

1 Помістіть на форму компоненти TDBGrid, TDBNavigator, TDataSource, TTable, TDatabase

2 Заповніть код програми:

```
void __fastcall TForm2::FormCreate(TObject *Sender)
{
 Database1->DatabaseName="Mybase";//ім'я БД
 Database1->DriverName="STANDARD";//драйвер BDE
 //задати параметри з'єднання для BDE
 Database1->Params->Add("PATH=C:\\My_db");
 Database1->Params->Add("DEFAULT DRIVER=PARADOX");
 Database1->Params->Add("ENABLE BCD=FALSE");
 Table1->Active=false;//деактивувати таблицю
 DataSource1->DataSet=Table1;//вказати набір даних
для джерела даних
 DBGrid1->DataSource=DataSource1;//вказати джерело
даних для таблиці
 DBNavigator1->DataSource=DataSource1;//вказати
джерело даних для навігатора
 Table1->DatabaseName="Mybase";//псевдонім БД
 Table1->TableName="table1.db";//таблиця БД
 Table1->Active=true;//активувати таблицю
 Table1->IndexName="n";//задати ім'я індексу
}
```

Робота з базами даних у C++ Builder за допомогою ADO (TADOConnection, TADOTable, TADOQuery)

1 Створіть базу даних у MS Access по аналогії з попередньою. Заповніть таблицю **Таблица1**. Збережіть базу даних у файлі D:\db1.mdb.

2 Помістіть на форму компоненти TDBGrid, TDBNavigator, TMemo, TButton, TDataSource, TADOTable, TADOConnection, TADOQuery

3 Для доступу до баз даних **Paradox** потрібно установити джерело даних за допомогою стандартної програми **Источники данных (ODBC)** . Запустіть програму ...\system32\odbcad32.exe і задайте DSN користувача (задайте **myparadox**), вибравши драйвер для **Paradox**.

4 Властивість **ConnectionString** компонента **ADOConnection1** можна задати за допомогою помічника на етапі проектування.

5 Заповніть код програми

```
//обробник події створення форми
```

```
void __fastcall TForm3::FormCreate(TObject *Sender)
{
 ADOTable1->Active=false;//деактивувати таблицю
 DataSource1->DataSet=ADOTable1;//вказати набір даних
для джерела даних
 DBGrid1->DataSource=DataSource1;//вказати джерело
даних для таблиці
 DBNavigator1->DataSource=DataSource1;//вказати
джерело даних для навігатора
 DBNavigator1->ShowHint=true;//показувати підказки
кнопок
 //установити з'єднання з БД Access
 ADOConnection1-
>ConnectionString="Provider=Microsoft.Jet.OLEDB.4.0;D
ata Source=D:\\db1.mdb;Persist Security Info=False";
 //або установити з'єднання з БД Paradox
//Увага! В рядку слід замінити символи \ на \\ і " на
""
 //ADOConnection1-
>ConnectionString="Provider=MSDASQL.1;Persist
Security Info=False;Extended
Properties=""DSN=myparadox;DBQ=C:\\MY_DB;DefaultDir=C
:\\MY_DB;DriverId=538;FIL=Paradox
5.X;MaxBufferSize=2048;PageTimeout=600;""";
 ADOTable1->Connection=ADOConnection1;//указати
з'єднання для таблиці
 ADOTable1->TableName="Таблица1";//таблиця БД Access
 //ADOTable1->TableName="table1";//або таблиця БД
Paradox
 ADOTable1->Active=true;//активувати таблицю
 Memo1->ScrollBars=ssVertical;
 Memo1->Text=ADOConnection1-
>ConnectionString;//відобразити рядок з'єднання
}
//обробник події натиску на кнопці Button1
void __fastcall TForm3::Button1Click(TObject *Sender)
{
 DataSource1->DataSet=ADOQuery1;//вказати набір даних
для джерела даних
 ADOQuery1->Connection=ADOConnection1;//указати
з'єднання для запиту
 ADOQuery1->SQL->Add("SELECT * FROM Таблица1 WHERE
(B=4)");//сформувати рядок запиту SQL
```

```
 ADOQuery1->Open();//виконати запит
}
```

4.10 Компоненти Indy для роботи з мережею (TIdTCPServer, TIdPeerThread, TIdTCPClient)

Постановка задачі: розробити програму для обміну текстовими повідомленнями через мережу.

Послідовність виконання:

1 Помістіть на форму програми-сервера компоненти `TIdTCPServer`, `TMemo` і заповніть код:

```
//конструктор форми
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
IdTCPServer1->DefaultPort=8090;//задати порт
IdTCPServer1->Active=true;//зробити сервер активним
Memo1->Lines->Clear();//очистити Memo1
}
//обробник події OnExecute компонента IdTCPServer1
//AThread - потік, який використовується для
клієнтських з'єднань
void __fastcall
TForm1::IdTCPServer1Execute(TIdPeerThread *AThread)
{
 try
 {//добавити в Memo1 отриманий рядок
  Memo1->Lines->Add(AThread->Connection->ReadLn());
  //добавити в Memo1 віддалену IP адресу
  Memo1->Lines->Add(AThread->Connection->Binding-
>PeerIP);
  //надіслати локальну IP адресу
  AThread->Connection->WriteLn(AThread->Connection-
>Binding->IP);
 }
 catch(...)
 {//при помилці надіслати "помилка"
  AThread->Connection->WriteLn("помилка");
 }
  AThread->Connection->Disconnect();//роз'єднати
з'єднання
}
```

3 Для можливості роботи програми-сервера на комп'ютері без середовища Borland C++ Builder 6 задайте опції проекту: Project/Options.../Packages/зніміть прапорець Build with runtime packages. На цей комп'ютер також необхідно скопіювати файли: borlndmm.dll і cc3260mt.dll.

4 Скомпілюйте програму-сервер.

5 Помістіть на форму програми-клієнта компоненти `TEdit` (2 штуки), `TButton`, `TIdTCPClient`, і заповніть код:

```
//конструктор форми
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
Edit1->Text="127.0.0.1";//локальний адрес
}
//обробник події OnClick
void __fastcall TForm1::Button1Click(TObject *Sender)
{
IdTCPClient1->Host=Edit1->Text;//вказати адресу
віддаленого комп'ютера
IdTCPClient1->Port=8090;//вказати порт
try
 {
  IdTCPClient1->Connect();//з'єднати з сервером
  IdTCPClient1->WriteLn(Edit2->Text);//надіслати
рядок
  Edit2->Text = IdTCPClient1->ReadLn();//отримати
рядок
 }
  catch(...)
  {//при помилці вивести повідомлення
   ShowMessage("Помилка з'єднання");
  }
  IdTCPClient1->Disconnect();//роз'єднати з'єднання
}
```

6 Скомпілюйте програму-клієнт.

7 Скопіюйте програму-сервер та файли borlndmm.dll і cc3260mt.dll на віддалений комп'ютер з адресою, наприклад, 192.168.1.5. Виконайте програму-сервер на віддаленій машині і програму клієнт на локальній. Введіть в Edit1 клієнта адресу віддаленого комп'ютера і передайте йому будь-який текст. Для тестування програм без мережі на локальній машині слід вводити адресу 127.0.0.1.

5 Створення компонентів VCL

Постановка задачі: розробити компонент VCL `TButtonEx` на основі класу `TButton`, який додатково має властивість `State` логічного типу, яка визначає стан компоненту (`true` або `false`). Стан (`State`) компоненту змінюється на протилежний при натиску на кнопку (коли відбувається подія `OnClick`).

Послідовність виконання:

1 Закриваємо все: *File/Close All*

2 Створюємо новий компонент: *File/New/Other.../Component/Ok*

3 В діалоговому вікні вводимо: Ancestor type – TButton [StdCtrls], Class name – TButtonEx, Palette page – Samples, UnitFileName – вказуємо C:\Program Files\Borland\CBuilder6\Projects\ButtonEx.cpp. Натискаємо *Ok*.

4 Зберігаємо все: *File/Save All*, закриваємо все: *File/Close All*

5 Створюємо новий додаток: *File/New/Application*

6 В unit1.cpp дописуємо директиву `#include "ButtonEx.cpp"`

7 В модулі unit1.cpp після рядка `TForm1 *Form1;` оголосіть об'єкт типу `TButtonEx`:

```
TButtonEx *btn;
```

8 За допомогою *Class Explorer* в клас `TButtonEx` добавляємо властивість *New Property...*

9 В діалоговому вікні вводимо: Property name – `State`, Type – `bool`, create Set method, implement in: ButtonEx.cpp, create field, use this field for the read specifier, use this field for implementing the Set method, Default: `true`.

10 За допомогою *Class Explorer* в клас `TButtonEx` добавляємо метод *New Method...*

11 В діалоговому вікні вводимо: Method name – `Click`, Arguments – `void`, Function result – `void`, Visibility – `public`, Directives - `__fastcall`, Call Inherited.

12 В класі `TButtonEx` позначте цей метод словом `DYNAMIC` так:

```
DYNAMIC void __fastcall Click(void);
```

13 Вставте в реалізацію цього методу код:

```
(FState)?this->SetState(false):this->SetState(true);
```

перед рядком

```
TButton::Click();
```

14 За допомогою *Class Explorer* в клас `TForm1` добавляємо метод *New Method...*

15 В діалоговому вікні вводимо: Method name – `btn_click`, Arguments – `TObject *Sender`, Function result – `void`, Visibility – `public`, Directives - `__fastcall`.

16 В реалізацію методу вставте код:
```
if (btn->State) btn->Caption="true";
else btn->Caption="false";
Application->MessageBoxA("Changed","State",0);
```
17 В конструктор класу `TForm1::TForm1` вставте код:
```
btn=new TButtonEx(this);
btn->Parent=this;
btn->Left=100;
btn->Top=100;
btn->OnClick=btn_click;
```
18 Зберігаємо все: *File/Save All*.
19 Виконайте програму: *Run/Run*.
20 *Component/Install Component*, виберіть *Into new package* і вкажіть назву пакету TButtonEx.
21 Закриваємо все: *File/Close All*
22 Приклад використання нового компоненту. Створіть новий додаток, який використовує новий компонент `TButtonEx`. Добавте на форму компонент `ButtonEx` і створіть обробник події `OnClick`. В функцію `TForm1::ButtonEx1Click` вставте код:
```
Application->MessageBoxA(BoolToStr(ButtonEx1->State,
true).c_str(),"State",0);
```
і виконайте програму.

Код компоненту TButtonEx
Файл ButtonEx.h

```
//файл ButtonEx.h з класом компоненту TButtonEx
#ifndef ButtonExH
#define ButtonExH
#include <SysUtils.hpp>
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
//клас компоненту TButtonEx, який успадковується від
TButton
//макрос PACKAGE вказує, що клас може бути
імпортований і експортований з файлу bpl
class PACKAGE TButtonEx : public TButton
{
private:
        bool FState;//поле для збереження значення
властивості State
```

```
        void __fastcall SetState(bool value);//метод
запису властивості State
protected:
public:
        __fastcall TButtonEx(TComponent*
Owner);//конструктор
       DYNAMIC void __fastcall
Click(void);//динамічний метод Click
__published://розділ з властивостями, які показуються
у Object Inspector
        //оголошення властивості State логічного типу
        //читання з поля FState, запис методом
SetState, по замовчуванню true
        __property bool State  = { read=FState,
write=SetState, default=true };
};
#endif
```

Файл ButtonEx.cpp

```
//файл ButtonEx.cpp з визначенням методів класу
TButtonEx
#include <vcl.h>
#pragma hdrstop
#include "ButtonEx.h"
#pragma package(smart_init)
//використовується для перевірки, чи немає чисто
віртуальних функцій
static inline void ValidCtrCheck(TButtonEx *)
{
        new TButtonEx(NULL);
}
//конструктор
__fastcall TButtonEx::TButtonEx(TComponent* Owner):
TButton(Owner)
{}//пустий
namespace Buttonex//простір імен Buttonex
{
        void __fastcall PACKAGE Register()//функція
реєстрації компоненту
        {
                TComponentClass classes[1] =
{__classid(TButtonEx)};
```

```
                RegisterComponents("Samples",
classes, 0);//реєструє на вкладці Samples
        }
}
//визначення методу SetState, який установлює
значення властивості
void __fastcall TButtonEx::SetState(bool value)
{
//якщо поле FState не рівне value, то присвоїти value
if(FState != value) {FState = value;}
}
//визначення методу Click, який уточнює реакцію на
подію OnClick
void __fastcall TButtonEx::Click(void)
{
//якщо FState істина, то викликати SetState(false),
інакше SetState(true)
(FState)?this->SetState(false):this->SetState(true);
TButton::Click();//виклик методу базового класу
}
```

6 Створення DLL

Постановка задачі: створити бібліотеку dll, яка містить простий клас.
Послідовність виконання:
Створюємо бібліотеку DLL:
1 *File/New/Other/DLL Wizard*
2 *Source Type: C++*
3 Забрати прапорець з: *Use VCL*, *Ok*
4 Добавляємо файл заголовків File1.h: *File/New/Other/Header File*
5 *File/Save/File1.h*
6 *Project/Add To Project.../File1.h*
7 Вставте у файл File1.h код:

```
__declspec( dllexport ) class My
{
  private:
     double x;
  public :
     My();
     double RetSqr();
};
```

8 Добавте до Unit1.cpp:

```
#include "File1.h"
```

а також реалізацію методів:

```
My::My()
{
   x=2;
}
double My::RetSqr()
{
   return x*x;
}
```

9 Компілюємо: *Project/Build Project1*

Створюємо додаток, який використовує DLL:

10 Створюємо додаток: *File/New/Application*

11 Помістіть на форму поле і кнопку.

12 Добавте до модуля cpp:

```
#include "File1.h"
```

13 Добавляємо до проекту бібліотеку: *Project/Add To Project.../Project1.lib*

14 Добавте в клас `TForm1` в розділ `private`:

```
My *obj;
```

15 В конструктор класу `TForm1::TForm1` вставте код:

```
obj = new My();
```

16 В метод `TForm1::Button1Click` вставте код:

```
{
Edit1->Text=AnsiString(obj->RetSqr());
}
```

17 *File/Save Project As...* Назвіть модуль Unit2, а проект Project2.

18 *Run*.

Повинен появитись результат: 4.

7 Створення і використання компонентів COM

Постановка задачі: створити компонент-сервер COM з методом, який повертає число помножене на 2, створити клієнта COM, який використовує об'єкт COM у C++ Builder і VBA.

Послідовність виконання:

Спочатку створюємо і реєструємо бібліотеку типів Typelib2

(Очистіть папку з минулими проектами.)

1 *File/Close All*

2 *File/New/Other/ActiveX/Type Library*

3 Добавляємо новий інтерфейс: *New Interface*

4 Добавляємо новий метод: *New Method*

5 Добавляємо в метод параметр для запису:
Parameters/Add/Type = double, Modifier=In

6 Добавляємо в метод параметр для читання:

Parameters/Add/Type =double, Modifier=Out*
7 *Refresh Implementation*
8 Реєструємо бібліотеку: *Register Type Library*

Створюємо COM об'єкт
9 Створюємо додаток типу EXE: *File/New/Application*
10 Створюємо COM об'єкт: *File/New/Other/COM Object*
11 Називаємо CoClass: *CoClass Name=My*
12 Вказуємо раніше створений інтерфейс:
Interface/List/Add/Interface1(знайти у списку)
13 *Refresh Implementation*
14 У метод `TMyImpl::Method1` вставте код:

```
{
*Param2=Param1*2;
  return S_OK;
}
```

15 *Refresh Implementation*
16 *Run*
17 *File/Save Project As...*
18 *File/Close All*

Створюємо COM клієнта
19 Створюємо додаток: *File/New/Application*
20 Помістіть на форму поле і кнопку.
21 Добавте до модуля cpp:

```
#include "Project1_TLB.cpp"
```

22 В метод `TForm1::Button1Click` вставте код:

```
{
TCOMInterface1 obj;
obj=Project1_tlb::CoMy::Create();
double x;
obj->Method1(2,&x);
Edit1->Text=AnsiString(x);
}
```

23 *File/Save Project As...* Назвіть модуль Client, а проект ProjectClient.
24 *Run*.
Повинен появитись результат: 4.

Створюємо COM клієнта у VBA
25 На форму добавляємо поле і кнопку.
26 Вибираємо створену бібліотеку типів:
Tools/References/Project1 Library
27 Записуємо код:

```
Private Sub CommandButton1_Click()
Dim obj As New My
Dim x As Double
obj.Method1 2, x
TextBox1.Text = x
End Sub
```

28 *Run*

Повинен появитись результат: 4.

8 Створення компоненту ActiveX у C++ Builder

Постановка задачі: створити і використати ActiveX компонент на основі класу VCL `TButton`.

Послідовність виконання:

1 Закрити все: *File/Close All*

2 Добавити новий ActiveX компонент: *File/New/Other.../ActiveX Control*

3 Вибрати VCL клас, на основі якого створюється ActiveX компонент і ім'я ActiveX компоненту:

VCL Class Name: `TButton`,

New ActiveX Name: `ButtonX`,

Натиснути *OK*.

4 Зберегти проект: *File/Save Project As...*

5 Зареєструвати ActiveX сервер: *Run/Register ActiveX Server*

Використання створеного ActiveX компоненту ButtonX

Послідовність виконання:

1 Відкрити MS Excel

2 Вибрати *Сервис/Макрос/Редактор Visual Basic*

3 Вставити нову форму: *Insert/User Form*

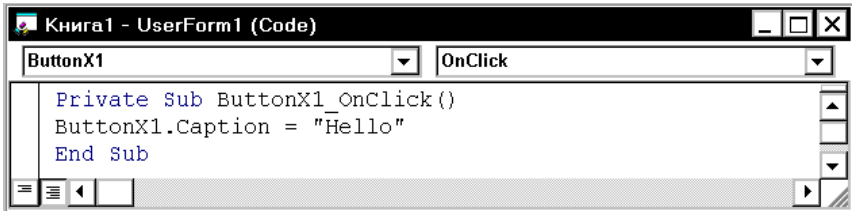
4 Добавити додатковий компонент ButtonX: *Tools/Additional Controls, вибрати ButtonX Control*

5 Добавте компонент ButtonX1 на форму

6 Відкрити код програми: *View/Code*

7 Добавте в код процедуру (див. рис.):

```
Private Sub ButtonX1_OnClick()
ButtonX1.Caption = "Hello"
End Sub
```



8 Виконати програму: *Run/Run*

Відміна реєстрації ActiveX серверу ButtonX

Послідовність виконання:

1 Відкрити збережений проект у C++ Builder: *File/Open Project.../ButtonXControl1.bpr*

2 Відмінити реєстрацію: *Run/Unregister ActiveX Server*

9 Створення активної форми (Active Form)

Постановка задачі: створити і використати активну форму ActiveX.

Послідовність виконання:

1 Закрити все: *File/Close All*

2 Добавити нову форму Active Form : *File/New/Other.../Active Form*

3 Помістити на форму три компоненти `TEdit` і один `TButton`

4 Запрограмуйте функцію оброки події `OnClick` для `Button1`:

```
void    __fastcall   TActiveFormX::Button1Click(TObject
*Sender)
{
float a,b;
a=StrToFloat(Edit1->Text);
b=StrToFloat(Edit2->Text);
Edit3->Text=a+b;
}
```

5 Зберегти проект: *File/Save Project As...*

6 Зареєструвати ActiveX сервер: *Run/Register ActiveX Server*

Використання створеної форми ActiveFormX

Послідовність виконання:

1 Відкрити MS Excel

2 Вибрати *Сервис/Макрос/Редактор Visual Basic*

3 Вставити нову форму: *Insert/User Form*

4 Добавити додатковий компонент ActiveFormX: *Tools/Additional Controls, вибрати ActiveFormX Control*

5 Добавте компонент ActiveFormX на форму

6 Виконати програму: *Run/Run* (див.рис.)

10 Робота з компонентами-серверами COM MS Word і MS Excel

Постановка задачі: створити додаток, який використовує компоненти-сервери COM MS Word і MS Excel.

Послідовність виконання:

1 Створюємо додаток: *File/New/Application*

2 Добавте на форму компоненти `TWordApplication`, `TExcelApplication`, `TExcelWorksheet`.

3 Створіть функцію-обробник події `OnCreate` форми `Form1`:

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
WordApplication1->Connect();//з'єднатись з сервером
WordApplication1->set_Visible(true);//зробити видимим
TVariant Visible=true;
WordApplication1->Documents-
>Add(EmptyParam,EmptyParam,EmptyParam,Visible);
//добавити документ
WordApplication1->Selection-
>InsertAfter(TVariant("Hello!\n"));
//вставити "Hello!"
TVariant FileName="hello.doc";
//зберегти документ
WordApplication1->ActiveDocument->SaveAs(&FileName);
WordApplication1->Disconnect();//роз'єднатись з
сервером

ExcelApplication1->Connect();//з'єднатись з сервером
//добавити робочу книгу
ExcelApplication1->Workbooks->Add(EmptyParam,0);
//в активну комірку записати число 1
ExcelApplication1->ActiveCell-
>set_Value(TVariant(1));
//в комірку A2 записати число 2
ExcelApplication1-
>get_Range(TVariant("A2"),EmptyParam)-
>set_Value(TVariant(2));
//з'єднатись з сервером
ExcelWorksheet1->ConnectTo(ExcelApplication1-
>ActiveSheet);
//створити вказівник на комірку A3
Excel_2k::RangePtr r=ExcelWorksheet1-
>get_Range(TVariant("A3"),EmptyParam);
//в комірку A3 записати число 3
```

```
r->set_Value(TVariant(3));
//вказівник вказує на комірку (4,1) (або "A4")
r=ExcelWorksheet1->Cells-
>get__Default(TVariant(4),TVariant(1));
//в комірку (4,1) записати число 4
r->set_Value(TVariant(4));
TVariant FileName2="hello.xls";
//зберегти робочу книгу
ExcelApplication1->ActiveWorkbook-
>SaveAs(FileName2,EmptyParam,
EmptyParam,EmptyParam,EmptyParam,EmptyParam,0,EmptyPa
ram,EmptyParam,EmptyParam,EmptyParam,0);
ExcelWorksheet1->Disconnect();//роз'єднатись з
сервером
ExcelApplication1->Disconnect();//роз'єднатись з
сервером
}
```

4 Збережіть усе: *File/Save All*
5 Виконайте програму: *Run*.

11 Створення компонентів CORBA

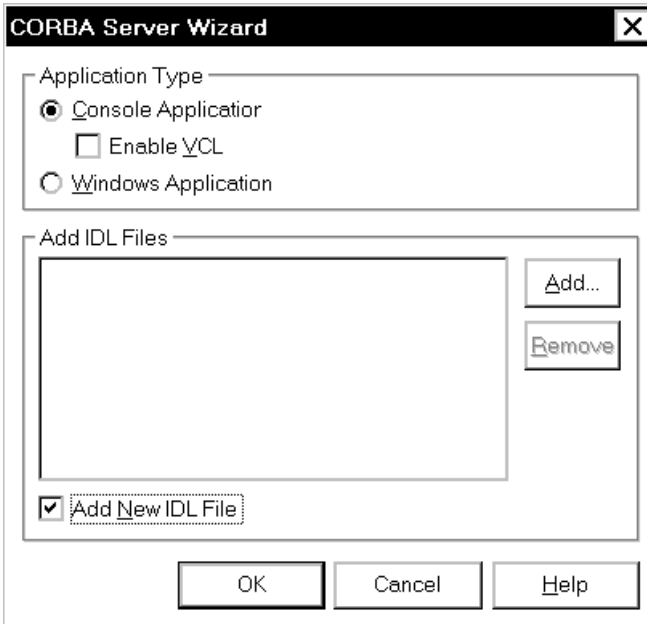
Постановка задачі: створити компонент-сервер CORBA з методом, який повертає квадрат числа, створити клієнт CORBA, який використовує об'єкт CORBA.

Послідовність виконання:

Створення сервера CORBA

Перед початком роботи установіть Borland VisiBroker for C++ 4.5 в папку c:\Inprise\vbroker

1 Закрийте існуючі проекти: *File/Close All*
2 Вибрати прапорці на *View/Toolbars/CORBA* і *Tools/VisiBroker SmartAgent*
3 Створіть CORBA сервер: *File/New/Other/Multitier/CORBA Server*
4 Виберіть тип серверу: *Console Applcation* та добавте новий IDL файл: *Add New IDL File*



5 У файл з розширенням .idl вставте код:

```
module MyModule
{
  interface My
  {
    float f(in float x);
  };
};
```

6 Збережіть усе: *File Save All*, назвіть файл .idl MyServ, а проект Server

7 Вкажіть шлях до бібліотек vboker:

Project/Options/Directories|Conditionals

Натисніть кнопку "..." справа від *Include Path* і виправте

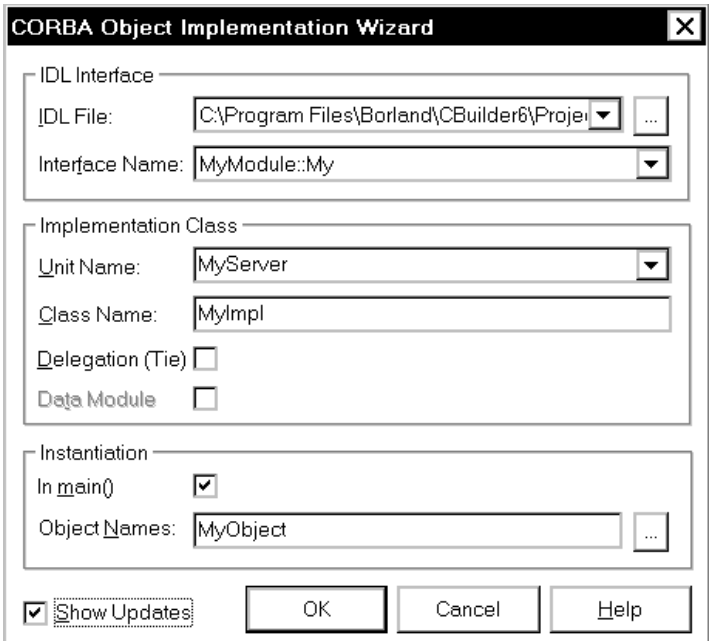
$(VBROKERDIR)\include на C:\Inprise\vbroker\include

Натисніть кнопку "..." справа від *Library Path* і виправте

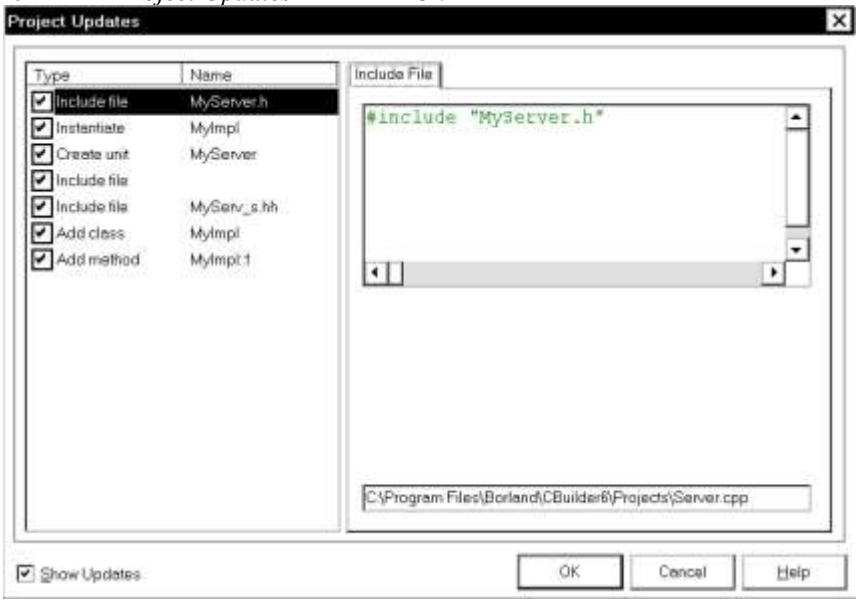
$(VBROKERDIR)\lib на C:\Inprise\vbroker\lib

8 Натисніть кнопку *New CORBA Object Implementation*

9 Виберіть *Interface Name*: MyModule::My, та виберіть *Show Updates*



10 У вікні *Project Updates* натисніть *Ok*



11 Змініть реалізацію функції `f` наступним чином:

```
CORBA::Float MyImpl::f(CORBA::Float _x)
```

```
{
 return (_x*_x);
}
```
12 Скомпілюйте проект: *Project/Build Server*

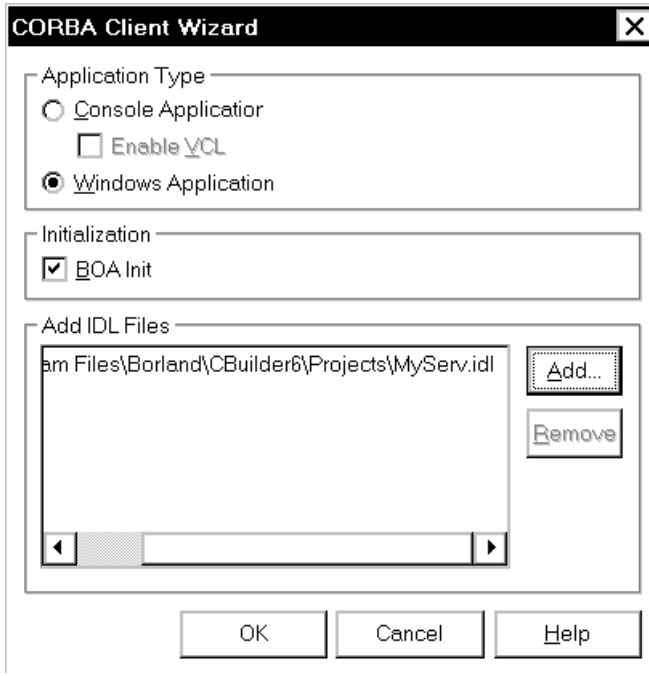
Створення клієнта CORBA

Послідовність виконання:

1 Закрийте існуючі проекти: *File/Close All*

2 Створіть CORBA клієнт: *File/New/Other/CORBA Client*

3 Виберіть Application Type: *Windows Application*, добавте файл (*Add IDL Files*) MyServ.idl



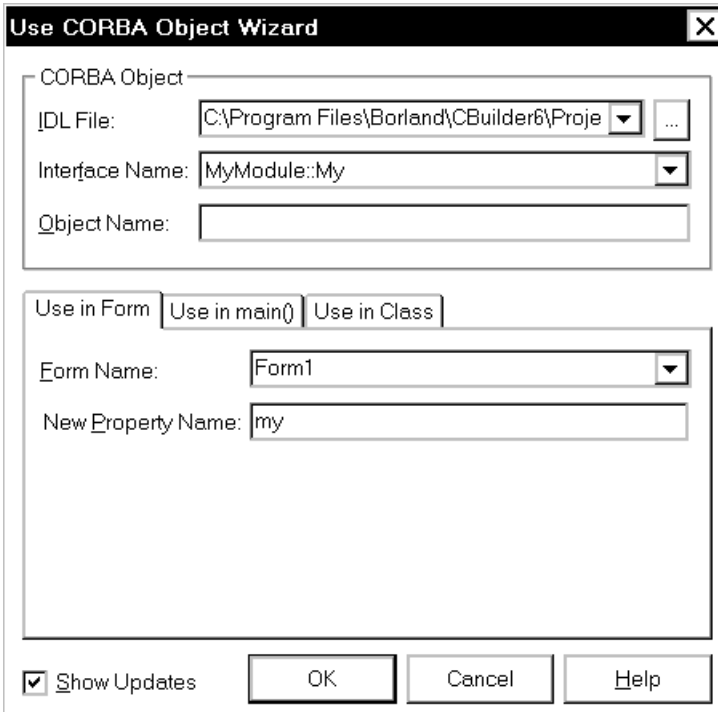
4 Добавте на форму компоненти поле Edit і кнопку Button

5 *File/Save All*

6 Вкажіть шлях до бібліотек vboker

7 Натисніть кнопку *Use CORBA Object* (при помилці добавте в файл idl пустий рядок)

8 Виберіть *Interface Name*: MyModule::My



9 Добавте обробник події `OnClick` для `Button1`:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Edit1->Text=my->f(StrToFloat(Edit1->Text));
}
```

10 Скомпілюйте проект: *Project/Build Project1*

11 Виконайте програму Server.exe і, не закриваючи її, програму Project1.exe

Наприклад, при вводі в поле числа 2 і натиску кнопки повинен появитись результат: 4

12 Бібліотека основних класів Microsoft MFC

CEdit - поле редагування (1)

```
//обробник події BN_CLICKED (виконано натиск на
кнопці)
void CMy2Dlg::OnBnClickedButton1()
{
double a,b,c;//дійсні змінні
```

```
CString s;//рядок
m_edit1.GetWindowTextW(s);//отримати текст з поля
редагування
a=_wtof(s);//конвертувати в дійсне
m_edit2.GetWindowTextW(s); );//отримати текст з поля
редагування
b=_wtof(s); //конвертувати в дійсне
c=a+b;
s.Format(_T("%.3f"),c);//конвертувати дійсне в рядок
m_edit3.SetWindowTextW(s);//вивести текст у поле
редагування
}
```

CEdit - поле редагування (2)

```
//обробник події BN_CLICKED
void CMy4Dlg::OnBnClickedButton2()
{
double x;//дійсна змінна
UpdateData();//помістити дані з поля редагування у
рядок s1 типу CString
x=_wtof(s1); //конвертувати в дійсне
x+=1;//добавити 1
s1.Format(_T("%10.3f"),x);//конвертувати дійсне в
рядок
UpdateData(false);//помістити дані з рядка s1 у поле
редагування
}
```

CButton - прапорець

```
//обробник події BN_CLICKED
void CMy5Dlg::OnBnClickedButton1()
{
      UpdateData();//помістити дані з елементів
керування у змінні
      if (chk1) chk1=false;//якщо прапорець
установлений то забрати його
      else chk1=true;//інакше установити його
      UpdateData(false);//помістити дані зі змінних у
елементи керування
}
```

CButton - кнопка (1)

```
//обробник події BN_CLICKED
void CMy2Dlg::OnBnClickedButton1()
```

```
{
m_button1.SetWindowTextW(L"hello");//змінити надпис
на кнопці m_button1 класу CButton
}
```

CButton - кнопка (2)

```
//обробник події BN_CLICKED
void CMy2Dlg::OnBnClickedButton1()
{
      wchar_t s2[64];//рядок типу wchar_t
      static int i=0;//статична змінна цілого типу
      _itow(i++,s2,10);//збільшити на 1 і конвертувати
в рядок
      CString s=L"Click ";//рядок типу CString
      s+=s2;//добавити до рядка s рядок s2
      m_button1.SetWindowTextW(s);//установити надпис
на кнопці
}
```

CButton - перемикач

```
//обробник події BN_CLICKED
void CMy5Dlg::OnBnClickedButton1()
{
CButton
*b=(CButton*)GetDlgItem(IDC_RADIO1);//отримати
вказівник типу CButton * на об'єкт з ідентифікатором
IDC_RADIO1
if (b->GetCheck()==BST_UNCHECKED)//якщо перемикач b
не вибрано
{
b->SetCheck(BST_CHECKED);//вибрати його
b->SetWindowTextW(L"Checked");//установити надпис на
перемикачі
}
else //інакше
{
b->SetCheck(BST_UNCHECKED);//установити невибраний
стан
b->SetWindowTextW(L"Unchecked");}//установити надпис
на перемикачі
}
//обробник події BN_CLICKED
void CMy5Dlg::OnBnClickedRadio1()
{
```

```
CWnd *w=GetDlgItem(IDC_STATIC); //отримати вказівник
типу CWnd * на об'єкт з ідентифікатором IDC_STATIC
(Group Box)
w->SetWindowTextW(L"1");//установити надпис на
IDC_STATIC
}
//обробник події BN_CLICKED
void CMy5Dlg::OnBnClickedRadio2()
{
CWnd *w=GetDlgItem(IDC_STATIC); //отримати вказівник
типу CWnd * на об'єкт з ідентифікатором IDC_STATIC
(Group Box)
w->SetWindowTextW(L"2");//установити надпис на
IDC_STATIC
}
```

CListBox - список

```
//обробник події BN_CLICKED
void CMy5Dlg::OnBnClickedButton2()
{
//m_list1-список класу CListBox
m_list1.AddString(L"String1");//добавити рядок у
список
m_list1.DeleteString(1);//видалити рядок 1
m_list1.InsertString(1,L"String2");//вставити рядок
на місце 1
CString s;//рядок
m_edit.GetWindowTextW(s);//отримати текст з поля
редагування
m_list1.AddString(s);//добавити рядок у список
s.Format(_T("%d"),m_list1.GetCount());//помістити в
рядок кількість елементів списку
m_edit.SetWindowTextW(s);//помістити текст у поле
редагування
}
//обробник події LBN_SELCHANGE (вибір змінено)
void CMy5Dlg::OnLbnSelchangeList1()
{
CString s;//рядок
m_list1.GetText(m_list1.GetCurSel(),s);//помістити в
рядок s текст з вибраного рядка списку
m_edit.SetWindowTextW(s);//помістити текст у поле
редагування m_edit
```

```
}
```

CComboBox - комбінований список

```
//обробник події BN_CLICKED
void CMy1Dlg::OnBnClickedButton1()
{
	//m_combo1-комбінований список класу CComboBox
	CString s;//рядок
	m_combo1.GetWindowTextW(s);//отримати текст з
поля редагування комбінованого списку m_combo1
	m_combo1.AddString(s);//добавити рядок у список
	m_combo1.AddString(L"String1");//добавити рядок
у список
	m_combo1.InsertString(1,L"String2");//вставити
рядок у список в позицію 1
	m_combo1.SetCurSel(1);//вибрати позицію 1
}
//обробник події CBN_SELCHANGE (вибір змінено)
void CMy1Dlg::OnCbnSelchangeCombo1()
{
	CString s;//рядок
	m_combo1.GetLBText(m_combo1.GetCurSel(),s);//пом
істити в рядок s текст з вибраного рядка списку
	m_button1.SetWindowTextW(s);//установити надпис
на кнопці m_button1
}
```

CString - рядок

```
//обробник події BN_CLICKED
void CMy1Dlg::OnBnClickedButton2()
{
 UpdateData();//отримати дані з елементів керування
 CString s("hello");//рядок s класу CString
 s+=m_edit1;//добавити вміст рядка m_edit1 класу
CString
 s.Trim();//видалити пробіли
 if (s!=m_edit1) s=s.Mid(2,4);//якщо рядки s і
m_edit1 не рівні то помістити в рядок s 4 символи
рядка s починаючи з 2 позиції
 s.SetAt(1,L'*');//установити в позицію 1 символ '*'
 s.Insert(1,L"string");//вставити в позицію 1 рядок
"string"
```

```
 int i=s.Find(L"str",0);//знайти позицію рядка "str"
в рядку s
 s.Format(L"%s%d%s%.3f",m_edit1,s.GetLength(),"
",2.2513);//помістити в рядок дані різного типу
 wchar_t *p=s.GetBuffer();//перетворити в тип wchar_t
*
 s=p;//присвоїти рядку CString дані типу wchar_t *
 char *p2=new char[s.GetLength() + 1];//рядок типу
char * довжиною s+1
 wcstombs_s(0, p2, s.GetLength() + 1, s,
_TRUNCATE);//перетворити в тип char *
 s=p2;//присвоїти рядку CString дані типу char *
 double x;//дійсна змінна
 s="-12.4583";
 x=_wtof(s);//конвертувати рядок в дійсне
 int n;//ціла змінна
 s="-10";
 n=_wtoi(s);//конвертувати рядок в ціле
 m_edit1=s;//присвоїти рядку m_edit1 рядок s
 UpdateData(false);//помістити дані в елементи
керування
}
```

CImage - рисунок, CDC - контекст пристрою

```
void CMy6Dlg::OnBnClickedButton1()
{
//отримати вказівник на Picture Control
CStatic *s=(CStatic*)GetDlgItem(IDC_STATIC);
//рисунок
CImage image;//включіть atlimage.h у ваш файл
stdafx.h
CFileDialog dlg(true);//діалог відкриття файлу
dlg.DoModal();//зробити модальним
image.Load(dlg.GetFileName());//завантажити рисунок
CDC *pDC=s->GetDC();//контекст пристрою Picture
Control
CDC *iDC=CDC::FromHandle(image.GetDC());//контекст
пристрою рисунку
iDC->MoveTo(20,0);//перемістити поточну позицію
iDC->LineTo(20,20);//рисувати лінію з поточної
позиції
image.ReleaseDC();//вивільнити DC
```

```
image.Draw(pDC->GetSafeHdc(),0,0);//рисувати рисунок
на Picture Control
CFileDialog sdlg(false);//діалог збереження файлу
sdlg.DoModal();//зробити модальним
image.Save(sdlg.GetFileName());//зберегти рисунок
}
```

CClientDC - контекст пристрою, CPoint - точка

```
//обробник події WM_MOUSEMOVE
void CMy7Dlg::OnMouseMove(UINT nFlags, CPoint point)
{
if (nFlags&&MK_LBUTTON)//якщо натиснута ліва клавіша
миші
{
static CPoint anchor=point;//точка для збереження
попередніх координат
CClientDC *pDC=new CClientDC(this);//отримати
вказівник на контекст пристрою
pDC->MoveTo(anchor.x,anchor.y);//перемістити до точки
anchor
pDC->LineTo(point.x,point.y);//рисувати лінію до
точки point
anchor=point;//запам'ятати точку
delete pDC;//знищити pDC
}
CDialog::OnMouseMove(nFlags, point);//обробник
базового класу
}
```

Створення додатків Win32 з графічним інтерфейсом

Постановка задачі: за допомогою бібліотеки MFC створити додаток Win32 з графічним інтерфейсом. В головному вікні програми розмістити елемент керування – поле, запрограмувати функцію-обробник повідомлення натиску на кнопку, яка виводить в поле слово "Hello".
Примітка: Borland C++ Builder 6 містить бібліотеку MFC 6.0 та дозволяє конвертацію проектів Microsoft Visual C++.

Код програми

```
//My_MFC.h : файл з класом додатку
#pragma once
#ifndef __AFXWIN_H__
```

```
        #error "include 'stdafx.h' before including
this file for PCH"
#endif
#include "resource.h"//заголовочний файл з макросами
ресурсів
//клас додатку CMy_MFCApp успадкований від CWinApp
class CMy_MFCApp : public CWinApp
{
public:
CMy_MFCApp();//конструктор
virtual BOOL InitInstance();//функція ініціалізації
DECLARE_MESSAGE_MAP()//декларація карти повідомлень
};
//специфікатор вказує, що theApp визначений в
окремому файлі
extern CMy_MFCApp theApp;

//My_MFCDlg.h : файл з класом діалогового вікна
#pragma once
#include "afxwin.h"
//клас діалогового вікна CMy_MFCDlg успадкований від
CDialog
class CMy_MFCDlg : public CDialog
{
public:
CMy_MFCDlg(CWnd* pParent = NULL);//стандартний
конструктор
enum { IDD = IDD_MY_MFC_DIALOG };
protected:
virtual void DoDataExchange(CDataExchange*
pDX);//підтримка DDX/DDV
protected:
HICON m_hIcon;//іконка
//функції-обробники повідомлень
virtual BOOL OnInitDialog();
afx_msg void OnPaint();
afx_msg HCURSOR OnQueryDragIcon();
DECLARE_MESSAGE_MAP()//декларація карти повідомлень
public:
afx_msg void OnBnClickedOk();
public:
CEdit edit1;//елемент керування поле edit1
};
```

```
// My_MFC.cpp : визначення поведінки класу додатку
#include "stdafx.h"//заголовочні файли MFC
#include "My_MFC.h"//файл класу додатку
#include "My_MFCDlg.h"//файл класу діалогу
#ifdef _DEBUG
#define new DEBUG_NEW
#endif
//макрос карти повідомлень для CMy_MFCApp, базовий
клас CWinApp
BEGIN_MESSAGE_MAP(CMy_MFCApp, CWinApp)
        ON_COMMAND(ID_HELP,
&CWinApp::OnHelp)//відповідає OnHelp
END_MESSAGE_MAP()
//конструктор CMy_MFCApp
CMy_MFCApp::CMy_MFCApp()
{}
//створити об’єкт-додаток класу CMy_MFCApp
CMy_MFCApp theApp;
//ініціалізація додатку
BOOL CMy_MFCApp::InitInstance()
{
CWinApp::InitInstance();//виклик функції базового
класу
//зберігати інформацію додатку в реєстрі
SetRegistryKey(_T("Local AppWizard-Generated
Applications"));
CMy_MFCDlg dlg;//створити об’єкт-діалогове вікно
класу CMy_MFCDlg
m_pMainWnd = &dlg;//указує на головне вікно
INT_PTR nResponse = dlg.DoModal();//викликати як
модальне
if (nResponse == IDOK)//якщо натиснуто OK
{}
else if (nResponse == IDCANCEL)//якщо натиснуто
Cancel
{}
return FALSE;//вихід з додатку
}

// My_MFCDlg.cpp : визначення поведінки класу
діалогового вікна
```

```
//у файлі stdafx.h підключені заголовочні файли MFC
такі як:
//#include <afxwin.h>//ядро MFC і стандартні
компоненти
//#include <afxext.h>//розширення MFC
#include "stdafx.h"
#include "My_MFC.h"//файл класу додатку
#include "My_MFCDlg.h"//файл класу діалогу
#ifdef _DEBUG
#define new DEBUG_NEW
#endif
//конструктор CMy_MFCDlg
CMy_MFCDlg::CMy_MFCDlg(CWnd* pParent /*=NULL*/):
CDialog(CMy_MFCDlg::IDD, pParent)
{m_hIcon = AfxGetApp()-
>LoadIcon(IDR_MAINFRAME);}//завантажити іконку
//виконує обмін даними членів з елементами керування
void CMy_MFCDlg::DoDataExchange(CDataExchange* pDX)
{
CDialog::DoDataExchange(pDX);
DDX_Control(pDX, IDC_EDIT1, edit1);
}
//макрос карти повідомлень класу CMy_MFCDlg, базовий
клас - CDialog
BEGIN_MESSAGE_MAP(CMy_MFCDlg, CDialog)
        ON_WM_PAINT()//відповідає OnPaint
        ON_WM_QUERYDRAGICON()//відповідає
OnQueryDragIcon
        ON_BN_CLICKED(IDOK,
&CMy_MFCDlg::OnBnClickedOk)//відповідає OnBnClickedOk
END_MESSAGE_MAP()
//обробники повідомлень класу CMy_MFCDlg
//реалізація OnInitDialog - ініціалізація діалогового
вікна
BOOL CMy_MFCDlg::OnInitDialog()
{
CDialog::OnInitDialog();//виклик функції базового
класу
SetIcon(m_hIcon, TRUE);//установка великої іконки для
діалогу
SetIcon(m_hIcon, FALSE);//установка малої іконки для
діалогу
```

```
return TRUE;  //повертає TRUE, якщо ви установлюєте
фокус в елемент керування
}
//реалізація OnPaint, викликається при необхідності
перерисування
void CMy_MFCDlg::OnPaint()
{
if (IsIconic())
{
CPaintDC dc(this); //контекст пристрою для рисування
//надіслати повідомлення WM_ICONERASEBKGND
SendMessage(WM_ICONERASEBKGND,
reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);
//центрувати іконку в клієнтському прямокутнику
int cxIcon = GetSystemMetrics(SM_CXICON);
int cyIcon = GetSystemMetrics(SM_CYICON);
CRect rect;//об'єкт, який описує прямокутник
GetClientRect(&rect);//відшукує координати
клієнтської площі вікна
int x = (rect.Width() - cxIcon + 1) / 2;
int y = (rect.Height() - cyIcon + 1) / 2;
dc.DrawIcon(x, y, m_hIcon);//рисувати іконку в
координатах x, y
}
else//інакше викликати функцію базового класу
{CDialog::OnPaint();}
}
//викликається щоб отримати курсор при перетягуванні
мінімізованого вікна
HCURSOR CMy_MFCDlg::OnQueryDragIcon()
{return static_cast<HCURSOR>(m_hIcon);}
//викликається при натиску на кнопку IDOK
void CMy_MFCDlg::OnBnClickedOk()
{edit1.SetWindowTextW(L"Hello");}//помістити в edit1
текст "Hello"
```

13 Створення додатків Win32 за допомогою Visual C++ .NET

Постановка задачі: за допомогою Microsoft Visual C++ .NET створити додаток Win32 з графічним інтерфейсом. В головному вікні програми розмістити елемент керування – поле, запрограмувати функцію-обробник події натиску на кнопку, яка виводить в поле слова "Hello World!".

Код програми

```
//файл з описом форми Form1
#pragma once//файл буде підключений тільки раз при
компіляції
namespace My_first {//простір імен My_first
//використовувати простори імен
using namespace System;//базових класів
using namespace System::ComponentModel;//для
компонентної моделі
using namespace System::Collections;//для різних
зборок, наприклад, масивів
using namespace System::Windows::Forms;//для
створення Windows додатків
using namespace System::Data;//для роботи з базами
даних
using namespace System::Drawing;//для роботи з GDI+
//клас форми Form1, успадкований від Form
public ref class Form1 : public
System::Windows::Forms::Form
{
public:
Form1(void)//конструктор
{InitializeComponent();}//виклик методу ініціалізації
protected:
~Form1()//деструктор
{if (components)
{delete components; }//вивільнити компоненти
}
private: System::Windows::Forms::Button^  button1;
//компонент button1
private: System::Windows::Forms::TextBox^  textBox1;
//компонент textBox1
//^ замінює * в новому синтаксисі для вказівників на
об’єкти в керованій купі
protected:
private:
System::ComponentModel::Container ^components;
//містить компоненти
//опис методів форми
#pragma region Windows Form Designer generated code
void InitializeComponent(void)//метод ініціалізації
форми
{//створити об’єкти button1 і textBox1
```

```
this->button1 = (gcnew
System::Windows::Forms::Button());
this->textBox1 = (gcnew
System::Windows::Forms::TextBox());
this->SuspendLayout();//тимчасово зупиняє логіку
розміщення елементу керування
//властивості об'єкту button1
this->button1->Location = System::Drawing::Point(96,
140);//координати
this->button1->Name = L"button1";//ім'я
this->button1->Size = System::Drawing::Size(88,
47);//розмір
this->button1->TabIndex = 0;//перший отримує фокус
при натиску Tab
this->button1->Text = L"button1";//надпис
this->button1->UseVisualStyleBackColor = true;//фон
рисується використовуючи стилі
//пов'язує подію з функцією обробки події
this->button1->Click += gcnew
System::EventHandler(this, &Form1::button1_Click);
//властивості об'єкту textBox1
this->textBox1->Location = System::Drawing::Point(96,
68);//координати
this->textBox1->Name = L"textBox1";//ім'я
this->textBox1->Size = System::Drawing::Size(100,
22);//розмір
this->textBox1->TabIndex = 1;//другий отримує фокус
при натиску Tab
//властивості об'єкту Form1
//установлює розміри і режим автоматичної зміни
розміру
this->AutoScaleDimensions = System::Drawing::SizeF(8,
16);
this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
this->ClientSize = System::Drawing::Size(292,
263);//розмір клієнтської області
this->Controls->Add(this->textBox1);//добавити
елемент керування textBox1
this->Controls->Add(this->button1);//добавити елемент
керування button1
this->Name = L"Form1";//ім'я
this->Text = L"Form1";//надпис
```

```
this->ResumeLayout(false);//відновлює логіку
розміщення елементу керування
this->PerformLayout();//виконує логіку розміщення
елементу керування
}
#pragma endregion
//функція обробки події
private: System::Void button1_Click(System::Object^
sender, System::EventArgs^  e) {
textBox1->Text="Hello World!";//вивести в поле
textBox1 текст
}
};
}

// My_first.cpp : головний файл проекту
#include "stdafx.h"
#include "Form1.h"//директива підключення файлу
Form1.h
using namespace My_first;//використовувати простір
імен My_first
[STAThreadAttribute]//атрибут потоковості STA
int main(array<System::String ^> ^args)//головна
функція
{//дозволяє Windows XP візуальні ефекти
Application::EnableVisualStyles();
Application::SetCompatibleTextRenderingDefault(false)
;
//створити головне вікно Form1 і виконати
Application::Run(gcnew Form1());
return 0;
}
```

14 Задачі
Задачі 1 Основи програмування мовою С++

Дано функцію f(x), аргумент x змінюється від x_{min} до x_{max} з кроком Δx (14.1). Розв'язати наступні задачі. Для вводу-виводу даних використати команди потокового вводу-виводу.

$$f(x)=\begin{cases} a\cdot tg^2(\omega+x) & 0<x<5 \\ b\cdot ctg(\varphi+x) & x\le 0 \end{cases} \qquad a=\frac{x^3+e^{x+1}}{\sqrt[3]{|x|+2}}+1$$

$$b=\log_3(x+1)$$

$$\omega=2.15 \qquad \varphi=1.05 \qquad x_{min}=-1 \qquad x_{max}=2 \qquad \Delta x=0.01 \qquad (14.1)$$

Задача 1.1 Лінійні програми. Знайти f(x) при заданому x, якщо значення x задовольняє тільки першій умові.
Задача 1.2 Розгалуження. Знайти f(x) при заданому x.
Задача 1.3 Цикли з лічильником. Протабулювати або побудувати графік функції на проміжку з заданим кроком.

Задача 1.4 Цикли. Знайти суму $\sum_{x=x_{min}}^{x_{max}} f(x)$ або добуток $\prod_{x=x_{min}}^{x_{max}} f(x)$.

Використати:
а) оператор циклу з лічильником,
б) оператор циклу з передумовою,
в) оператор циклу з післяумовою.
Задача 1.5 Цикли і розгалуження. Знайти мінімальне $f(x)_{min}$ і максимальне значення $f(x)_{max}$, на проміжку.
Задача 1.6 Цикли і розгалуження. Знайти значення функції, які більше 0 і менші 10 на проміжку.
Задача 1.7 Ітераційні цикли. Знайти корінь рівняння f(x)=0 (знайти при якому значенні x f(x)=0).
Задача 1.8 Підпрограми. Знайти f(x) при заданому x. Визначення значення функції виконати у вигляд підпрограми-функції.

Задача 1.9 Підпрограми. Знайти інтеграл $\int_{x_{\min}}^{x_{\max}} f(x)dx$. Визначення значення функції виконати у вигляд підпрограми-функції.

Задача 1.10 Одновимірні масиви. Дано масив значень аргументу $X = \begin{bmatrix} 5.5 & 0 & 3.2 & 2.5 \end{bmatrix}$. Побудувати відповідний масив значень функції f(x). Знайти в ньому кількість елементів, які менші 0 і замінити їх на 1. Використати:

а) статичні масиви,

б) динамічні масиви.

Задача 1.11 Двовимірні масиви. Дано масив значень аргументу:

$$X = \begin{bmatrix} 2.1 & 1 & 0.5 & 2 \\ 3.2 & 3.5 & 3 & 7 \\ 5.2 & 4.1 & 9.7 & 1 \\ 0 & 3.5 & 0 & 1 \end{bmatrix}.$$

Впорядкувати рядки масиву X по зростанню. Побудувати відповідний масив значень функції f(x). Використати:

а) статичні масиви,

б) динамічні масиви.

Задача 1.12 Текстові файли послідовного доступу. Записати у текстовий файл значення функції і аргументу на проміжку з заданим кроком. Виконати пошук в створеному файлі певних даних.

Задача 1.13 Структури. Створити структуру, яка містить такі члени: аргумент, функція, похідна функції $f'(x) \approx (f(x+\Delta x) - f(x))/\Delta x$. Створити масив екземплярів структури.

Задача 1.14 Бінарні файли довільного доступу. Записати у бінарний файл екземпляри структури (задача 1.13), якщо аргумент змінюється від мінімального до максимального з заданим кроком. Виконати пошук в створеному файлі певних даних.

Задача 1.15 Рядки. Створити довільний рядок (масив символів). За допомогою операторів циклу і умови замінити задані символи (або слова) в рядку на інші.

Задача 1.16 Функції для роботи з рядками. Створити довільні рядки. Обробити рядки за допомогою функцій: strlen, strcat, strcpy, srtchr, strstr, atoi, itoa, gcvt, strtod.

Задачі 2 Об'єктно-орієнтоване програмування мовою С++

Задача 2.1 Класи. Створити клас, який описує задане поняття (таблиця 14.1). Передбачити задані члени класу, у тому числі конструктор та конструктор копії. Статично та динамічно створити екземпляри класу.

Задача 2.2 Успадкування. На основі базового класу (задача 2.1) створити похідний клас, який описує задане поняття (таблиця 14.1). Передбачити задані члени класу, у тому числі конструктор та конструктор копії. Обґрунтувати специфікатори доступу членів класів. Статично та динамічно створити екземпляри базового та похідного класів.

Задача 2.3 Вкладені класи. Композиція. Створити клас, який описує задане поняття (таблиця 14.1). Передбачити задані члени класу, в тому числі поле (поля), класового типу. Статично та динамічно створити екземпляри класу.

Задача 2.4 Поліморфізм. На основі базового класу (задача 2.1) створити похідний клас, який описує задане поняття (таблиця 14.1). Передбачити задані члени класу, у тому числі віртуальні функції. Статично та динамічно створити екземпляри класу.

Задача 2.5 Шаблони класів. Створити шаблони класів, які розроблені в попередніх задачах. Створити екземпляри класів.

Таблиця 14.1 – Варіанти для задач 2

№ вар.	**Назва класу (поняття)**	**Базовий клас (поняття)**	**Члени класу (атрибути об'єктів і операції над ними)**	
			Поля (атрибути)	**Методи (операції)**
1	студент	-	прізвище, ім'я, група, курс	конструктор, друк повної інформації
	студент стаціонарної форми навчання	студент	ціна року навчання	конструктор, оплачено суму, друк повної інформації
	група	-	назва, масив студентів	конструктор, друк списку студентів
2	офісна техніка	-	тип, назва, ціна	конструктор, ціна, друк повної інформації

Продовження табл. 14.1

№ вар.	Назва класу (поняття)	Базовий клас (поняття)	Члени класу (атрибути об'єктів і операції над ними)	
			Поля (атрибути)	Методи (операції)
	комплектуючі ПК	офісна техніка	характеристики	конструктор, друк повної інформації
	персональний комп'ютер	офісна техніка	комплектуючі ПК	конструктор, сумарна ціна комплектуючих, друк повної інформації
3	арифметична операція	-	операнди a,b	конструктор, результат
	додавання	арифметична операція		конструктор, результат
	множення	арифметична операція		конструктор, результат
4	гральна карта однієї масті	-	ранг	конструктор, сума очок
	гральна карта	гральна карта однієї масті	масть, колір	конструктор, сума очок
	гравець в покер	-	масив гральних карт	конструктор, сума очок
5	шахова фігура	-	колір, тип, цінність, позиція на дошці	конструктор, масив клітинок можливих ходів
	кінь	шахова фігура	-	конструктор, масив клітинок можливих ходів
	пішак	шахова фігура	-	конструктор, масив клітинок можливих ходів, перетворення в іншу фігуру

Продовження табл. 14.1

№ вар.	Назва класу (поняття)	Базовий клас (поняття)	Члени класу (атрибути об'єктів і операції над ними)	
			Поля (атрибути)	**Методи (операції)**
	гравець	-	колір, масив фігур	конструктор, друк інформації
6	література	-	назва, видавництво, рік видання	конструктор, друк повної інформації
	книга	література	автор	конструктор, друк повної інформації
	журнал	література	номер	друк повної інформації
	читач	-	ім'я, номер читацького квитка, масив літератури	конструктор, список літератури
7	колір RGB	-	компоненти кольорів: червоного (R), зеленого (G), синього (B)	конструктор, установка кольору: R=0..255, G=0..255, B=0..255 назва: R=255, G=255, B=0 - жовтий, R=255, G=153, B=204–рожевий, R=0, G=255, B=0 - зелений
	колір Grayscale (сірий)	колір RGB		конструктор, установка кольору: R=G=B=0..255
	рисунок	-	координати точки (x,y), колір	конструктор, записати у файл, прочитати з файлу
8	точка	-	координати x, y, колір	конструктор
	коло	точка	радіус r	конструктор, довжина: $p = 2\pi r$, площа: $S = \pi r^2$

Продовження табл. 14.1

№ вар.	Назва класу (поняття)	Базовий клас (поняття)	Члени класу (атрибути об'єктів і операції над ними)	
			Поля (атрибути)	Методи (операції)
	лінія	точка	координати другої точки x2, y2	конструктор, довжина: $d = \sqrt{\Delta x^2 + \Delta y^2}$, $\Delta x = x_2 - x$, $\Delta y = y_2 - y$
	лінія (інший варіант)	-	точка 1, точка 2	конструктор, довжина: $d = \sqrt{\Delta x^2 + \Delta y^2}$, $\Delta x = x_2 - x$, $\Delta y = y_2 - y$
	прямокутник	лінія	-	конструктор, периметр: $p = 2(\Delta x + \Delta y)$, площа: $S = \Delta x \cdot \Delta y$
	трикутник	лінія	координати третьої точки x3, y3	конструктор, периметр: $p = d_1 + d_2 + d_3$, площа: $S = \sqrt{0.5p(p/2 - d_1)(p/2 - d_2)(p/2 - d$
	набір геометричних фігур	-	масиви точок, кіл, ліній, прямокутників, трикутників	конструктор, сумарна довжина ліній, сумарна площа
9	матриця	-	розмір m×n, елементи a_{ij}	конструктор, додавання матриць m×n: $c_{ij} = a_{ij} + b_{ij}$, множення на скаляр: $c_{ij} = \alpha \cdot a_{ij}$, множення матриці m×n на матрицю n×r це матриця m×r: $c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$, транспонування матриці m×n це

Продовження табл. 14.1

№ вар.	Назва класу (поняття)	Базовий клас (поняття)	Члени класу (атрибути об'єктів і операції над ними)	
			Поля (атрибути)	Методи (операції)
				матриця n×m: $c_{ji} = a_{ij}$
	квадратна матриця 3×3	матриця	-	конструктор, визначник: $D=a_{11}\cdot a_{22}\cdot a_{33}-a_{11}\cdot a_{23}\cdot a_{32}-a_{12}\cdot a_{21}\cdot a_{33}+a_{12}\cdot a_{31}\cdot a_{23}+a_{21}\cdot a_{13}\cdot a_{32}-a_{13}\cdot a_{22}\cdot a_{31}$
	вектор	матриця	{a1,a2,a3}	конструктор, довжина: $\lvert a\rvert = \sqrt{a_1^2 + a_2^2 + a_3^2}$, додавання: $c_i = a_i + b_i$, скалярне множення: $c = \sum_{i=1}^{3} a_i b_j$, векторне множення: $c = \begin{Bmatrix} a_2 \cdot b_3 - a_3 \cdot b_2 \\ a_3 \cdot b_1 - a_1 \cdot b_3 \\ a_1 \cdot b_2 - a_2 \cdot b_1 \end{Bmatrix}$
10	функція	-	змінна x, dx	конструктор, значення f(x) в точці, значення похідної $f'(x) \equiv \frac{d}{dx} f(x)$ в точці, значення інтегралу $\int_{x_{\min}}^{x_{\max}} f(x)dx$
	функція двох змінних	функція	змінна x2, dx2	конструктор, значення f(x,x2) в точці, диференціали

Закінчення табл. 14.1

№ вар.	Назва класу (поняття)	Базовий клас (поняття)	Члени класу (атрибути об'єктів і операції над ними)	
			Поля (атрибути)	Методи (операції)
				$df, \frac{\partial f}{\partial x}dx, \frac{\partial f}{\partial x_2}dx_2,$ часткові похідні $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial x_2}$
	графік функцій (таблиця значень)	-	$x_{\min}, x_{\max}$, масив функцій	конструктор, друк або запис у файл

Задачі 3 Сучасні об'єктно-орієнтовані технології

Задача 3.1 STL. За допомогою STL створити два вектори, які міститимуть значення функції і аргументу (14.1). Використовуючи компоненти STL над векторами виконати такі перетворення:

а) добавити в кінець векторів дані, якщо аргумент змінюється від мінімального до максимального з заданим кроком;

б) видалити усі елементи векторів, які задовольняють певній умові;

в) визначити розмір вектора;

г) створити третій вектор, який містить значення $f(x_i)\cdot\Delta x$;

д) знайти $\sum_{x_i=x_1}^{x_2} f(x_i)\cdot\Delta x$.

Задача 3.2 VCL. Створити додаток типу Win32 Application з використанням заданих класів і компонентів VCL (таблиця 14.2). Використати не менше 3-4 властивостей і методів кожного компоненту.

Таблиця 14.2 – Варіанти для задачі 3.2

№ вар.	Класи	Компоненти Standard	Компоненти Additional	Компоненти Win32	Компоненти System і діалогові вікна
1	AnsiString (String)	TFrame	TBitBtn TControlBar	TTabControl TPageScroller	TTimer
2	TApplication	TMainMenu TPopupMenu	TSpeedButton	TPageControl	TPaintBox

№ вар.	Класи	Компоненти Standard	Компоненти Additional	Компоненти Win32	Компоненти System і діалогові вікна
		TActionList	TApplicationEvents	TCoolBar	TCanvas TBrush TPen
3	TScreen TForm	TLabel TScrollBar	TMaskEdit	TImageList TBitmap	TMediaPlayer
4	Set	TButton TPanel	TStringGrid TLabeledEdit	TRichEdit TToolBar	TOleContainer
5	TList TObject	TRadioButton	TDrawGrid	TTrackBar TStatusBar	TDdeServerConv TDdeServerItem TDdeClientConv TDdeClientItem
6	TComponent TControl	TCheckBox	TImage TPicture	TProgressBar THeader Control	TOpenDialog
7	TWinControl TGraphicControl	TEdit	TShape TBrush TPen	TUpDown TListView TComboBox Ex	TSaveDialog
8	TStrings TStringList	TMemo	TScrollBox TValueList Editor	THotKey TTreeView	TPaintBox TCanvas TBrush TPen
9	TStream TMemoryStream	TListBox	TCheckList Box TChart	TAnimate TMonthCalendar TDate	TTimer
1(	TThread	TComboBox	TSplitter TColorBox TColor	TDateTime Picker TDateTime	TMediaPlayer

Задача 3.3 VCL. За допомогою Borland C++ Builder та VCL розробити додаток Win32 з графічним інтерфейсом для задач 1.

Задача 3.4 VCL. За допомогою Borland C++ Builder та VCL розробити додаток Win32 з графічним інтерфейсом для задач 2.

Задача 3.5 Створення компонентів VCL. Розробити новий компонент VCL. Добавити нові властивості, методи і події. В якості базового компоненту вибрати компонент з таблиці 14.2.

Задача 3.6 DLL. Створити бібліотеку DLL, яка містить клас (див. табл. 14.1). Створити додаток, який використовує DLL.
Задача 3.7 COM. Створити компонент-сервер COM на основі класу (див. табл. 14.1). Створити клієнта COM.
Задача 3.8 ActiveX. Створити компонент ActiveX на основі розробленого компоненту VCL (див. задачу 3.5). Створити клієнта, який використовує цей компонент.
Задача 3.9 ActiveX. Створити форму ActiveX на основі розробленого додатку у C++ Builder (див. задачу 3.2). Створити клієнта, який використовує цю форму.
Задача 3.10 CORBA. Створити компонент-сервер CORBA на основі класу (див. табл. 14.1). Створити клієнта CORBA.
Задача 3.11 MFC. За допомогою Microsoft Visual C++ та MFC розробити додаток Win32 з графічним інтерфейсом для задач 2.
Задача 3.12 .NET. За допомогою Microsoft Visual C++ та .NET розробити додаток Win32 з графічним інтерфейсом для задач 2.

Перелік рекомендованих джерел

Мова програмування С++ для початківців

1 Глинський Я.М., Анохин В.Є., Ряжська В.А. С++ і C++ Builder. – Львів: Деол, СПД Глинський, 2003. – 192 с., іл.
2 Дэвис, Стефан, Р. С++ для "чайников", 4-е издание. : Пер. с англ. : — М. : Издательский дом '"Вильямс", 2003. — 336 с. : ил. : Парал. тит. англ.
3 Шилдт Г. Самоучитель С++: Пер. с англ. — 3-е изд. — СПб.: БХВ-Петербург, 2005. — 688 с.
4 Стенли Б. Липпман. С++ для начинающих: Пер. с англ. 2тт. - Москва: Унитех; Рязань: Гэлион, 1992, 304-345сс.
5 В.В. Подбельский. Язык С++: Учебное пособие. - Москва: Финансы и статистика, 1995. - 560с.
6 К. Джамса. Учимся программировать на языке С++: Пер. с англ. - Москва: Мир, 1997. - 320с.
7 Дж. Либерти. Освой самостоятельно С++ за 21 день. Пер. с англ. - Москва: Вильямс, 2001. - 832 с.: ил.
8 Прата Стивен. Язык программирования С++. Лекции и упражнения. Учебник: Пер. с англ./Стивен Прата – СПб.: ООО "ДиаСофтЮП", 2005. – 1104 с.
9 Язык С++: Учеб. пособие/И.Ф.Астахова, С.В.Власов, В.В.Фертиков, А.В.Ларин. – Мн.: Новое знание, 2003. – 203 с.
10 Лаптев В. В. С++. Экспресс-курс. - СПб.: БХВ-Петербург, 2004. - 512 с.: ил.
11 Франка П. С++: учебный курс. — СПб.: Питер, 2003. — 521 с.: ил.
12 Боднарев В.М. Программирование на С++. 2-е изд. – Харьков: "Компания СМИТ", 2005. – 284 с.
13 Х. Дейтел, П. Дейтел. Как программировать на С++: Пер. с англ. - Москва: ЗАО "Издательство БИНОМ", 1998. - 1024с.
14 Программирование на С++: Уч. пос./ Аверкин В.П. и др.- СПб.: КОРОНА принт, 1999.- 256 с.

Об’єктно-орієнтоване програмування

15 Бадд Т. Объектно-ориентированное программирование в действии /Пер. с англ.– СПб.: Питер, 1997.

16 Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. – СПб.: "Невский Диалект", 2001. – 560 с.
17 Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб: Питер, 2001. — 368 с.: ил.
18 Бєгун А.В. Технологія програмування: об'єктно-орієнтований підхід: Навч.-метод. посібник для самост. вивч. дисц. – К.: КНЕУ, 2000. – 200 с.
19 Мухортов В. В., Рылов В.Ю. Объектно-ориентированное программирование, анализ и дизайн. Методическое пособие. Новосибирск, 2002. – 108с.
20 Гайсарян С.С. Объектно-ориентированные технологии проектирования прикладных программных систем. - М.: ЦИТ, 2001.
21 М.Бен-Ари. Языки программирования. Практический сравнительный анализ.: Пер. с англ.-М.:Мир, 2000.-366 с.

Мова програмування C++

22 Клюшин Д.А. Полный курс C++. Профессиональная работа. – М.: Издательский дом «Вильямс», 2004. – 672 с.: ил.
23 Страуструп Б. Язык программирования C++. –М.: И.В.К.-СОФТ, 1991.
24 М. Эллис, Б. Строуструп. Справочное руководство по языку C++ с комментариями: Пер. с англ. - Москва: Мир, 1992. - 445с.
25 Страуструп Б. Дизайн и эволюция C++: Пер. с англ. – М.:ДМК Пресс; СПб.: Питер, 2006.- 448 с.: ил.
26 Б.Керниган, Д.Ритчи. Язык программирования Си.\Пер. с англ., 3-е изд., испр. СПб.: «Невский Диалект», 2001. – 352 с.:ил.
27 Глушаков С.В., Коваль А.В., Смирнов С.В. Язык программирования C++. – Х.: Фолио, 2002.
28 С.В.Глушаков, А.В.Коваль, С.В.Смирнов. Язык программирования C++. Учебный курс. – Харьков: Фолио, 2001. - 505 с.
29 Дьюхарст С., Старк К. Программирование на C++. – К.: ДиаСофт, 1993. - 272с.
30 Кораблев В. С и C++. – К.:BHV, 2002.
31 Шилдт, Герберт. Полный справочник по C++, 4-е издание.: Пер. с англ. – М.: Издательский дом "Вильямс", 2007. – 800с.: ил.
32 Бруно Бабэ. Просто и ясно о Borland C++: Пер. с англ. - Москва: БИНОМ, 1994. -400с.

33 Ирэ Пол. Объектно-ориентированное программирование с использованием C++: Пер. с англ. - Киев: НИИПФ ДиаСофт Лтд, 1995. - 480с.
34 Т. Фейсон. Объектно-ориентированное программирование на Borland C++ 4.5: Пер. с англ. - Киев: Диалектика, 1996. - 544с.
35 Т. Сван. Освоение Borland C++ 4.5: Пер. с англ. - Киев: Диалектика, 1996. - 544с.
36 У. Сэвитч. C++ в примерах: Пер. с англ. - Москва: ЭКОМ, 1997. - 736с.
37 В.А. Скляров. Язык C++ и объектно-ориентированное программирование: Справочное издание. - Минск: Вышэйшая школа, 1997. - 480с.
38 Р.Лафоре. Объектно-ориентированное программирование в C++. - СПб.: Питер, 2004. - 922с.
39 Ален И. Голуб. Веревка достаточной длины, чтобы... выстрелить себе в ногу. Правила программирования на Си и Си++. – Москва, 2001.
40 Александреску, Андрей. Современное проектирование на C++. Серия C++ In-Depth, т. 3.: Пер. с англ. — М. : Издательский дом "Вильямс", 2002. — 336 с. : ил.
41 Джесс Либерти. C++. Энциклопедия пользователя. - К.: ДиаСофт, 2001. – 590 с.
42 Р. Вайнер, Л. Пинсон. C++ изнутри. - Киев: НПИФ "ДиаСофт", 1993.
43 Фридман А., Кландер Л., Михаэлис М., Шильдт Х. C/C++. Архив программ – М.: ЗАО "Издательство БИНОМ", 2001 г. – 640 с.: ил.
44 Мейерс С. Эффективное использование C++. 50 рекомендаций по улучшению ваших программ и проектов: Пер. с англ. - М.: ДМК Пресс; Спб.: Питер, 2006. - 240 с: ил.
45 Мэйерс С. Эффективное использование C++. 55 верных способов улучшить структуру и код ваших программ - М.: ДМК Пресс, 2006. - 300 с: ил.
46 Кёниг, Эндрю, Му, Барбара, Э. Эффективное программирование на C++. Серия C++ In-Depth, т. 2. : Пер. с англ. — М. : Издательский дом "Вильямс", 2002. — 384 с. : ил.
47 Эккель Б., Эллисон Ч. Философия C++. Практическое программирование. – СПб.: Питер, 2004. – 608 с.: ил.
48 Эккель Б., Эллисон Ч. Философия C++. Введение в стандартный C++. 2-е изд. – СПб.: Питер, 2004. – 572 с.: ил.
49 Седжвик Роберт. Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск: Пер. с англ./Роберт Седжвик. - К.: Издательство «ДиаСофт», 2001.- 688 с.

50 Седжвик Роберт. Фундаментальные алгоритмы на C++. Алгоритмы на графах: Пер. с англ./Роберт Седжвик. - СПб.: ООО «ДиаСофтЮП», 2002.- 496 с.
51 Хэзфилд Ричард, Кирби Лоуренс и др. Искусство программирования на С. фундаментальные алгоритмы, структуры данных и примеры приложений. Энциклопедия программиста: Пер. с англ./ Ричард Хэзфилд, Лоуренс Кирби и др. – К.: Издательство "ДиаСофт", 2001. – 736 с.
52 Бентли Дж. Жемчужины программирования. 2-е издание. — СПб.: Питер, 2002. — 272 с: ил.
53 Крис Х. Паппас, Уильям Х. Мюррей III. Отладка в C++. Руководство для разработчиков. - М.: Бином, 2001. – 512 с.
54 Уоррен, Генри, С. Алгоритмические трюки для программистов.: Пер. с англ. – М.: Издательский дом "Вильямс", 2003.- 288с.: ил.
55 Саттер, Герб. Решение сложных задач на C++. Серия C++In-Depth, т.4.:Пер. с англ. – М.: Издательский дом "Вильямс", 2002. – 400 с.: ил.
56 Саттер, Герб. Новые сложные задачи на C++. : Пер. с англ. — М. : Издательский дом "Вильямс", 2005. — 272 с. : ил.
57 Джефф Элджер. C++. Библиотека программиста. – СПб.: Питер, 2001, 320с.
58 Draft International Standard ISO/IEC JTC 1/SC22/WG21 N2135:2006-11-06, Programming Languages — C++.
59 Саттер, Герб, Александреску, Андрей. Стандарты программирования на C++. Пер. с англ. – М. Издательский дом "Вильямс", 2005. – 224 с.: ил.
60 Сабуров С.В. Языки программирования С и C++. - М.: Бук пресс, 2006. - 647 с. (Справочное руководство пользователя персонального компьютера).
61 Громов Ю.Ю., С.И.Татаренко. Языки СИ и C++ для решения инженерных и экономических задач. Учебное пособие. – Тамбов: издательство ТГТУ, 2001. – 150 с.

Стандартна бібліотека C++

62 Александр Степанов, Менг Ли. Руководство по стандартной библиотеке шаблонов (STL). – Москва, 1999.
63 Мейерс С. Эффективное использование STL. Библиотека программиста. – СПб.: Питер, 2002.- 224 с.: ил.
64 Леен Аммерааль. STL для программистов на C++. Пер. с. англ. / Леен Аммерааль – М.:ДМК, 1999 – 240 с., ил.

65 C++ Стандартная библиотека. Для профессионалов/ Н.Джосьютис. – СП Питер, 2004. – 730 с.: ил.

Програмування на Borland C++ Builder

66 Borland C++ Builder 6. Для профессионалов / В.А.Шамис. – СПб.:Питер, 2004. – 798 с.: ил.
67 Холингворт Джаррод и др. Borland C++ Builder 6. Руководство разработчика. Пер. с англ. – М.: Издательский дом «Вильямс», 2003. – 976 с.: ил.
68 Баттерфилд Дэн, Боб Сворт, Холингвэрт Дж. C++ Builder 5: Руководство разработчика: Т. 1: Основы / пер. с англ., М.: Вильямс, 2001.
69 Баттерфилд Дэн, Боб Сворт, Холингвэрт Дж.C++ Builder 5: Руководство разработчика: Т. 2: Сложные вопросы программирования / пер. с англ., М.: Вильямс, 2001.
70 А.Я.Архангельский, М.А.Тагин Программирование в C++ Builder 6 и 2006.—М.:ООО "Бином-Пресс", 2007г.—1184 с.:ил.
71 А.Я.Архангельский Приемы программирования в C++ Builder 6 и 2006.—М.:ООО "Бином-Пресс", 2006г.—992 с.:ил.
72 Архангельский А.Я. C++Builder 6. Справочное пособие. Книга 1. Язык C++.-- М.: Бином-Пресс, 2002 г. – 544 с.:ил.
73 Архангельский А.Я. C++Builder 6. Справочное пособие. Книга 2. Классы и компоненты.-- М.: Бином-Пресс, 2002. – 528 с.:ил.
74 Кент Рейсдорф, Кен Хендерсон. Borland C++ Builder. Освой самостоятельно за 21 день. – М.: Вильямс, 2003. - 700 с.
75 М. Л. Подкур, П. Н. Подкур, Н. К. Смоленцев. Программирование в среде Borland C++ Builder с математическими библиотеками MATLAB C/C++ . – М.: ДМК Пресс, 2006. - 496с.
76 С. Бобровский. Самоучитель программирования на языке C++ в системе Borland C++Builder 5.0. – М.:ДЕСС КОМ, 2001. -272с.
77 Хомоненко А.Д., Ададуров С.Е. Работа с базами данных в C++ Builder. — СПб.: БХВ-Петербург, 2006.—496 с: ил.
78 Послед Б.С. Borland C++ Builder 6. Разработка приложений баз данных — СПб.: ООО "ДиаСофтЮП", 2003 — 320 с.
79 Культин Н.Б. C++ Builder в задачах и примерах. - СПб.: БХВ-Петербург, 2005. — 336 с: ил.
80 Herbert Schildt, Greg Guntle. Borland® C++ Builder™: The Complete Reference. - Osborne/McGraw-Hill, 2001, -1009pp.
81 В.М.Баканов. Разработка WINDOWS - интерфейса прикладных программ с помощью интегрированных сред DELPHI / C++Builder. -М.: МГАПИ, 1998, -69 С.

82 В.М.Баканов. Разработка прикладных программ для ОС WINDOWS с помощью интегрированных сред Delphi / C++Builder. – М.: МГАПИ, 2000, -84с.
83 Вячеслав Ермолаев, Тарас Сорока. C++ Builder: Книга рецептов. - М.: КУДИЦ-ОБРАЗ, 2006. - 208 с.

Технологія COM

84 Макаревич Л. Г.ActiveX с нуля. Практическое руководство. Для студентов 3-го курса факультета АВТФ направления 542200 Информатика и вычислительная техника.
85 Бокс Д. Сущность технологии COM. Библиотека программиста. — СПб.: Питер, 2001. — 400 с.: ил.
86 Роджерсон Дейл. Основы COM. - М.: Издательство: Русская Редакция, 2000. - 400 с.

Програмування на Visual C++ MFC та .NET

87 К. Паппас, У.Мюррей. Эффективная работа: Visual C++ .NET. – СПб.:Питер, 2002. – 816 с.: ил.
88 Хаймен Майкл, Арнсон Боб. Visual C++ .NET для «чайников». – Пер. с англ. – М.: Издательский дом «Вильямс», 2002. – 288 с.: ил.
89 Оберг, Роберт, Дж., Торстейнсон, Питер.Архитектура .NET и программирование с помощью Visual C++.: Пер. с англ. —М.: Издательский дом "Вильяме", 2002. — 656 с.: ил. — Парад, тит. англ.
90 А.Мешков, Ю.Тихомиров. Visual C++ и MFC. Программирование для Windows NT и Windows 95. В 3-х томах. СПб.: "BHV-СПб", 1999.
91 Ю.Олафсен, К.Скрайбнер, К.Д. Уайт и др. VISUAL C++ 6 и MFC. Энциклопедия пользователя. - К.: ДиаСофт, 2000, - 716 с.
92 Секунов Н.Ю. Самоучитель Visual C++.NET. - СПб.: БХВ-Петербург, 2002.—736 с: ил.
93 Шеферд Джордж. Программирование на Microsoft Visual C++ .NET /Пер. с англ. 2-е изд. — М.: Издательско-торговый дом "Русская Редакция"; СПб.: Питер, 2005. — 928 стр.: ил.
94 Круглински Д., Уингоу С, Шеферд Дж. Программирование на Microsoft Visual C++ 6.0 для профессионалов/Пер, с англ. -СПб: Питер; М.: Издательско-торговый дом "Русская Редакция", 2004. — 861 с.: ил.
95 С.Гилберт, Б.Маккарти. Самоучитель Visual C++ 6 в примерах. Пер. с англ. –Диасофт, 2003, 496с.

96 Том Арчер, Эндрю Уайтчепел. Visual C++ .NET. Библия пользователя.: Пер. с англ. – М.: Издательский дом "Вильямс", 2005. – 1216 с.: ил.
97 Аравинд Корера, Стивен Фрейзер, Сэм Джентайл, Нираджан Кумар, Скотт Маклин, Саймон Робинсон, д-р П.Г. Саранг. Visual C++ .NET: Пособие для разработчиков C++. - М.: Издательство – "ЛОРИ", 2003.

Програмування на C# .NET

98 Лабор В. В. Си Шарп: Создание приложений для Windows / В. В. Лабор.— Мн.: Харвест, 2003. - 384 с.
99 Шилдт, Герберт. Полный справочник по C#.: Пер. с англ. – М.: Издательский дом "Вильямс", 2004. – 752с.: ил.
100 Ватсон Карли и др. C#.: Пер. с англ. – М.: Лори, 2005. – 862с.
101 Шилдт Г. C#: учебный курс. – СПб.: Питер; К.: BHV, 2003. - 512 с.: ил.
102 Троелсен. Э. C# и платформа .NET. Библиотека программиста. — СПб.: Питер, 2004. —796 с.: ил.
103 Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 2.0 на языке C#. Мастер-класс. / Пер. с англ. — М. : Издательство «Русская Редакция» ; СПб. : Питер , 2007. — 656 стр. : ил.
104 Microsoft Corporation: Брайан Джонсон, Крэйг Скибо, Марк Янг. Основы Microsoft Visual Studio .NET 2003 /Пер. с англ. - М.: Издательско-торговый дом "Русская Редакция", 2003. — 464 стр.: ил.
105 Просиз Дж. Программирование для Microsoft .NET /Пер. с англ. — М.: Издательско-торговый дом "Русская Редакция", 2003. — 704 стр.: ил.
106 Климов А.П. Программирование КПК и смартфонов на .NET Compact Framework. –СПб.: Питер, 2007. – 320 с.: ил.
107 Рихтер Дж. Программирование на платформе Microsoft .NET Framework /Пер. с англ. — 2-е изд., испр. — М.: Издательско-торговый дом «Русская Редакция», 2003- — 512 стр.: ил.
108 Microsoft Corporation. Разработка Windows-приложений на Microsoft Visual Basic .NET и Microsoft Visual C# .NET. Учебный курс MCAD/MCSD/Пер. с англ. — М.: Издательско-торговый дом «Русская Редакция», 2003. — 512 стр.: ил.
109 Платт Д. С. Знакомство с Microsoft .NET/Пер. с англ. — М.: Издательско-торговый дом Русская Редакция, 2001. — 240 с.: ил.

110 Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 1./Пер. с англ. — М.: Издательско-торговый дом "Русская Редакция", 2002.- 576 с.: ил.
111 Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 2./Пер. с англ. — М,: Издательско-торговый дом "Русская Редакция", 2002.— 624 с.: ил.
112 Либерти, Джесс. Программирование на C#. – СПб.: "Символ-Плюс", 2003. - 684с.