**Лабораторна робота №5.** Складання програм з використанням найпростіших класів та об’єктів.

**Мета:** Набуття навичок в розробці найпростіших класів та роботі з об’єктами класів.

**Порядок виконання роботи**

1. Ознайомитися з теоретичними основами та принципами С++ як мови ООП, поняттями інкапсуляція та приховування інформації, синтаксисом оголошення класу, встановленням специфікаторів доступу.
2. Зберегти результати попередніх ЛР для використання відпрацьованих алгоритмів в подальших ЛР.
3. Створити клас *Person, який вміщує* інформацію про деяку людину. Тіло класу включає дані (члени-дані), функції (члени-функції) і специфікатори доступу до членів класу. Клас *Person* повинен вміщувати наступні дані: код особи, прізвище, ім’я та по–батькові; рік народження; телефон; стать (1 – жіноча, 2 чоловіча). На цьому етапі створити один екземпляр класу *Person.* Створити методи-члени класу для операцій: заповнення членів-даних значеннями по замовчуванню (для початкового тестування), консольне введення і виведення значень екземплярів класу *Person*, причому код особи задавати програмно.

Винесіть оголошення класу (інтерфейсу) у файл заголовка, який зазвичай має розширення «.h». Реалізацію (визначення функцій-членів) запишіть у файл з розширенням «.cpp».

1. В головній програмі створити меню, аналогічне розробленим в попередніх ЛР з позиціями:

"interactive data entry about the person" (інтерактивне введення даних про особу) (1),

"interactive data entry about the teacher" (інтерактивне введення даних про викладача) (2),

«interactive data entry about the student» (інтерактивне введення даних про студента) (3),

"reading from a data file about persons " (читання з файлу даних про осіб) (4),

"reading from the teacher data file " (читання з файлу даних про викладачів) (5),

"reading from a data file about students " (читання з файлу даних про студентів) (6),

«calculation of the average score» (розрахунок середнього балу) (7),

"sorting teacher data by phone number" (сортування за номером телефону даних про викладачів) (8),

"sorting student data by phone number" (сортування за номером телефону даних про студентів) (9).

"***Exit***" ("Вихід").

По незадіяним позиціям меню надати повідомлення "Not designed".

Встановити перевірки щодо виклику позицій меню, наприклад, виконання позицій щодо обчислення середніх балів та сортування інформації по студентах виконуються при виконанні позицій щодо введення відповідних даних. При невиконанні умов надати підказку щодо необхідності введення даних.

1. Розробити тест для перевірки правильності роботи даної програми та виконати її.
2. В першому рядку програми та заголовкового файлу записати в коментарі номер групи та прізвище, а також номер ЛР.
3. Результати надсилати на електронну адресу викладача

[**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)у вигляді заголовкового файлу та cpp-файлу з іменем у форматі

**<Номер групи> <Номер лабораторної><Прізвище англійською>**

Наприклад, 31-01Ivanov.cpp.

Тему в заголовку листа записати

**ООП<Номер групи>-ЛР <Номер лабораторної>-<Прізвище>**

**Строк відсилки ЛР ІПЗ-31 08.10.2021**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, тему в заголовку листа записати

**ООП<Номер групи>-Запитання-<Прізвище >**.

**Вимоги до програм**

Програми здаються у вигляді вихідних текстів програми (\*.cpp, \*.h) з іменами у форматі **<Номер групи><Номер лабораторної>[-<Номер завдання>]<Прізвищеанглійською>**. Наприклад,3102-1buts.cpp.

Програмне забезпечення лабораторної роботи повинно бути основане на принципах об’єктно-орієнтованого та структурного програмування. Програма повинна включати:

* опис структури кожного класу, яка містить блок опису даних класу; конструктори, деструктори, функції введення даних класу, функції виведення результатів розв’язання задачі та функції обробки даних класу,
* опис всіх функцій кожного класів;
* функцію main(), в якої описані декілька об’єктів розроблених класу та програмний код використання цих об’єктів для розв’язання поставленої задачі з доцільними коментарями.

**Контрольні запитання**

* 1. В чому полягають основні принципи ООП?
  2. Що таке об’єкт та клас?
  3. Чим відрізняються поняття класу та структури у С++? Дайте загальне визначення класу?
  4. Структура класу в С++? Формат типу class в С++.
  5. В чому сутність принципу інкапсуляції?
  6. Для чого використовують інкапсуляцію?
  7. Які ви знаєте специфікатори доступу.
  8. Як описуються функції-члени класу?

**Теоретичні відомості**

Об'єктно-орієнтований підхід використовує наступні базові поняття:

- ·об'єкт;

- ·властивість об'єкта;

- ·метод обробки;

- ·подія ;

- ·клас об'єктів.

Розглянемо кожне з цих понять.

**Об'єкт** це сукупність властивостей (параметрів) визначених сутностей і методів їх обробки (програмних засобів). Об'єкт містить інструкції (програмний код), що визначають дії, які може виконувати об'єкт, та оброблювані дані.

**Властивість** - характеристика об'єкта, його параметр. Всі об'єкти наділені певними властивостями, що у сукупності виділяють об'єкт із множини інших об'єктів. Об'єкт має якісну визначеність, що дозволяє виділити його з множини інших об'єктів і обумовлює незалежність створення й обробки від інших об'єктів.

Одним із властивостей об'єкта є метод його обробки. **Метод** - програма дій над об'єктом чи його властивостями. Метод розглядається як програмний код, пов'язаний з певним об'єктом; здійснює перетворення властивостей, змінює поведінку об'єкта. Об'єкт може мати набір заздалегідь визначених вбудованих методів обробки, або створених користувачем чи запозичених у стандартних бібліотеках, що виконуються при настанні заздалегідь визначених подій, наприклад, введення даних, зміна значень властивостей і т.п.

**Подія** - зміна стану об'єкта. Зовнішні події генеруються користувачем (наприклад, , запуск макросу ); внутрішні події генеруються системою.

Об'єкти можуть поєднуватися в класи (групи чи набори - у різних програмних системах можлива інша термінологія).

**Клас** - це структурований користувацький тип, що поєднує дані і функції, що їх перетворюють, в єдине ціле. Механізм класів дозволяє створювати типи в повної відповідності з принципами абстракції даних, тобто клас задає певну структурну сукупність типизованих даних та дозволяє визначити набір операцій над цими даними.

Формат класу в алгоритмічної мові С++ має наступний вигляд:

**сlass ім’я\_класу { список\_компонентів класу};**

де сlass – службове слово мови С++, ім’я\_класу – вільно обраний програмістом ідентифікатор, список\_компонентів – опис та визначення типізованих даних та функцій класу.

Компонентами класу можуть бути дані, функції, класи, бітові поля, ім’я типів. Список компонентів в фігурних дужках називається тілом класу. Визначення класу в С++ завжди закінчується символом “;”. Функції, що належать класу, називають функції-члени класу (member functions), а дані, що належать класу, називають дані-члени класу (data members).

В програмуванні клас визначає множину об’єктів, які об’єднаються за однаковими властивостями (єдиною групою даних) та сукупністю однакових функцій. Для опису об’єкту класу в С++ використовують наступний формат:

**ім’я\_класу ім’я\_об’єкту;**

Як тільки об’єкт класу визначається в програмі, виникає можливість звертатися до його компонент (або даних, або функцій) за допомогою оператора доступу, формат якого представлений нижче:

**ім’я\_об’єкту.ім’я\_даного;** або

**ім’я\_об’єкту.ім’я\_функції();**

або через вказівник на об’єкт класу формат має вигляд:

**вказівник\_на\_об’єкт\_класу -> ім’я\_даного;**

**вказівник\_на\_об’єкт\_класу->і’м’я\_функції();**

Вказівник на об’єкт класу дозволяє викликати функції-члени класу для обробки даних того об’єкту, який адресується вказівником. Так, наприклад, викликати функцію-член Display() класу Dsp можна оператором виду

Dsp D;

\*DPtr = &D;

&DRef= D;

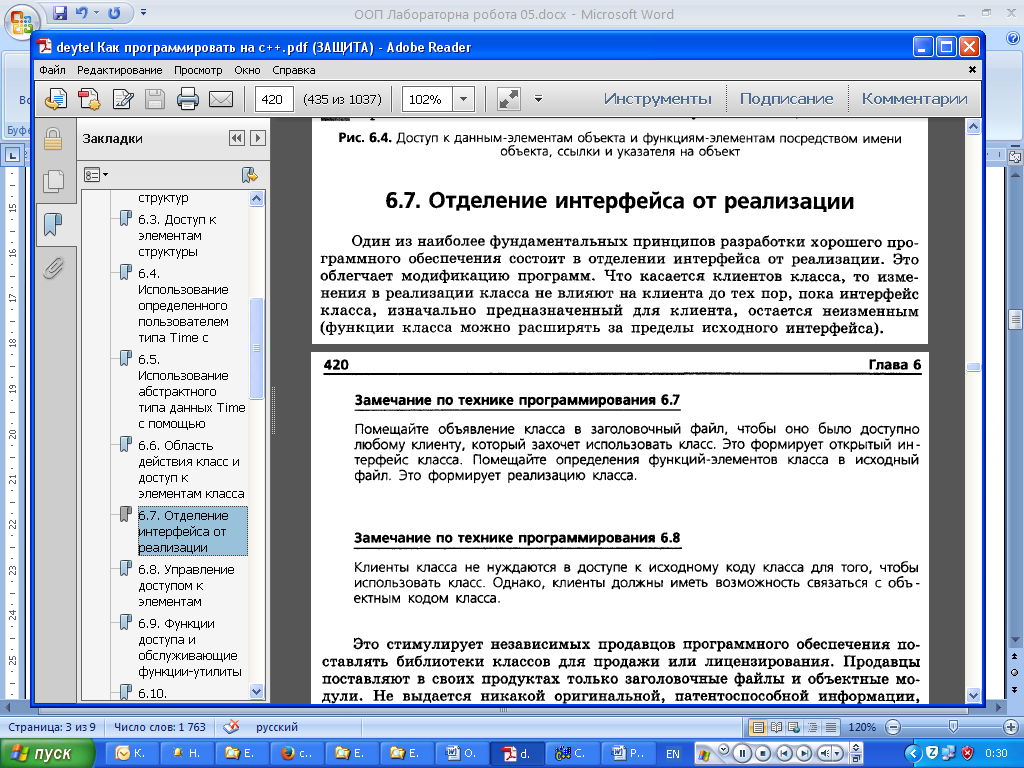
D.Display();

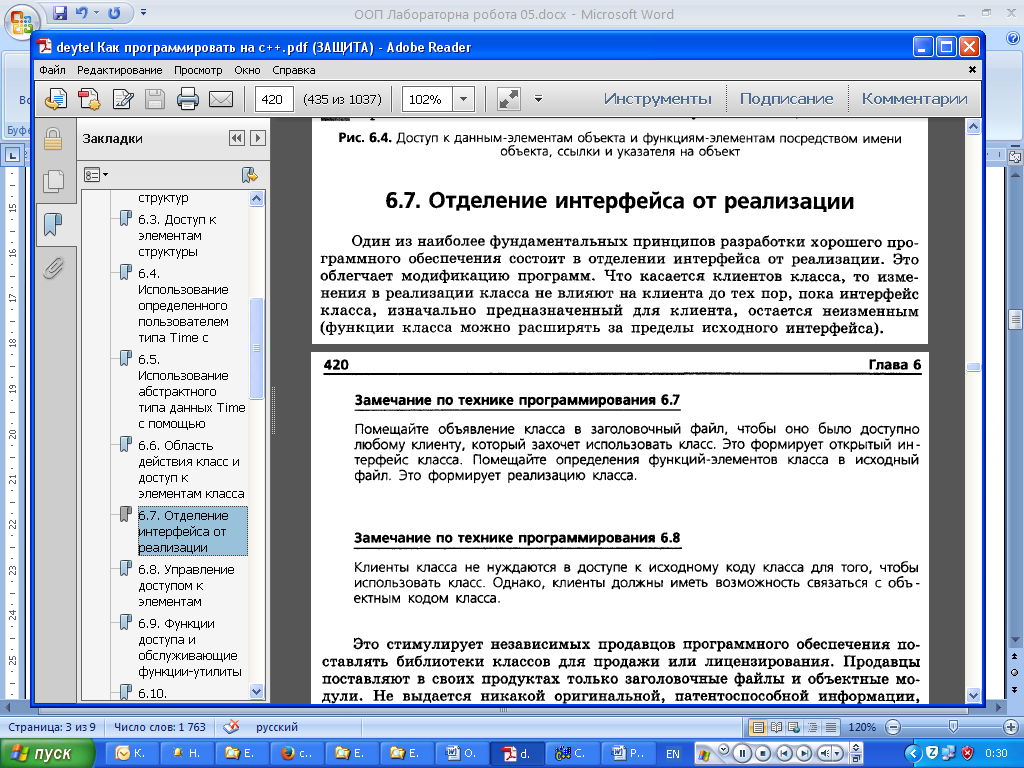
DPtr -> Display();

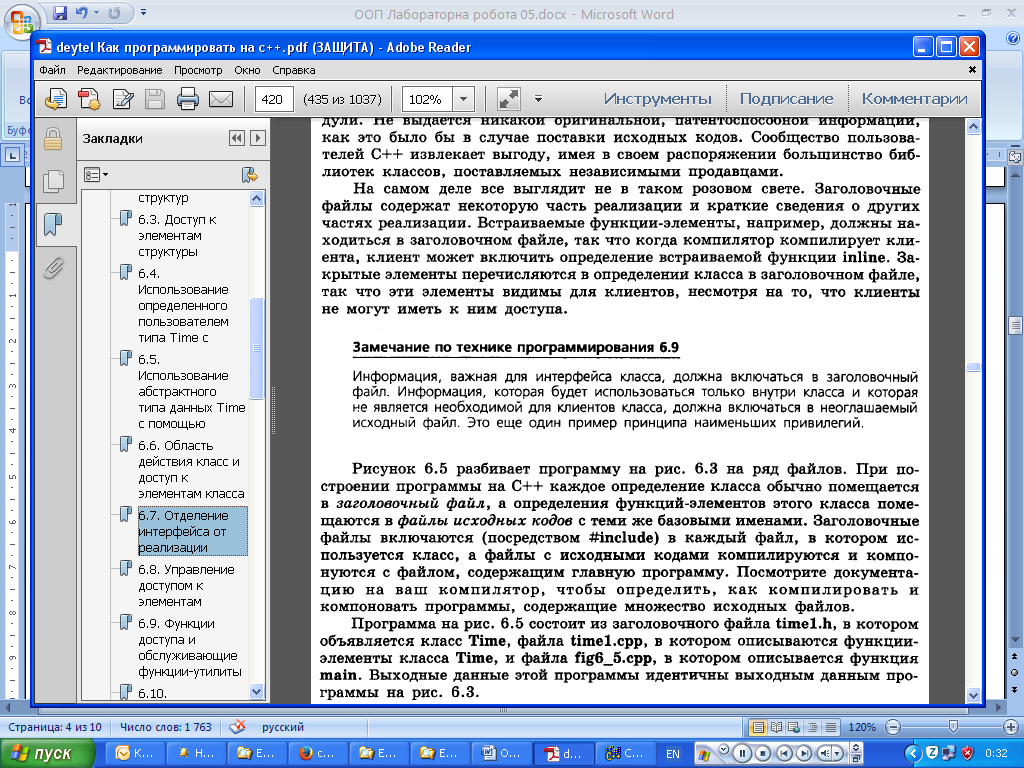
DRef.Display();

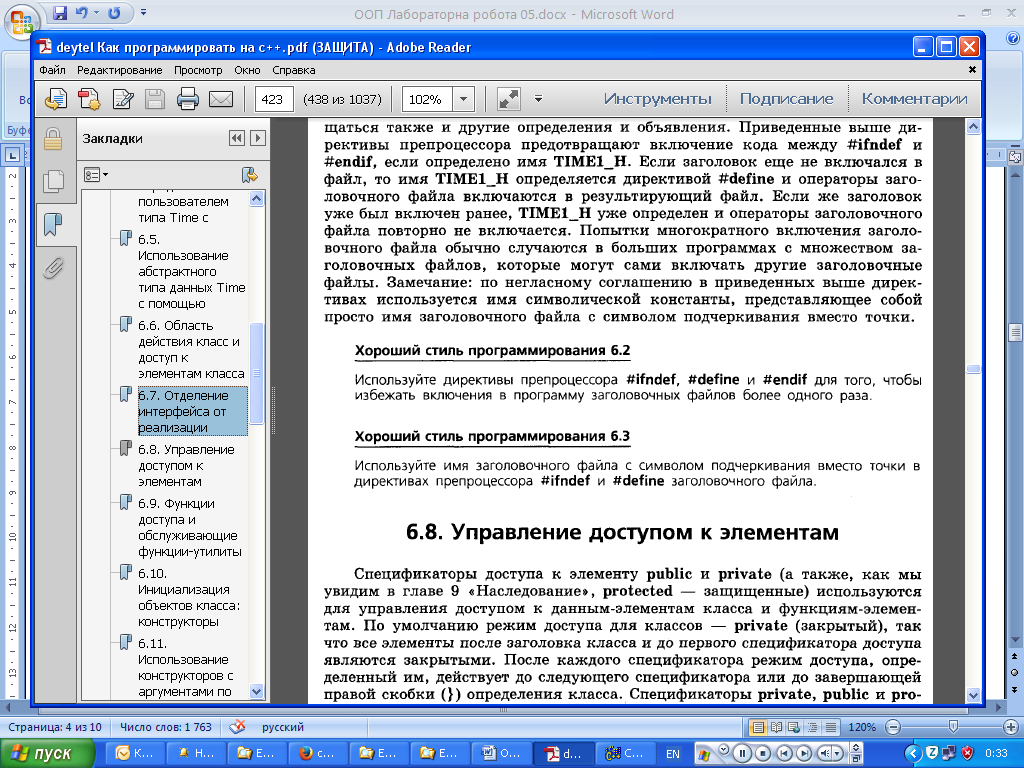
**Відділення інтерфейсу від реалізації**

З книги Харві Дейтела "Как программировать на С++"





****



Директива **#define** приймає дві форми:

* визначення констант;
* визначення макросів.

Директива **#ifndef** перевіряє, чи був раніше визначений макрос або символічна константа як #define. Якщо – так, компілятор включає в програму код, який знаходиться між директивами #else і #endif, або, якщо немає директиви #else, компілятор відразу переходить до #endif. Директива #ifndef може бути використана для підключення заголовних файлів. якщо вони не підключені, для цього використовувати символічну константу, як індикатор підключеного до проекту функціоналу. Наприклад, в заголовному файлі є інтерфейс класу, який необхідно підключити до проекту, якщо раніше цей клас не був підключений.

#ifndef PRODUCT\_H

#define PRODUCT\_H

class Product

{

// код класса...

};

#endif  PRODUCT\_H

У цьому випадку використовується порожня символьна константа PRODUCT\_H, яка може бути визначена в програмі тільки разом з класом Product. Тому, якщо ми виявимо, що константа PRODUCT\_H вже визначена, значить клас теж і тоді ми виключимо повторне визначення класу, яке може привести до помилки перевизначення.

**Приклад**. Розробимо структуру класу CTіme, що містить дані-члени класу: year, month, day, hour, minute і функції-члени класу для їхнього введення, виведення та обробки. Структура класу наведена нижче.

classCTime{

Public: *//* специфікатор доступу до членів класу зі зовнішнього середовища

int year; // дані – члени класу

int month;

int day;

int hour;

int minute;

void Display (void);*//* функція-член класу}

Опис класу схожий на опис структури. Специфікатор доступу publіc - контролює можливість використання членів класу в зовнішніх програмах, і відкриває доступ до всіх членів класу, що знаходяться за ним, для всіх користувачів класу, тому такі члени класу називаються **відкритими.**

Дані-члени класу можуть бути: змінними, покажчиками, посиланнями, масивами, структурами, об'єктами класу, і т.д. Структура класу відображує перший принцип об’єктно-орієнтованого програмування – інкапсуляцію. Інкапсуляція означає сполучення даних із методами їх обробки в абстрактних типах даних - класах об'єктів ( рисунок 1.1). Тому функції-члени класу описуються в тілі класу прототипами функцій і призначені для виконання певних операції над даними-членами класу.



Рисунок 1.1 - Графічна інтерпретація властивості класу - інкапсуляції

Нижче наведений приклад використання об’єктів класу в функції main ().

**Приклад**

# include < iostream.h >

# include < stdio.h >

class CTime {

public: *//* специфікатор

int year;

int month;

 int day;

 int hour;

 int minute;

 void Display ( void ); // функція-член

 };

int main ()

{

**CTime** object1; *//* об’єкт типу *CTime*; екземпляр типу класа

object.month=7; // ініціалізація даних-членів об’єкту;

object.day=14;

object.year=2003; ініціалізація даних об’єкту

object.hour=8;

object.minute=30;

object.Display (); // виклик функції об’єкту

cout << ′ \ n The end ′;

return 0;  *}*

//опис функції-члена класу

void CTime::Display ( void )

{ char s [32];

sprintf(s,“Data:%02d/%02d%/%04dTime:%02d:%02d\n”=,

month, day, year, hour, minute );

cout << s ;}

У даному прикладі клас CTіme - усього лише шаблон ( схема ), що описує формат членів класу, та для роботи з ним в функції main()створений об'єкт цього класу object. При ініціалізації даних-членів класу для доступу до даних об'єкту класу використовується оператор крапки (object.day).

Якщо в програмі використовують декілька об’єктів одного класу CTіmе:

CTіme today;

CTіmе tomorrow;

CTіmе yesterday;

то для цих об'єктів функцію Dіsplay( ) можна викликати таким чином: today.Dіsplay(); - виклик функції об'єктуtoday;

tomorrow.Dіsplay();  ***-*** виклик функції об'єктуtomorrow;

yesterday.Dіsplay(); *-* виклик функції об'єктуyesterday*;*

**Організація доступу до даних-членів класу**. **Специфікатори доступу**

В одному класі можуть бути дані-члени, для деяких з них доступ з зовнішнього середовища відкритий, а для деяких - закритий.

Для організації доступу до даних та функцій класу використовують спеціальні специфікатори доступу: prіvate - доступ закритий*;* publіc **-** доступ відкритий.

Звичайно prіvate використовується для даних-членів класу з метою сховати від користувача деталі збереження даних в об'єктах, у той же час забезпечуючи їх методами можливість використання цих даних. У результаті в програмі можна модернізувати способи збереження й обробки даних у середині класу, не переписуючи при цьому методи доступу і виклику їх у зовнішньому коді.

Члени класу можуть бути закриті за замовчуванням, але використання в в структурі класу специфікатора prіvate бажано. В класі можуть бути декілько секцій відкритих і закритих компонентів класу, що розташовуються в довільному порядку

Для використання закритих (prіvate) даних-членів існує лише один спосіб – через виклик відкритих для зовнішнього середовища функцій-членів класу, тому що закриті (prіvate) члени класу доступні тільки членам цього класу і нікому більше, тому що закриті члени невидимі поза класом. Графічна інтерпретація спосібів доступу до даних та функцій- членів класу з зовнішнього середовища представлена на рисунку 1.2.

Нижче наведений приклад програми, що використовується спеціфікатори доступу private і public



Рисунок 1.2 - Графічна інтерпретація видів доступу до даних та методів класу з зовнішнього середовища

**Приклад**

# include < iostream.h >

Специфікатори

доступа

 #include <stdio.h>

class CTime {

private:

int year;

** int month*;* закриті дані члени

** int day;

** int hour;

** int minute*;*

 public:

 void Display(void);

** void GetTime(int &m,int &d,int&y,int &h2,int &min);

** void SetTime(int m,int d,int y,int h,int min);

};

int main ( )

 {

CTime obj1; // об’єкт типу CTime

int month, day, year, hous, minute;

obj1.SetTime (7, 14, 2003, 8, 30);

cout <<“obj1==“; obj1.Display();

obj1.GetTime(month, day, year, hous, minute);

obj1.GetTime(month,day,year, ++hous, minute);

cout <<“ Next hous==“; obj1.Display();

*Оператор дозволу області видимості*

return ();}

void CTime::Display(void)

{

char s [32];

Sprintf(s,“Data:%02d/%02d%/%04d Time:%02d:%02d\n” =, month, day, year, hour, minute);

cout <<s;

}

void CTime::Get Time(int&m,int&d,int&y,int&h2,int& min);

*}*

m=month; *//*  Повернення даних-членів тому, хто викликав функцію

d=day;

y=year;

h2=hare;

min=minute;

}

void CTime ::Set Time (int m, int d, int y, int h, int min);

{

 month = m;//Присвоювання аргументів даним-членам класу

day = d;

year = y;

hare = h2;

minute = min;

}

**Вбудовані функції-члени**

В мові С++ дозволена можливість використання класу з вбудованими функціями-членами,тобтоякщо тіло функції складається з декількох операторів, то її можна повністю описати в тілі класу.

Функції-члени класу, що вбудовуються в тіло класу, використовуються так само, як і інші функції, але при проектуванні структури класу треба пам’ятати, що компілятор вбудовує код функції в кожне місце її виклику замість оператору виклику, тобто у скомпільованому коді таки функції не викликаються, а вставляються безпосередньо в скомпільовану програму (рисунок 1.3).



Рисунок 1.3 – Геометрична інтерпретація вбудованих функцій членів класу

Наприклад,

# include < iostream.h >

# include < time.h >

# include < string.h >

class CTime {

private:

*long dt;* //дані дата і час перетворяться в секунди і представлені у виді довгого цілого. Це зручно для використання різних бібліотечних функцій, використаних такою формою збереження даних повертає у виді символьного рядка дату і час.

Вбудована

функція- член класу

рublic:

void Display(void){cout<<ctime(&dt);}

void Get Time (int & m, int & d, int & y, int & h2, int & min)*;*

void Set Time (int m, int d, int y, int h, int min*);*

char \* Get STime ( void )

{char \*cp = strdup(ctime (&dt)); }

return cp; }

//зсув в часі додає до поточного часу n mіnutes хвилин

void Change Time(long n minutes){dt+=(n minutes×60);}

}