ЛАБОРАТОРНА РОБОТА

ТЕМА: РОЗРОБКА ПРОГРАМ З ВИКОРИСТАННЯМ ПРИНЦИПІВ ПРОСТОГО СПАДКУВАННЯ

Мета: Набуття навичок в проектуванні найпростіших ієрархій класів та розробка найпростіших програм з використанням ієрархії класів

Порядок виконання роботи

1. Ознайомитися з теоретичними основами розробки ієрархії класів та правилами їх використання.

2. Розробити ієрархію з мінімум 2 класами для роботи з текстовими даними (масивом рядків) яка складається з:

а) базового класу, який містить: два конструктора, деструктор, функцію визначення довжини тексту, функцію визначення кількості рядків в тексті, функцію визначення кількості слів в тексті, функцію визначення кількості пробілів у тексті, функцію видалення зайвих пробілів з тексту.

б) похідного класу, що містить функцію з завдання 1.

3. Розробити тести для перевірки вірності даної програми.

4. Оформити звіт до лабораторної роботи.

Завдання 1

Варіант 1. Розробити метод-член похідного класу копіювання заданої кількості символів з заданої позиції в заданий рядок.

Варіант 2. Розробити метод-член похідного класу визначення частоти повторення символів.

Варіант 3. Розробити метод-член похідного класу видалення заданої кількості символів з заданої позиції.

Варіант 4. Розробити метод-член похідного класу видалення символів, які повторюються.

Варіант 5. Розробити метод-член похідного класу визначення кількості слів, які повторюються.

Варіант 6. Розробити метод-член похідного класу визначення кількості повторювань заданого набору символів.

Варіант 7. Розробити метод-член похідного класу визначення кількості слів в заданому тексті.

Варіант 8. Розробити метод-член похідного класу вставки заданого рядка в початок тексту.

Варіант 9. Розробити метод-член похідного класу вставки заданого рядка в кінець тексту.

Варіант 10. Розробити метод-член похідного класу вставки в заданою позицію заданого рядка в текст.

Теоретичні відомості

Принципи спадкування в ООП

Одним з важливих принципів технології ООП є спадкування. Тобте від одного класу можна утворювати підкласи. При побудові підкласів здійснюється спадкування даних і методів обробки об'єктів вихідного класу. Механізм спадкування дозволяє або перевизначати або додавати нові дані та методи їх обробки в успадкованих класах, що дозволяє створбвати ієрархії класів.

На даний час в ООП використовують два виду спадкування: просте та множинне.

Просте спадкування описує споріднення між двома класами, один з яких успадковує властивості та функції іншого класу, що називається базовим класом. Інші класи називаються похідними класами (рисунок 3.1).

Правила спадкування

1. З одного базового класу можна створювати багато похідних класів (рисунок 3.2).

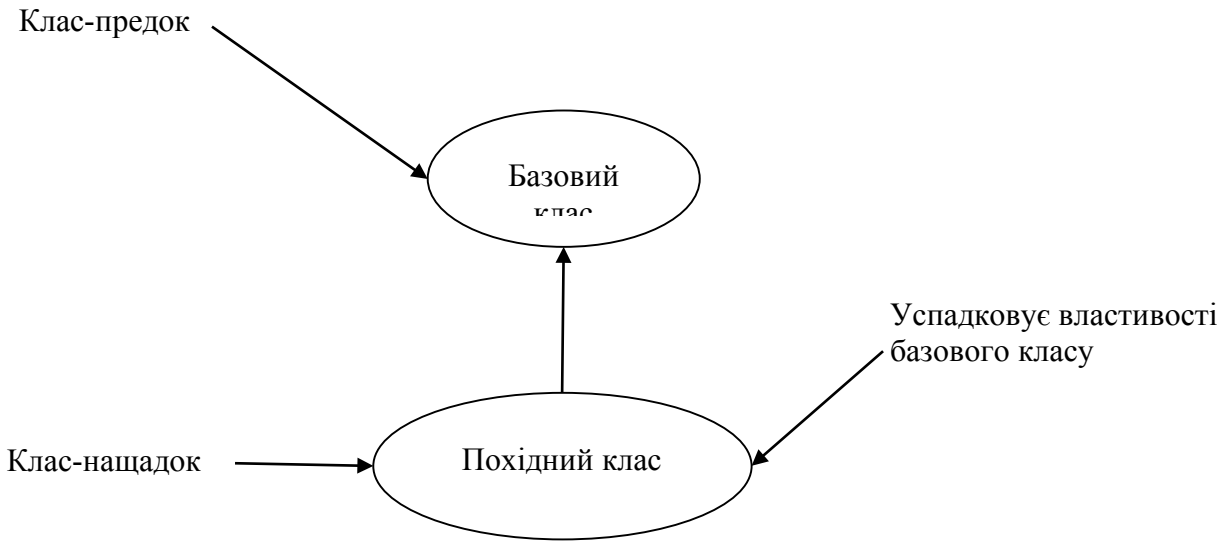

Рисунок 3.1 – Графічне представлення зв’язків між класами при спадкуванні

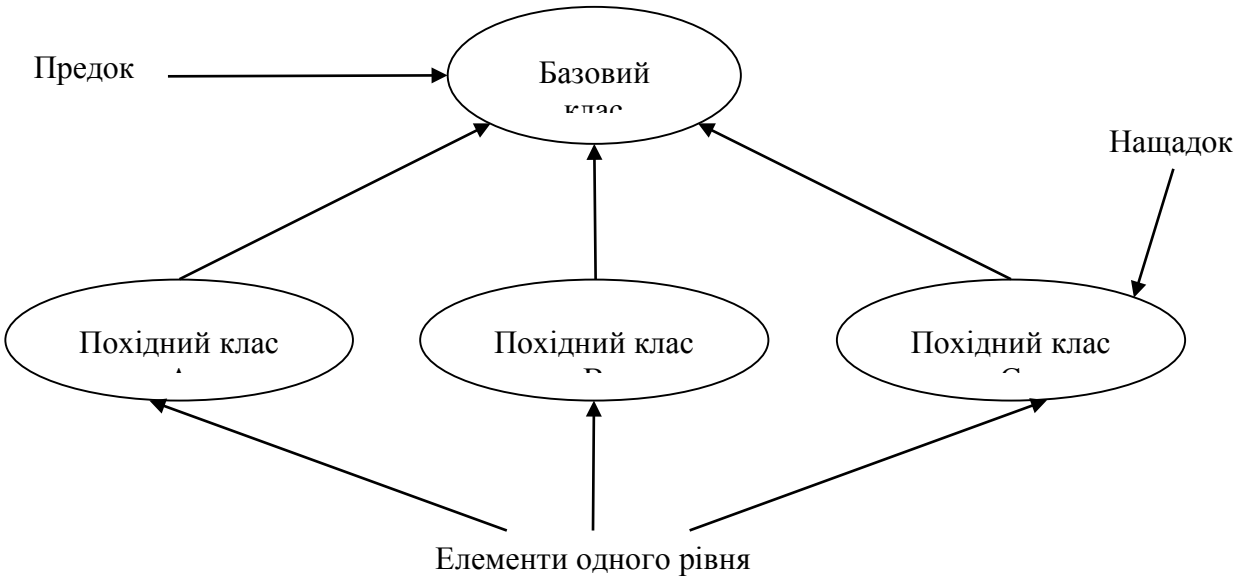

Рисунок 3.2 – Графічне представлення зв’язків між класами при простом спадкуванні

2. Похідний клас може бути базовим, тобто успадковувати інші класи (рисунок 3.3). Незважаючи на очевидну складність ієрархії класів, показану на рисунку 3.3, усі продемонстровані взаємозв'язки використовують просте спадкування, тому що кожен похідний клас має лише один базовий. Не існує яких-небудь обмежень на число похідних класів, що успадковують властивості базового.

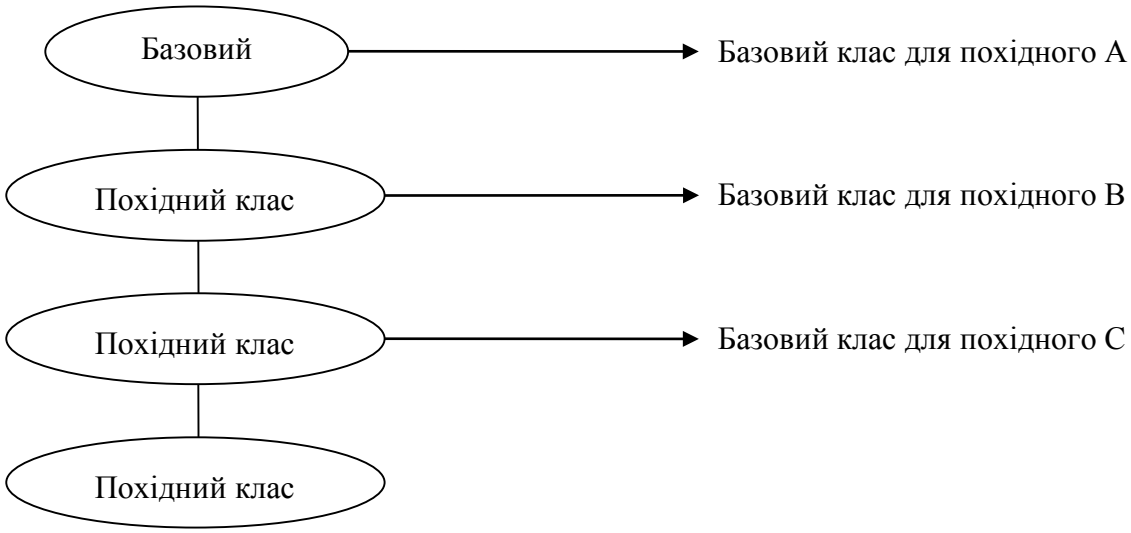

Рисунок 3.3 – Графічне представлення зв’язків між класами в ієрархії класів

3. Похідний клас успадковує з базового дані-члени і функції-члени, але не конструктори і деструктори. Наприклад, на малюнку 3.3 похідний клас А успадковує властивості базового класу; похідний клас В успадковує крім властивостей базового класу ще нові властивості класу А; похідний клас С успадковує властивості верхнього базового класу і похідних класів А і В. Тобто, похідний клас починає своє існування з копіювання членів базового класу, включаючи всі члени, успадковані з більш далеких.

Оголошення похідних класів

В C++ формат заголовку похідного класу має вигляд:

`class` ім’я похідного класу **:** специфікатор доступу ім’я базового класу`{`

```
  тіло класу
};
```

На рисунку 3.4 показани способи використання специфікаторів доступу при розробці похідних класів.

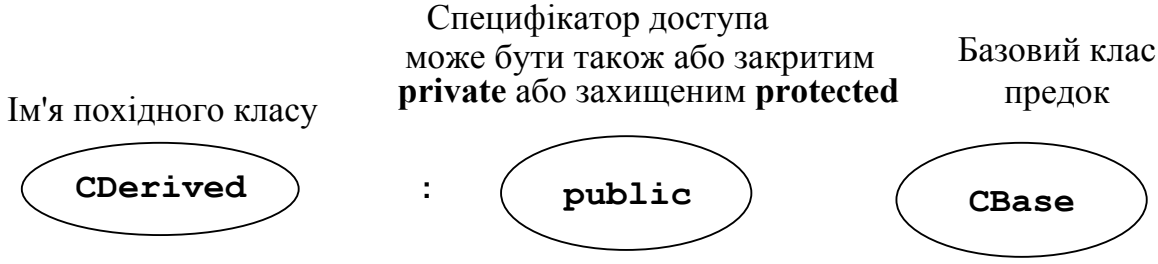

Рисунок 3.4 – Приклади використання специфікаторів доступу при розробці похідних класів

Наприклад, `CBase` - базовий клас виду:

```
Class CBase
{
   Private:
        int count;//Закриті дані класу
   Public:
 CBase ();{count=0;}
 void Set Count(int n){count=n;}// Конструктор
 int Get Count (void) {return count;}
};
```

де член `count` відноситься до закритих даних класу. Тільки функції-члени класу `CBase` можуть безпосередньо посилатися на `count`. Конструктор `CBase()` ініціалізує `count` нульовим значенням; функція-член `SetCount()` задає `count` нове значення. Функція-член `GetCount()` повертає поточне значення `count`. Для спрощення коду конструктор `CBase()` і обидві функції виконані як вбудовані.

Створимо похідний клас `CDerived`, що має усі властивості базового `CBase`, але крім того, здатний збільшувати і зменшувати значення параметра класу на задану величину. Для такого похідного класу `CDerived` заголовок буде мати вигляд:

```
Class CDerived: public CBase
{
  public:
    CDerived():CBase (){}
void ChangeCount(int n){SetCount(GetCount()+n); }
};
```

У даному прикладі відкриті члени відкритого класу `CBase` залишаються відкритими в похідному класі `CDerived`. До закритих членів класу `CBase` в класі `CDerived` не можна одержати доступ.

Похідні класи повинні викликати конструктори своїх базових класів. У даному прикладі конструктор не робить ніяких дій, тому не містить операторів.

```
Class:
{
   public:
      CDerived():CBase(){}
      ……….
};
```

Конструктори похідного класу

Основні правила для створення конструкторів:

1. У похідному класі є конструктор, якщо він є в базовому;
2. Конструктор похідного класу повинний викликати конструктор базового класу. Наприклад, `CDerived():CBase(){}`
3. При виклику імені конструктора базового класу йде після імені конструктора похідного класу.
4. Конструктор похідного класу може містити операторы:

`CDerived():CBase(){count <<"Iniutialized"}`

5. Конструктор похідного класу може бути описаний за межами класу.

Наприклад,

У класі `class CDerived: public CBase`

```
{
  public:
        CDerived ();
  voidChangeCount(int n){SetCount(GetCount()+n);}
};
```

де

ім'я класу *ім'я конструктора*

`CDerived::CDerived ();` // заголовок конструктора

`: CBase` // виклик конструктора базового класу

{

оператори конструктора

}

У даному прикладі виконується ініціалізація всіх членів, успадкованих з базового класу. У даному випадку конструктор базового класу встановлює закритий член класу `count` рівним нулю.

Функції члени похідного класу

Основні правила використання:

1. Якщо в базовому класі член класу `private`, то в похідному класі одержати до нього безпосередній доступ не можна, тому що тільки члени класу можуть мати доступ до закритих членів цього ж класу.

2. Якщо в похідному класі оголошені закриті члени, то з ними може працювати конструктор і функції-члени цього класу.

Наприклад,

```
class CDerived:public CBase
 {
  Private:
       int second Count;
  Public:
      CDerived (); // Конструктор похідного класу
   void Change Count(int n){Set Count (Get Count
() +n);}
  };
```

```
CDerived::CDerived(); // Заголовок конструктора
    : CBase () // Ініціалізація даних членів CBase
   {
    Second Count=0; // Ініціалізація даних членів Cderived
   }
```

3. Конструктор базового класу повинний викликатися першим, що забезпечить ініціалізацію всіх наступних членів до того, як будуть ініціалізовані члени похідного класу.

4. Функції-члени похідного класу не мають доступ до закритих даних членів базового класу, але можуть з ним працювати через `public` функції члени базового класу.

Захищені `(protected)` члени класу

1. Захищений член класу доступний членам цього класу і всім членам похідних класів (рисунок 3.5).
2. Поза класом захищені члени невидими (рисунок 3.5).
3. Подібно відкритим членам, захищені успадковуються похідними класами і доступні функціям-членам похідних класів(рисунок 3.5).

На рисунку 3.5 показані правила організації родинних зв'язків між даними базового і похідного класів з різними специфікаторами доступу.

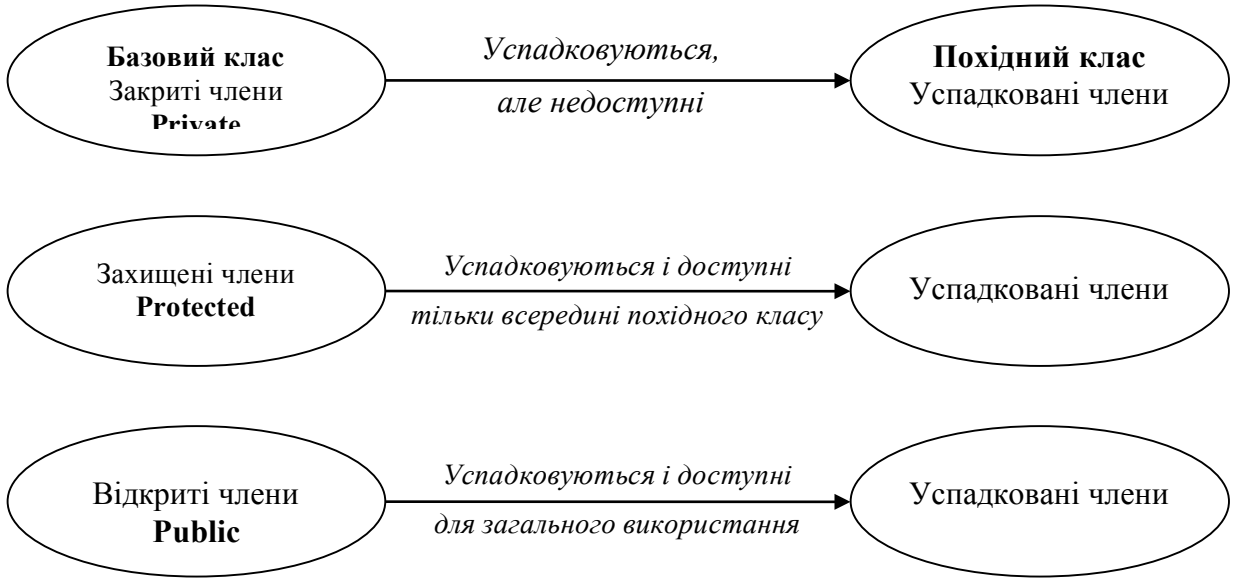

Рисунок 3.5 – Правила використання специфікаторів доступу в похідних класів

Правила успадкування специфікаторів доступу

– закриті члени доступні тільки в класі, у якому оголошені;

– захищені члени доступні членам їхнього власного класу і всіх членів похідних класів;

– відкриті члени доступні членам їхнього власного класу, членам похідних класів і всіх інших користувачів цього класу;

– специфікатори доступу `private, protected, public` ставляться перед ім'ям базового класу в оголошенні похідного класу і впливають на статус наслідуваних членів;

– члени відкритого базового класу в похідному класі зберігають свої специфікації доступу `private, protected, public`;

– відкриті члени захищеного базового класу стають захищеними в похідному класі. Захищені і закриті члени зберігають свої первісні специфікації доступу;

– усі члени закритого базового класу стають закритими в похідному класі незалежно від первісної специфікації доступу цих членів. Наприклад,

```
class CBase                    class CDerived: private
{                               {
 protected:                          Public:
       int x;                           void f(void);
 Public                          };
      int y;
 };
```

В данному прикладі функція-член `f()` класу `CDerived` має доступ до членів x та y, успадкованих з класу `CBase`. Однак, тому що `CBase` оголошений закритим базовим класом для класу `CDerived`, статус `x` та `y` змінився на `private`.

Розглянемо інший приклад.

```
class CDerived: public CBase
{
     Public:
         void g (void);
    };
```

У даному класі функція-член `g()` не має доступу до членів `x, y,` незважаючи на те, що спочатку вони мали статус `Protected` та `Public`.

Розглянемо приклад кваліфікації доступу до окремих членів. Клас A - базовий клас виду:

```
class A
  {
    public:
        int x;
      A(){count <<"A is constructor\n";x=0}
     void Display (void) {count<<"x=="<<x<<'\n';}
```

```
};
```

Похідний клас B може успадковувати A в якості закритого базового класу:

```
Class B:private A
  {
   public
       B():A(){count <<"B is constructor \n";}
  };
```

Оскільки клас A оголошений закритим в B, усі колись відкриті в A члени закриті у B. Це означає, що у всіх наступних похідних від B чи класах зовнішніх стосовно B операторах не можна викликати функцію-член `Display();` успадковану з класу A.

Наприклад, для базового класу A в програмі використовується объект класу `objA:`

```
A objA; // объект класу A
objA.Display (); // оператор записаний вірно.
```

А для похідного класу B: наступний запис невірний:

```
B obj B;
obj B. Display (); ⟶ Невірно!!!
```

Якщо необхідно в похідному класі вибірково зробити відкритими, то для цього базовий клас A з'являється закритим в B, а у відкритої `Public` секції B кваліфікуються члени, які необхідно залишити відкритими:

```
Class B:private A {
 Public:
/*вказує компілятору, що цей член класу A ідкритий*/
    A :: Display;
    B():A(){count <<"B is constructor \n" ;}
```

```
};
```

У цьому випадку оператор

```
B obj B;
obj B.Display (); ⟶ Вірний!!!
```

Приклад виконання Завдання 1

Варіант 1. Похідний клас містить метод (функцію) копіювання заданої кількості символів з заданої позиції.

Дана програма містить два класи, що працюють з текстовим файлом. Базовий клас `CMain` містить конструктор `CMain()`, за допомогою якого файл відкривається автоматично після ініціалізації даних в головній програмі, деструктор `~CMain()`, що автоматично викликається на заключному етапі виконання програми і виконує закриття файлу. Похідний клас `CCSuccessor` містить функцію `Copy_to_file()`, що призначена для копіювання заданої кількості символів з заданої позиції.

Структура класу має вигляд:

```
class CMain{
char c;
int Num,UK,KIL;
protected:
FILE* stream;
public:
CMain();
~CMain();
void Quantity_of_chars();
void Number_of_space();
void Number_of_words();
void Delete_wrong_space();
};
class CCSuccessor: public CMain {
public:
CCSuccessor():CMain(){}
void Copy_to_file();
};
```

Лістинг програми:

```
#include <cstring>
#include <cstdio>
#include <cstdlib>
class CMain{
char c;
int Num,UK,KIL;
protected:
FILE* stream;
public:
CMain();
~CMain();
void Quantity_of_chars();
void Number_of_space();
void Number_of_words();
void Delete_wrong_space();
};
class CSuccessor: public CMain {
public:
CSuccessor():CMain(){}
void Copy_to_file();
};
int main (){
CSuccessor obj;
obj.Quantity_of_chars();
obj.Number_of_space();
obj.Number_of_words();
obj.Delete_wrong_space();
obj.Copy_to_file();
return 0;
}
CMain::CMain()
{
stream = fopen( "Dat.txt", "r+");
}
CMain::~CMain()
{
fclose(stream);
}
void CMain::Quantity_of_chars(){
Num = 0;
while ((c = getc(stream)) != EOF){
    Num++;}
printf("Quantity of chars=%i\n", Num);
}
```

```
void CMain::Number_of_space(){
stream = fopen( "Dat.txt", "r+");
int i = 0;
while ((c = getc(stream)) != EOF) {
    if (c == ' ') i++;}
printf("Quantity of space symbols=%i\n", i);
}
void  CMain::Number_of_words(){
stream = fopen( "Dat.txt", "r+");
char *Mas = new char[Num];
int i = 0;
while ((c = getc(stream)) != EOF){
    Mas[i]=c;
    i++;
}
UK=0;
KIL = 0;
for(i=0; i<=Num; i++) {
 if((Mas[i]==' ')||(Mas[i]=='\n'))
    UK++;
 if(((Mas[i]!=' ')&&(Mas[i]!='\n'))&&(UK>=1))
 {
        KIL++;
        UK = 0;
 }
}
printf("Quantity of words=%i\n", KIL+1);
delete Mas;
}
void CMain::Delete_wrong_space(){
UK=0;
stream = freopen( "Dat.txt", "r+",stream);
char *Mas = new char[Num];
int i = 0;
while ((c = getc(stream)) != EOF){
    if(c== ' ') UK++;
    else{
        Mas[i]=c;
        UK=0;
    }
    if((c == ' ')&&(UK==1)) Mas[i]=c;
    if((c== ' ')&&(UK >1)) i--;
    i++;
}
```

```
UK=i;
printf("\n");
for(i=0; i<UK; i++){
    printf("%c",Mas[i]);
}
printf("\n");
}
void CCSuccessor::Copy_to_file(){
int pos=0,kil=0;
printf("\nInput position and number of
symbols\n ");
scanf("%i %i",&pos,&kil);
char c;
int i=0,j=0;
FILE *stream2;
stream = freopen("Dat.txt","r+",stream);
stream2 = fopen( "Dat2.txt", "w");
while ((c = getc(stream)) != EOF) {
    i++;
    if((i>=pos)&&(j<kil)){
        putc(c,stream2);
        j++;
    }
}
}
```

Результати тестування даної програми представлені ніжче.

Для перевірки правильності роботи програми створимо файл такого виду:

```
aaa   bb     ccccc
ffff
ggggggg
```

Перевіримо, який результат видасть програма:

```
Quantity of chars=32
Quantity of space symbols=8
Quantity of words=5
aaa bb ccccc
ffff
ggggggg
Input position and number of symbols
 7 2
Press any key to continue . . .
```

Зміст файлу Dat2.txt буде наступним:

```
bb
```

Контрольні запитання

1. В чому основний зміст принципу спадкування класів в ООП?
2. У чому полягає просте спадкування? Навести приклад.
3. Умови спадкування членів базового класу.
4. Умови формування ієрархії класів. Навести приклад.
5. Послідовність формування ієрархії класів за правилами простого спадкування.
6. У чому полягає множинне спадкування? Навести приклад.
7. Правила використання специфікаторів доступу при розробці похідних класів.
8. Правила організації захищених членів базового класу. Навести приклад.
9. Правила організації захищених членів похідного класу. Навести приклад.
10. Правила доступу до членів базових класів з похідних класів. Навести приклад.
11. Вимоги до конструкторів базового та похідних класів. Навести приклад.
12. Вимоги до деструкторів базового та похідних класів.
13. Умови спадкування в похідних класах. Навести приклад.