Міністерство освіти і науки України
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ
до самостійної роботи
з дисципліни
“Об’єктно-орієнтоване програмування”
для студентів
напряму підготовки
121 «Інженерія програмного забезпечення»
(всіх форм навчання)

2017

Методичні вказівки до самостійної роботи з дисципліни «Об’єктно-орієнтоване програмування» для студентів напряму підготовки 121 «Інженерія програмного забезпечення»(всіх форм навчання) / Г. В Табунщик, Н. О. Миронова, Т. І. Каплієнко. – Запоріжжя: ЗНТУ, 2017. – 68 с.

Автори: Г. В. Табунщик, к.т.н., професор
Н. О. Миронова, к.т.н., доцент
Т. І. Каплієнко, к.т.н., доцент

Рецензент: Т.В. Федорончак, к.т.н., доцент

Відповідальний
за випуск: Г. В. Табунщик, к.т.н., професор

Затверджено
на засіданні кафедри
програмних засобів

Протокол №11
від 06 червня 2017

ЗМІСТ

ВСТУП

Метою даного курсу є вивчення теоретичних основ та практичних аспектів об'єктно-орієнтованого програмування. Дисципліна “Об’єктно-орієнтоване програмування” спрямована на отримання студентом базових знань та практичних навичок з основ сучасної технології створення складних програмних продуктів на базі ідей і принципів об'єктно-орієнтованого методу. Такі знання призначені для використання у розробках програмного забезпечення інформаційних технологій у проектуванні з урахуванням сучасних вимог у відношенні до надійності, якості інтерфейсу та ефективності програмних продуктів, які створюються. Отримані знання та практичні навички мають служити базою для опанування у подальшому нових майбутніх систем програмування, які базуються на ідеях візуального програмування, CASE-технологіях, штучного інтелекту і та інше.

В якості інструментальної мови програмування для виконання самостійних робіт рекомендовано використовувати мову програмування C++. Головною вимогою для використання компілятора є підтримка стандарту *ISO/IEC 14882 “Standard for the C++ Programming Language”*.

САМОСТІЙНА РОБОТА №1
СТВОРЕННЯ ПРОЕКТУ В QT

Мета роботи

Навчитись створювати консольні проекти в **Qt**.

Как создать проект в Qt

Для створення нового проекту в **Qt** потрібно виконати наступну послідовність дій: Файл->Новый файл или проект… ->Проект без использования **Qt** -> Простой проект на языке C++. Далі виконати введення назви проекту, шлях розміщення. Далі 2 рази кнопку «Завершить».

У файлі main.cpp написати програму, яка наведена нижче.. Потім запустити на виконання (зелений трикутник зліва).

Завдання 1.1 Використання компаратора в структурах даних

Одного разу, незручна секретарка переплутала особові справи учнів. Тепер їх знову необхідно впорядкувати спочатку по класам, а всередині класу за прізвищем.

У першому рядку дано число N ($1 \le N \le 1000$ чоловік) – кількість особових справ. Далі для кожного з N учнів наступні дані (кожне у своєму рядку): прізвище та ім'я, клас, дата народження. Прізвище та ім'я – рядки не більше ніж з 20 символів, клас – рядок складається з числа (від 1 до 11) і латинської літери (від "A" до "Z"), дата народження – дата в форматі "ДД.ММ.ГГ ". Гарантується, що всередині одного класу відстуні однофамільці.

Потрібно вивести N рядків, в кожному з яких записано дані по одному учню. Рядки повинні бути впорядковані спочатку по класам, а потім за прізвищами.

Лістинг 1.1.

```
#include <iostream>
#include <algorithm>
#include <string>
using namespace std;
```

```
const int N = 1001;

struct puple
{
   string sname;
   string fname;
   string _class;
   string date;
};

bool cmp(const puple& a, const puple& b)
{
   if (a._class.length()!=b._class.length())
      return a._class.length()<b._class.length();
   if (a._class!=b._class)
      return a._class<b._class;
      return a.sname<b.sname;
}

puple array[N];

int main()
{
int n;
cin>>n;
for (int i=0; i<n; i++)

cin>>array[i].sname>>array[i].fname>>array[i]._class>>array[i].date;
sort(array, array+n, cmp);
for (int i=0; i<n; i++)
   cout<<array[i]._class<<"  "<<array[i].sname<<"  "<<array[i].fname
<<" "<<array[i].date<<endl;
   cin.get(); cin.get();
return 0;
}
```

САМОСТІЙНА РОБОТА №2
ВВЕДЕННЯ В КЛАСИ

Мета роботи

Навчитись будувати класи та використовувати їх при створенні програм.

Завдання 2.1 Класу Coord з методами класу

Створити клас для роботи з трьохвимірними векторами. Передбачити функції для виконання наступних операцій: консольне введення і виведення значень вектора; ініціалізація вектора; складання векторів та віднімання векторів.

Лістинг 2.1.

```
#include <iostream>
#include <conio.h>
#include <stdio.h>
using namespace std;
class Coord {
    int x,y,z;
public:
    Coord() {x=0; y=0;z=0;}
    Coord(int i,int j, int k) {x=i;y=j;z=k;}
    void input();
    void show();
    Coord add(Coord obj1);
    Coord sub(Coord odj1);
};

Coord Coord::add(Coord obj2)
{
  Coord temp;
   temp.x=x+obj2.x;
   temp.y=y+obj2.y;
  temp.z=z+obj2.z;
   return temp;
```

```
}
Coord Coord::sub(Coord obj2)
{
  Coord temp;
   temp.x=x-obj2.x;
   temp.y=y-obj2.y;
  temp.z=z-obj2.z;
   return temp;
}

void Coord::input()
{
  cout<<"Input vector"<<endl;
  cin>>x>>y>>z;
}
void Coord::show()
{
  cout<<x<<' '<<y<<' '<<z<<endl;
}
int main()
{Coord a(10,10,10),b,c,d;
b.input();
c=b.add(a);
c.show();
d=b.sub(a);
d.show();
return 0;
}
```

Завдання 2.2 Класу Coord з використанням дружньої функції

Створити клас для роботи з трьохвимірними векторами. Передбачити функції для виконання наступних операцій: консольне введення і виведення значень вектора; ініціалізація вектора; дружні функції складання та віднімання векторів.

Лістинг 2.2.

```
#include <iostream>
#include <conio.h>
#include <stdio.h>
using namespace std;
class Coord {
    int x,y,z;
public:
    Coord() {x=0; y=0;z=0;}
    Coord(int i,int j, int k) {x=i;y=j;z=k;}
    void input();
    void show();
    friend Coord add(Coord obj1, Coord obj2);
    friend Coord sub(Coord obj1, Coord obj2);
};

Coord add(Coord obj1, Coord obj2)
{
  Coord temp;
   temp.x=obj1.x+obj2.x;
   temp.y=obj1.y+obj2.y;
  temp.z=obj1.z+obj2.z;
   return temp;
}

Coord sub(Coord obj1, Coord obj2)
{
  Coord temp;
   temp.x=obj1.x-obj2.x;
   temp.y=obj1.y-obj2.y;
  temp.z=obj1.z-obj2.z;
   return temp;
}

void Coord::input()
{
  cout<<"Input vector"<<endl;
```

```
  cin>>x>>y>>z;
}
void Coord::show()
{
  cout<<x<<' '<<y<<' '<<z<<endl;
}
int main()
{Coord a(10,10,10),b,c,d;
b.input();
c=add(a,b);
c.show();
d=sub(a,b);
d.show();
return 0;
}
```

Завдання 2.3 Клас *Alfa* з використанням статичної компоненти

Створити клас *Alfa* таким чином, щоб при створенні першого і знищенні останнього об'єкта цього типу на екран видавалися відповідні повідомлення. Вказівка: застосувати статичні компоненти класу.

Лістинг 2.3.

```
#include <iostream>
#include <conio.h>
using namespace std;

class Alpha{
public:
static int n;
 Alpha();
~Alpha();
};
int Alpha::n=0;

Alpha::Alpha()
{
```

```
  if (n==0) cout<<"First object has been created"<<endl;
     n++;
}

Alpha::~Alpha()
{
  n--;
  if (n==0) cout<<"Last element has been deleted";
}

int main()
{
  Alpha *a;
  a=new Alpha[5];
  delete [] a;
return 0;
}
```

САМОСТІЙНА РОБОТА №3
ДИНАМІЧНІ КЛАСИ ТА СТРУКТУРИ ДАНИХ

Мета роботи

Навчитись використовувати динамічні класи та структури даних при створенні програм.

Завдання 3.1 Клас DynArr (динамічний одномірний масив)

Створити клас для роботи з одновимірними динамічними масивами значень типа *unsigned int*. Передбачити функції-компоненти класу для виконання наступних операцій: динамічного присвоєння; об'єднання двох масивів у один (конкатенація).

Лістинг 3.1.

```
#include <iostream>
using namespace std;

class DynArray
```

```
{
  unsigned int *arr;
  int n;
public:
  DynArray(int count);
  DynArray(const DynArray &obj);
  ~DynArray();
  void show();
  DynArray& append(const DynArray &obj);
};

DynArray::DynArray(int count)
{
  n = count;
  arr = new unsigned int[n];
  for (int i=0; i<n; i++)
            arr[i] = i;
}

DynArray::DynArray(const DynArray &obj)
{
  n = obj.n;
  arr = new unsigned int[n];
  for (int i=0; i<n; i++)
            arr[i]=obj.arr[i];
}

DynArray::~DynArray() {
  delete [] arr;
}

void DynArray::show() {
  for (int i=0; i<n; i++)
            cout << arr[i] << " ";
  cout << endl;
}

DynArray& DynArray::append(const DynArray &obj) {
```

```
  unsigned int *tmp = new unsigned int[n + obj.n];
  int i;
  for (i=0; i<n; i++)
          tmp[i] = arr[i];
  for (i=0; i<obj.n; i++)
          tmp[n+i] = obj.arr[i];
  delete [] arr;
  arr = tmp;
  n += obj.n;
  return *this;}

int main() {
  DynArray obj1(5);
  DynArray obj2(3);

  obj1.show();
  obj2.show();

  obj1.append(obj2);
  obj1.show();
  cin.get();
  return 0;
}
```

Завдання 3.2 Динамічна структура даних лінійний однозв'язний список

Інформація про книжки зведена у документ.

Найменування	Автор	Кількість сторінок у книгах	Рік видання	Місце видання

Зберігати дані у однозв'язному списку. Написати функцію, що виконує сортування за роком видання.

Лістинг 3.2.

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

```
using namespace std;

struct books
{
  char name [30];
  char autor [20];
  int page;
  int year;
  char place [10];
};

struct elem
{
 books data;
 elem *pnext;
};

elem *plist=NULL,*pfirst;
int i,n,j;

void addend()
{
elem *tmp=new elem;
cout<<"Input name of book "; cin>>tmp->data.name;
cout<<"Input autor of book "; cin>>tmp->data.autor;
cout<<"Input page of book "; cin>>tmp->data.page;
cout<<"Input year of edition "; cin>>tmp->data.year;
cout<<"Input place of edition "; cin>>tmp->data.place;
if(!plist)
{
 tmp->pnext=NULL;
 plist=tmp;
 pfirst=plist;
}
else
{
 plist->pnext=tmp;
 tmp->pnext=NULL;
```

```
 plist=tmp;
}
}

void sort()
{
int x;
elem *pv;
    for(i=0;i<n-1;i++){
       pv=pfirst;
     for(j=0;j<n-1;j++){
       if (pv->data.year>pv->pnext->data.year)
         {
          x=pv->data.year;
          pv->data.year=pv->pnext->data.year;
          pv->pnext->data.year=x;
          }
     pv=pv->pnext;
                }
             }
}

void show()
{
elem*tmp=pfirst;
while(tmp)
{
cout<<tmp->data.name<<"       "<<tmp->data.autor<<"       "<<tmp-
>data.page<<" "<<tmp->data.year<<" "<<tmp->data.place<<endl;
tmp=tmp->pnext;
}
}

int main()
{
cout<<"Information of books"<<endl;
cout<<"Input num=";
cin>>n;
```

```
for(i=0;i<n;i++) addend();
show();
sort();
show();
return 0;
}
```

Завдання 3.3 Динамічна структура даних лінійний двозв'язний список

Інформація про книжки зведена у документ.

Найменування	Автор	Кількість сторінок у книгах	Рік видання	Місце видання

Зберігати дані у двозв'язаному списку. Написати функцію, що виконує пошук книг за автором.

Лістинг 3.3.

```
#include <iostream>
#include <conio.h>
using namespace std;

struct books
{
 char name [30];
 char autor [20];
 int page;
 int year;
 char place [10];
};

struct elem
{
 books data;
 elem *prev;
 elem *pnext;

};
```

```
elem *plist=NULL,*pfirst, *tmp;
int i,n,j;
char f1[20];

void addend()
{
elem *tmp=new elem;
cout<<"Input name of book "; cin>>tmp->data.name;
cout<<"Input autor of book "; cin>>tmp->data.autor;
cout<<"Input page of book "; cin>>tmp->data.page;
cout<<"Input year of edition "; cin>>tmp->data.year;
cout<<"Input place of edition "; cin>>tmp->data.place;
if(!plist)
{
 tmp->prev=NULL;
 tmp->pnext=NULL;
 plist=tmp;
 pfirst=plist;
}
else
{
 plist->pnext=tmp;
 tmp->prev=plist;
 tmp->pnext=NULL;
 plist=tmp;
}
}

bool find(elem *el)
{
   if(strcmp(f1,el->data.autor)==0) return true;
 return false;
}

void show()
{
elem*tmp=pfirst;
```

```
        while(tmp)
        {
        cout<<tmp->data.name<<"       "<<tmp->data.autor<<"       "<<tmp-
>data.page<<" "<<tmp->data.year<<" "<<tmp->data.place<<endl;
        tmp=tmp->pnext;
        }
        }


        int main()
        {
        cout<<"Information of books"<<endl;
        cout<<"Input num=";
        cin>>n;
        for(i=0;i<n;i++) addend();
        show();
        cout << "Vvedite avtora books: "; cin >> f1;
        tmp = pfirst;
          for (i=0;i<n;i++) {
                    if (find(tmp)) {
                     cout<<tmp->data.name<<"           "<<tmp->data.autor<<"
"<<tmp->data.page<<" "<<tmp->data.year<<" "<<tmp->data.place<<endl;
                    }
                    tmp = tmp->pnext;
          }
        return 0;
        }
```

Завдання 3.4 Клас *Stack*

Створити клас *Stack* – стек, що базується на структурі зв'язного списку. Тип значення елементів стеку – char. Передбачити функції-члени класу для виконання наступних операцій:

push() – занести у стек надане значення;

print() – вивести усі значення стека на екран.

Лістинг 3.4

```
#include <iostream>
#include <conio.h>
using namespace std;
class Node
{ public:
  char d; // дані
  Node *next; // покажчик на наступний вузол
  Node (char dat = 0) { d = dat; next = 0;} // конструктор
};

class Stack
{Node *top; // покажчик на вершину стека
public:
    Stack()   { top = 0;} // конструктор
  ~Stack(); // деструктор
  void push(char d); // занесення елементу до стеку
  void print(); // друк стеку
};

void Stack::push(char d)
{
  Node *pv = new Node(d); // виділення пам'яті під новий вузол
  if(!top) { top = pv;} // формування вершини стеку
  else // зв'язування нового вузла з попереднім:
  { pv->next= top;
     top =pv;} // оновлення покажчика на вершину стеку
}

void Stack::print()
{ Node *pv = top;
  cout << endl << " stack: ";
  while(pv) { cout << pv->d << " ";
                  pv = pv->next; }
  cout << endl;
}
```

```
Stack::~Stack()
{ if(top)
  {         Node *pv = top;
            while(pv) {pv = pv->next;
                       delete top;
                       top = pv;}
  }
}

int main()
{
  Stack S;
  S.push('B');
  S.push('E');
  S.push('T');
  S.push('A');
  S.print();
  return 0;
}
```

Завдання 3.5 Клас *Queue*

Створити клас *Queue* – структура типу "черга", що базується на структурі зв'язного списку. Тип значення елементів черги – char..Передбачити функції для виконання таких операцій:

add() – занести у кінець черги значення;

print() – вивести усі значення, що знаходяться у черзі, на екран.

Лістинг 3.5

```
#include <iostream>
#include <conio.h>
using namespace std;

class Node
{public:
  char d; // дані
  Node *next; // покажчик на наступний вузол
  Node (char dat = 0) { d = dat; next = 0;} // конструктор
```

```
};
class Queue
{ Node *pbeg, *pend; // покажчики на початок та кінець черги
public:
   Queue() {pbeg = 0;} // конструктор
  ~Queue();// деструктор
  void add(char d); // занесення елементу до черги
  void print();// друк черги
};
void Queue::add(char d)
{
  Node *pv = new Node(d); // виділення пам'яті під новий вузол
  if(!pbeg) {pbeg = pv; pend = pv;} // формування черги
  else // зв'язування нового вузла з попереднім:
{pend->next=pv;
  pend = pv;} // оновлення покажчика на кінець черги
}
void Queue::print()
{ Node *pv = pbeg;
  cout << endl << " queue: ";
  while(pv) {cout << pv->d << " ";
  pv = pv->next;  }
  cout << endl;
}

Queue::~Queue()
{ if(pbeg)
  {         Node *pv = pbeg;
            while(pv)
            {         pv = pv->next;
                      delete pbeg;
                      pbeg = pv;
            }
  }
}
int main()
{
  Queue Q;
```

```
  Q.add('B');
  Q.add('E');
  Q.add('T');
  Q.add('A');
  Q.print();
  return 0;
}
```

САМОСТІЙНА РОБОТА №4
СПАДКУВАННЯ

Мета роботи

Навчитись використовувати спадкування при розробці об'єктно-орієнтованих програм.

Завдання 4.1 Таблиця спадкування

тип спадкування тип в базовому класі	из main()	private	protected	public
private	-	-	-	-
protected	-	+	+	+
public	+	+	+	+

Лістинг 4.1.

```
#include <iostream>
using namespace std;

 class base
 {
   int a;
  protected:
    int b;
 public:
   int c;
 base (){ a=0, b=0, c=0; }
 base (int x, int y, int z)
```

```
{ a=x, b=y, c=z; }
void show ();
};
class derived1: private base
{
public:
  void set(int x, int y, int z)
  {
//                    a=x;
            b=y;
            c=z;
  }
  void show();
};

class derived2:protected base
{
  public:
  void set(int x, int y, int z)
  {
//                    a=x;
            b=y;
            c=z;
  }
  void show();
};
class derived3:public base
{
  public:
  void set(int x, int y, int z)
  {
//                    a=x;
            b=y;
            c=z;
  }
  void show();
};
void base::show()
```

```
{
   cout<<"In base"<< endl;
   cout<<a;
   cout<<b;
   cout<<c;
}
void derived1::show()
{
   cout<<"In derived 1"<< endl;
//           cout<<a;
   cout<<b;
   cout<<c;
}
void derived2::show()
{
   cout<<"In derived 2"<< endl;
//           cout<<a;
   cout<<b;
   cout<<c;
}
void derived3::show()
{
   cout<<"In derived 3"<< endl;
//           cout<<a;
   cout<<b;
   cout<<c;
}
int main ()
 {
  base ob(1,2,3);
  //  cout << "a="<<ob.a<<endl;
  //cout << "b="<<ob.b<<endl;
  cout << "c="<<ob.c<<endl;

 derived1 ob1;
 ob1.set(4,5,6);
 ob1.show();
```

```
  derived2 ob2;
  ob1.set(7,8,9);
  ob1.show();

  derived3 ob3;
  ob1.set(10,11,12);
  ob1.show();
return 0;
}
```

Завдання 4.2 Клас Human

Створити базовий клас Людина. Кожний об'єкт класу повинен містити наступні дані: ПІБ. Клас повинен виконувати наступні дії: ініціалізація інформації, введення-виведення інформації.

Створити похідний клас Студент, що має додаткові дані: рік вступу, № залікової книжки. Клас повинен виконувати наступні функції: ініціалізація інформації, виводити загальну інформацію про студента.

Лістинг 4.2.

```
#include <iostream>
#include <string>

using namespace std;

class Human {
protected:
   string name;

public:
   Human(string s);
   string getName();
   void setName(string s);
   void showData();
};
```

```
class Student : public Human {
protected:
   int zk_number;
public:
   Student(string s, int n);
   int getZk_number();
   void setZk_number(int n);
   void showData();
};

Human::Human(string s) {
   setName(s);
}

void Human::setName(string s) {
   name = s;

}

string Human::getName() {
   return name;
}

void Human::showData() {
   cout << "name: " << name << endl;
}

Student::Student(string s, int n) : Human(s) {
   zk_number = n;
}

int Student::getZk_number() {
   return zk_number;
}

void Student::setZk_number(int n) {
   zk_number = n;
}
```

```
void Student::showData() {
   cout << "name: " << name << endl;
   cout << "zk_number: " << zk_number << endl;
}

int main() {
   Human obj1("Ivan");
   obj1.showData();
   obj1.setName("Ivan Ivanov");
   cout << "Name of obj1 is " << obj1.getName() << "\n\n";

   Student obj2("Alex", 123456789);
   obj2.showData();
   obj2.setName("Alexandr");
   obj2.setZk_number(987654321);
   cout << "Name of obj2 is " << obj2.getName() << "\nzk_number is
" <<obj2.getZk_number() << endl;

   cin.get();
   return 0;
}
```

САМОСТІЙНА РОБОТА №5
ВВЕДЕННЯ/ВИВЕДЕННЯ ПОТОКАМИ. РОБОТА З ФАЙЛАМИ

Мета роботи

Навчитись маніпулювати потоками введення/виведення, працювати з файлами та форматувати дані при введенні/виведенні.

Завдання 5.1 Форматування потоку

Лістинг 5.1.

```
#include <iostream>
#include <iomanip>
```

```
#include <conio.h>
using namespace std;

int main()
{
  //флаги
  long fl;
  fl=cout.flags();
  cout << "Initial state flags: "<< fl<<"\n";

   cout.setf(ios::showpos);
   cout.setf(ios::scientific);
   cout << 123 << " " << 1.2345678 <<"\n";

   cout.setf(ios::hex | ios::showbase);
   cout.unsetf(ios::showpos);
   cout.width(20);
   cout.precision(10);
   cout << 123 << " " << 123.456 << " " << 1.2345678 << "\n";
   cout << "New state flags: " << cout.flags() << "\n";
   cout.flags(fl);
   cout << "After recovery the initial state flags: \n";
   cout << 123 << " " << 123.456 << 1.2345678 << "\n";
  //манипуляторы
  cout << setprecision(2) << 100.5375 << endl;
   cout << setw(20) << "MANIPULATORS \n";

  return 0;
}
```

Завдання 5.2 Методи консольного та файлового введення/ виведення

Для завдання 3.1 реалізувати методи консольного та файлового введення/виведення.

Лістинг 5.2.

```
#include <iostream>
#include <fstream>
using namespace std;

class DynArray {
private:
  int *arr;
  int n;

public:
  DynArray(int count);
  ~DynArray();
  DynArray(const DynArray& obj);

  void show();
  void toFile();
  void fromFile();
};

DynArray::DynArray(int count) {
  n = count;
  arr = new int[n];
  for (int i=0; i<n; i++)
        arr[i] = i;
}

DynArray::~DynArray() {
  delete [] arr;
}

DynArray::DynArray(const DynArray &obj) {
  n = obj.n;
  arr = new int[n];
  for (int i=0; i<n; i++)
        arr[i] = obj.arr[i];
}

void DynArray::show() {
```

```
  for (int i=0; i<n; i++)
        cout << arr[i] << " ";
  cout << endl;
}

void DynArray::toFile() {
  ofstream ofs("1.txt");
  if (ofs.is_open()) {
        ofs << n << " ";
        for (int i=0; i<n; i++)
              ofs << arr[i]*2 << " ";
  }
  ofs.close();
}

void DynArray::fromFile() {
  ifstream ifs("1.txt");
  if (ifs.is_open()) {
        ifs >> n;
        arr = new int[n];
        for (int i=0; i<n; i++)
              ifs >> arr[i];
  }
  ifs.close();
}

int main() {
  DynArray obj1(5);
  obj1.show();
  obj1.toFile();
  obj1.fromFile();
  obj1.show();
  cin.get();
  return 0;
}
```

Зміст файлу 1.txt

5 0 2 4 6 8

Завдання 5.3 Створення маніпуляторів *insetup* та *outsetup* для форматування відповідно потоків введення/виведення

Для завдання 3.1 створити маніпулятори *insetup* та *outsetup* для форматування відповідно потоків введення/виведення.

Лістинг 5.3.

```
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;

class DynArray {
private:
  unsigned int *arr;
  int n;

public:
  DynArray(int count);
  ~DynArray();
  DynArray(const DynArray& obj);

  void show();
  void toFile();
  void fromFile();
};

istream &insetup (istream &ifs)
{
     ifs.setf(ios::scientific | ios::skipws | ios::hex);
     return ifs;
}
ostream &outsetup (ostream &ofs)
{
    ofs.fill(' ');
     ofs<<setw(4);
```

```
    return ofs;
}

DynArray::DynArray(int count) {
  n = count;
  arr = new unsigned int[n];
  for (int i=0; i<n; i++)
            arr[i] = i;
}

DynArray::~DynArray() {
  delete [] arr;
}

DynArray::DynArray(const DynArray &obj) {
  n = obj.n;
  arr = new unsigned int[n];
  for (int i=0; i<n; i++)
            arr[i] = obj.arr[i];
}

void DynArray::show() {
  for (int i=0; i<n; i++)
            cout << arr[i] << " ";
  cout << endl;
}

void DynArray::toFile() {
  ofstream ofs("1.txt");
  if (ofs.is_open()) {
            ofs << n << " ";
            for (int i=0; i<n; i++) {
                      ofs << outsetup <<arr[i]*2 << " ";
            }
  }
  ofs.close();
}
```

```
void DynArray::fromFile() {
  ifstream ifs("1.txt");
  if (ifs.is_open()) {
            ifs >> n;
            arr = new unsigned int[n];
            for (int i=0; i<n; i++)
            {ifs >> insetup >> arr[i];}
  }
  ifs.close();
}

int main() {
  DynArray obj1(5);
  obj1.show();
  obj1.toFile();
  obj1.fromFile();
  obj1.show();
  cin.get();
  return 0;
}
```

САМОСТІЙНА РОБОТА №6
ПЕРЕВАНТАЖЕННЯ ОПЕРАЦІЙ

Мета роботи

Навчитись використовувати перевантаження математичних операцій та операцій введення-виведення при розробці класів.

Завдання 6.1 Клас ComplexNumber

Створити клас для виконання операцій з комплексними числами. Зробити перевантаження символів операцій:

"+" – складання комплексних чисел;
"+" – складання комплексного та цілого числа;
"*" – множення комплексних чисел;
"*" – множення на число;
" << " , " >> " – консольне введення-виведення значень.

Лістинг 6.1.

```
#include <iostream>
#include <cmath>
#include <conio.h>
using namespace std;

class ComplexNumber {
int real, imaginary;

public:
ComplexNumber();
ComplexNumber(double r, double im);
ComplexNumber operator+(ComplexNumber &cn);
ComplexNumber operator*(ComplexNumber &cn);
ComplexNumber operator*(int number);
double abs();

friend ComplexNumber operator+(int number, ComplexNumber
&vcn);
friend ostream& operator<<(ostream &stream, ComplexNumber
&cn);
friend istream& operator>>(istream &stream, ComplexNumber
&cn);

};

ComplexNumber::ComplexNumber() {
  real = 0;
  imaginary = 0;
}

ComplexNumber::ComplexNumber(double r, double im) {
  real = r;
  imaginary = im;
}

ComplexNumber ComplexNumber::operator +(ComplexNumber
&cn) {
  real += cn.real;
```

```
  imaginary = cn.imaginary;
  return *this;
}

double ComplexNumber::abs() {
  double tmp = real*real + imaginary*imaginary;
  return tmp;
}

ComplexNumber ComplexNumber::operator *(ComplexNumber
&cn) {
  int old_real = real;
  int old_im = imaginary;
  real = old_real*cn.real - old_im*cn.imaginary;
  imaginary = old_real*cn.imaginary + old_im*cn.real;
  return *this;
}

ComplexNumber ComplexNumber::operator *(int number) {
  real *= number;
  imaginary *= number;
  return *this;
}

ComplexNumber operator+(int number, ComplexNumber &cn) {
  ComplexNumber tmp;
  tmp.real = cn.real + number;
  tmp.imaginary = cn.imaginary;
  return tmp;
}

ostream& operator<<(ostream &stream, ComplexNumber &cn) {
  stream << cn.real;
  if (cn.imaginary >= 0) stream << "+";
  stream << cn.imaginary << "i" << endl;
  return stream;
}
```

```
istream& operator>>(istream &stream, ComplexNumber &cn) {
  stream >> cn.real >> cn.imaginary;
  return stream;
}

int main() {
  ComplexNumber obj1(2, 7);
  ComplexNumber obj2, obj3;

  cout << "Input real and imaginary parts of complex number fo
obj2: ";
  cin >> obj2;

  cout << "obj2 = " << obj2 << endl;
  obj3 = obj1 + obj2;
  cout << "obj1 = " << obj1 << "obj3 = " << obj3 << endl;

  obj1 = obj3 * 2;
  cout << "obj1 = " << obj1 << endl;
  obj2 = obj1*obj3;
  cout << "obj1 = " << obj1 << endl;

  obj3 = 2 + obj2;
  cout << "obj3 = " << obj3 << endl;
  return 0;
}
```

Завдання 6.2 Перевантаження операторів new и delete

Лістинг 6.2.

```
#include <iostream>
#include <string>
#include  <stdlib>
void *operator new(size_t tip,int kol)       // глобальная функция
operator
{ cout << "глобальная функция NEW" <<endl;
  return new char[tip*kol];   // вызов системной функции  new
```

```
}
class cls
{   char a[40];
 public:
    cls(char *aa)
    {   cout<<"конструктор класса cls"<<endl;
       strcpy(a,aa);
    }
    ~cls(){}
    void *operator new(size_t,int);
    void operator delete(void *);
};
void *cls::operator new(size_t tip,int n)   // локальная функция
operator
{ cout << "локальная  функция " <<endl;
  return new char[tip*n];   // вызов системной функции  new
}
void cls::operator delete(void *p)  // локальная функция operator
{ cout << "локальная функция DELETE" <<endl;
  delete p;              // вызов системной функции  delete
}
void  operator  delete[](void  *p)              // глобальная функция
operator
{ cout << "глобальная функция DELETE" <<endl;
  delete p;              // вызов системной функции  delete
}
int main()
{ cls obj("перегрузка операторов NEW и  DELETE");
  float *ptr1;
  ptr1=new  (5)  float;       // вызов  глобальной  ф-ции  доопр.
оператора new
  delete [] ptr1;         // вызов глобальной ф-ции  доопр.оператора
delete
  cls *ptr2=new (10) cls("aa");  // вызов локальной функции
                    // доопределения оператора  new  (из класса cls)
  delete ptr2;             // вызов локальной функции доопределения
                           // оператора delete (из класса cls)
return 0;}
```

САМОСТІЙНА РОБОТА №7
ВІРТУАЛЬНІ ФУНКЦІЇ

Мета роботи

Навчитись використовувати віртуальних функцій при проектуванні спадкування в об'єктно-орієнтованих програмах

Завдання 7.1 Створення ієрархії класів

Наданий такий клас:

```
class Base
{ public:
virtual void myname() { cout << "This is class Base" << endl; }
};
```

Від цього класу треба створити два похідних класу: DerA і DerB.

Від класів DerA і DerB шляхом множинного успадкування створити клас DerAB.

Перевизначити у кожному з створених класів функцію myname таким чином, щоб вона виводила на екран дійсне ім'я класу об'єкту, для якого вона викликається.

Для кожного з класів створеної ієрархії створити по одному об'єкту і для кожного з них викличте функцію myname.

Створіть масив із 4 покажчиків на базовий клас ієрархії. Для кожного з них створіть динамічний об'єкт, по одному для кожного з класів ієрархії. Запишіть виклик функції myname для кожного з динамічних об'єктів за допомогою покажчиків.

Лістинг 7.1.

```
#include <iostream>
#include <conio.h>
using namespace std;
class Base
{
public:
virtual void myname() { cout << "This is class Base" << endl; }
};

class DerA: public Base
```

```
{
public:
virtual void myname() { cout << "This is class DerA" << endl; }
};

 class DerB: public Base
{
public:
virtual void myname() { cout << "This is class DerB" << endl; }
};

class DerAB: public DerA, public DerB
{
public:
virtual void myname() { cout << "This is class DerAB" << endl; }
};

int main()
{
cout<<"Test object call"<<endl;
Base Ob1; DerA Ob2;
DerB Ob3; DerAB Ob4;
Ob1.myname(); Ob2.myname(); Ob3.myname(); Ob4.myname();

Base *Arr[4];
Arr[0]=new Base; Arr[1]=new DerA;
Arr[2]=new DerB; Arr[3] = (DerA *)new DerAB;
Arr[0]->myname(); Arr[1]->myname();
Arr[2]->myname(); Arr[3]->myname();
return 0;
}
```

САМОСТІЙНА РОБОТА №8
ВВЕДЕННЯ В УЗАГАЛЬНЕНЕ ПРОГРАМУВАННЯ

Мета роботи

Навчитись використовувати шаблони при розробці програм.

Завдання 8.1 Використання деяких функцій класу *vector* бібліотеки STL

В наступній програмі демонструються деякі функції *vector*. В якості даних використовується простий тип *int*.

Лістинг 8.1.

```
#include <iostream>
#include <vector>
using namespace std;
vector<int> arr;
vector<int>::iterator it;
int main(){
// додати в масив 10 елементів зі значенням 8
arr.insert(arr.begin(),10,8);
// вивести кількість елементів в масиві
cout<<"size of arr="<<arr.size()<<endl;
// вивести елементи масиву для перевірки виходу за границі масиву потрібно
// використовувати метод at (визиває виключення),
// а не операцію індексації
cout<<"1. elements of arr: ";
for(it = arr.begin();it != arr.end();++it)
cout<<*it<<' ';// виведення елементів з використанням ітератора
cout<<endl;
// в методі at на відмінну від операції []
// відбувається виключення при виході за границі масиву
arr.at(4)=10; // змінюємо 4 елемент
arr.at(5)=11; // змінюємо 5 елемент
cout<<"2. elements of arr: ";
for(it = arr.begin();it != arr.end();++it)
cout<<*it<<' ';// виведення елементів з використанням ітератора
cout<<endl;
//видаляємо з 0 по 4 елемент
arr.erase(arr.begin(),arr.begin()+4);
cout<<"3. elements of arr: ";
for(unsigned int i=0;i<arr.size();i++)
cout<<arr[i]<<" ";// звичайне виведення елементів
```

```
cout<<endl;
// видалити усі елементи та вивести кількість елементів для перевірки
arr.clear();
cout<<"size of arr="<<arr.size()<<endl;
// добавити в кінець один елемент зі значенням 9
arr.push_back(9);
cout<<"arr[0]="<<arr[0]<<endl;
return 0;
}
```

Завдання 8.2 Приклад використання класу *vector* бібліотеки STL

Створити клас для роботи з трьох вимірними векторами. Перевантажити операції введення та виведення, операції “>”, ”==”, ”+”використовувати контейнерний клас *vector*.

Лістинг 8.2.

```
#include <iostream>
#include <vector>
using namespace std;
class vector3d{
int x,y,z;
public:
vector3d() {x=y=z=0;}
vector3d(int a, int b, int c){x=a; y=b; z=c;}
vector3d &operator+(int a);
friend ostream &operator<<(ostream &stream, vector3d obj);
friend bool operator <( vector3d a, vector3d b);
friend bool operator==( vector3d a, vector3d b);
};

vector3d &vector3d::operator+(int a)
{ x+=a; y+=a; z+=a;
return *this; }

ostream &operator <<(ostream &stream, vector3d obj)
{ stream<<obj.x<<" "<<obj.y<<" "<<obj.z<<"\n";
```

```
return stream; }

bool operator< (vector3d a, vector3d b)
{return (a.x+a.y+a.z)<(b.x+b.y+b.z); }

bool operator== (vector3d a, vector3d b)
{return (a.x+a.y+a.z)==(b.x+b.y+b.z);}

int main () {
vector<vector3d> v;
unsigned int i;
for (i=0;i<10;i++)
v.push_back(vector3d(i,i+2,i+3));
for (i=0; i<v.size(); i++)
cout<<v[i];
cout<<endl;
for (i=0; i<v.size(); i++)
v[i]=v[i]+10;
for (i=0; i<v.size(); i++)
cout<<v[i];
return 0;
}
```

Завдання 8.3 Використання деяких функцій класу *list* бібліотеки STL

В наступній програмі демонструється деякі функцій класу *list*. В якості даних класу використовується простий тип *int*.

Лістинг 8.3.

```
#include <iostream>
#include <list>
#include <algorithm>
#include <iterator>
using namespace std;
template <class T> // прототип для шаблону функції printList
void printList(const list<T> &listRef);
int main()
```

```
{
const int SIZE=4;
int array[SIZE]={2,4,6,8};
list <int> l1, l2; // створити список для int
// вставити елементи в l1
l1.push_front(1);          l1.push_front(2);
l1.push_back(4);           l1.push_back(3);
cout<<"l1 contains:";      printList(l1);
l1.sort(); // сортувати значення
cout<<"\nl1 after sorting contains: ";      printList(l1);
// вставити в l2 елементи array
l2.insert(l2.begin(), array, array + SIZE );
cout<<"\nAfter insert, l2 contains: ";      printList(l2);
// видалити елементи l2 та вставити в кінець l1
l1.splice(l1.end(),l2);
cout<<"\nAfter splice, l1 contains: ";      printList(l1);
l1.sort(); //  сортувати значення
cout<<"\nAfter sort, l1 contains: ";                printList(l1) ;
// вставити в l2 елементи array
l2.insert(l2.begin(), array, array + SIZE);
l2.sort();
cout<<"\nAfter insert, l2 contains: ";      printList(l2);
// видалити елементи l2 та вставити в кінець l1, виконуючи
сортировку
l1.merge(l2);
cout<<"\nAfter merge: \nl1 contains: ";   printList(l1);
cout<<"\nl2 contains: ";   printList(l2);
l1.pop_front(); // видалити елемент з початку
l1.pop_back(); // видалити елемент з кінця
cout<<"\nAfter pop_front and pop_back:\nl1 contains: ";
printList(l1);
l1.unique(); // видалити дублікати
cout<<"\nAfter unique, l1 contains: ";      printList(l1);
// виконати обмін елементами l1 та l2
l1.swap(l2);
cout<<"\nAfter swap: \nl1 contains: ";    printList(l1);
cout<<" \nl2 contains: ";          printList(l2);
// замінити зміст l1 елементами l2
```

```
l1.assign(l2.begin(), l2.end());
cout<<"\nAfter assign, l1 contains: ";       printList(l2);
// видалити елементи l2 и вставити l1, виконуючи сортировку
l1.merge(l2);
cout<<"\nAfter merge, l1 contains: ";       printList(l1);
l1.remove(4); //видалити усі цифри чотири
cout<<"\nAfter remove(4), l1 contains: "; printList (l1);
cout<<endl;
return 0; }
// визначення шаблону функції printList; для виведення елементів списку
template  <class  T>
void  printList(const list <T> &listRef) {
if  (listRef.empty())         cout<<"List is empty";
else { list <T>::const_iterator p=listRef.begin();
        for (; p!=listRef.end(); ++p) cout<<*p<<' '; }
}
```

Завдання 8.4 Приклад використання класу *list* бібліотеки STL

Книжки характеризуються наступною інформацією: назва та автор. Використати клас *list* бібліотеки STL для зберігання каталогу книжок. Забезпечити операції введення-виведення інформації про книжки, додавання книжки в каталог.

Лістинг 8.4.

```
#include <iostream>
#include <string>
#include <list>
using namespace std;
class book{
  char name[30];
  char author[20];
  public:
  book() {strcpy(name,"");strcpy(author,"");}
  book(char*                    name1,char*                    author1)
{strcpy(name,name1);strcpy(author,author1);}
  friend book input_book();
  friend void show_books(const list<book> &books);
```

```
};

book input_book() {
  book tmp;
  cout<<"Book name: "; cin>>tmp.name;
  cout<<"Author: "; cin>>tmp.author;
  return tmp;}

void show_books(const list<book> &list_books)
{ list <book>::const_iterator p = list_books.begin();
  if (p == list_books.end()) { cout << "No books!!!" << endl;
return; }
  cout<<"Show catalog book"<<endl;
  for (; p!= list_books.end(); ++p) {
            cout<<"Book name: "<<p->name<<endl;
            cout<<"Author: "<<p->author<<endl; }
}

int main()
{
  ist <book> books;
 unsigned int i,n;
 cout<<"Input number book"<<endl;  cin>>n;
 for (i=0;i<n;i++)  books.push_back(input_book());
 show_books(books);
return 0;
}
```

Завдання 8.5 Використання деяких функцій класу *stack* бібліотеки STL

В наступній програмі демонструється декілька функцій класу *stack*. В якості даних класу використовується простий тип *int*, контейнерний класс *vector<int>*, контейнерний класс *list<int>*.

Лістинг 8.5.

```
#include <iostream>
#include <stack>
```

```
#include <vector>
#include <list>
using namespace std;
template <class T>
void show_el(T &s);

int main()
{
  stack<int> s1;
  stack<int,vector<int>> s2;
  stack<int,list<int>> s3;
  for (unsigned int i=0; i<10; ++i)
  {          s1.push(i);        s2.push(i*i);      s3.push(i*i*i);    }
  cout<<"Show elements type int"<<endl;
show_el(s1);
  cout<<"Show       elements       type       vector       <int>"<<endl;
show_el(s2);
  cout<<"Show elements type list <int>"<<endl;   show_el(s3);
return 0;
}

template <class T>
void show_el(T &s)
{
  if (s.empty()) {   cout << "No elements!!!" << endl;          return; }
  while(!s.empty()){ cout<<s.top()<<' ';     s.pop(); }
  cout<<endl;
}
```

Завдання 8.6 Приклад використання класу *stack* бібліотеки STL

Книжки характеризуються наступною інформацією: назва та автор. Використати клас *stack* бібліотеки STL для зберігання каталогу книжок. Забезпечити операції введення-виведення інформації про книжки, додавання книжки в каталог.

Лістинг 8.6.

```
#include <iostream>
```

```
#include <string>
#include <stack>
using namespace std;
class book
{  char name[30];
  char author[20];
  public:
  book() {strcpy(name,"");strcpy(author,"");}
  book(char*                name1,char*                author1)
{strcpy(name,name1);strcpy(author,author1);}
  friend ostream &operator<<(ostream &stream, book obj);
  friend void show_books(stack<book> &books);
  friend book input_book();
};

ostream &operator<<(ostream &stream, book obj)
{
  stream<<"Book        name:        "<<obj.name<<"        Author:
"<<obj.author<<endl;
  return stream;
}
book input_book()
{
  book tmp;
  cout<<"Book name: ";   cin>>tmp.name;
  cout<<"Author: ";      cin>>tmp.author;
  return tmp;
}

void show_books(stack<book> &s_books)
{
  if (s_books.empty())
  {       cout << "No books!!!" << endl;   return;  }
  cout<<"Show catalog book"<<endl;
  while(!s_books.empty())
  {       cout<<s_books.top();    s_books.pop();  }
}
```

```
int main()
{
 stack <book> books;
 unsigned int i,n;
 cout<<"Input number book"<<endl;
 cin>>n;
 for (i=0;i<n;i++)  books.push(input_book());
 show_books(books);
return 0;
}
```

Завдання 8.7 Використання деяких функцій класу *queue* бібліотеки STL

В наступній програмі демонструється декілька функцій класу *queue*. В якості даних класу використовується простий тип *double* та контейнерний класс *list<double>*.

Лістинг 8.7.

```
#include <iostream>
#include <queue>
#include <list>
using namespace std;
template <class T>
void show_el(T &q);

int main()
{
  queue<double> q1;
  queue<double,list<double>> q2;
  for (unsigned int i=0; i<10; ++i)
  {          q1.push(i*3.2+i);          q2.push(i*i+9.8); }
  cout<<"Show elements type double"<<endl;
  show_el(q1);
  cout<<"Show elements type list <double>"<<endl;
  show_el(q2);
return 0;
}
```

```
template <class T>
void show_el(T &q)
{
  if (q.empty())  {        cout  <<  "No  elements!!!"  <<  endl;
return;   }
  while(!q.empty())       {        cout<<q.front()<<' ';     q.pop();}
 cout<<endl;
}
```

Завдання 8.8 Приклад використання класу *queue* бібліотеки STL

Книжки характеризуються наступною інформацією: назва та автор. Використати клас *queue* бібліотеки STL для зберігання каталогу книжок. Забезпечити операції введення-виведення інформації про книжки, додавання книжки в каталог.

Лістинг 8.8.

```
#include <iostream>
#include <string>
#include <queue>
using namespace std;
class book
{
  char name[30];
  char author[20];
  public:
  book() {strcpy(name,"");strcpy(author,"");}
  book(char*                    name1,char*                    author1)
{strcpy(name,name1);strcpy(author,author1);}
  friend ostream &operator<<(ostream &stream, book obj);
  friend void show_books(queue<book> &books);
  friend book input_book();
};

ostream &operator<<(ostream &stream, book obj)
{
```

```
    stream<<"Book name: "<<obj.name<<" Author: "<<obj.author<<endl;
     return stream;
}

book input_book()
{
  book tmp;
  cout<<"Book name: ";   cin>>tmp.name;
  cout<<"Author: ";      cin>>tmp.author;
  return tmp;
}

void show_books(queue<book> &q_books)
{
  if (q_books.empty())  { cout << "No books!!!" << endl; return;  }
  cout<<"Show catalog book"<<endl;
  while(!q_books.empty())  { cout<<q_books.front(); q_books.pop();  }
}

int main()
{
 queue<book> books;
 unsigned int i,n;
 cout<<"Input number book"<<endl;  cin>>n;
 for (i=0;i<n;i++)  books.push(input_book());
 show_books(books);
 return 0;
}
```

Завдання 8.9 Приклад використання класу *map* бібліотеки STL

Програма для підрахунку кількості слів у тексті та виведення частоти зустрічі слів у тексті у відсотках. Необхідно створити файл in.txt, в якому міститься текст.

Лістинг 8.9.

```
#include <iostream>
#include <string>
#include <map>
#include <fstream>
using namespace std;

int main()
{

map <string,int> words;
ifstream in;
in.open("in.txt");
string word;
while (in>>word)
words[word]++;
ofstream out;
out.open("out.txt");
int count=0;
map <string,int>::iterator p;
out<<"Words count:"<<endl;
for (p =words.begin();p!=words.end();p ++)
{out<<(*p).first<<": "<<(*p).second<<endl;count+=(*p).second;}
out<<"Words %:"<<endl;

for (p=words.begin();p!=words.end();p++)
out<<(*p).first<<":
"<<(float)((float)(*p).second/(float)count)*100<<"%"<<endl;
return 0;
}
```

Завдання 8.10 Приклад використання класу *map* бібліотеки STL

Програма створення тлумачного словнику

Лістинг 8.10.

```
#include <iostream>
#include <map>
```

```
#include <cstring>
using namespace std;
class word {
  char str[20];
public:
  word() { strcpy(str, ""); }
  word(char *s) { strcpy(str, s); }
  char *get() { return str; }
};

// must define less than relative to word objects
bool operator<(word a, word b)
{
   return strcmp(a.get(), b.get()) < 0;
}

class meaning {
  char str[80];
public:
  meaning() { strcpy(str, "");}
  meaning(char *s) { strcpy(str, s); }
  char *get() { return str; }
};

int main()
{
  map<word, meaning> dictionary;
 // вставка в відображення об'єкти класів  words и meanings
  dictionary.insert(pair<word, meaning>(word("house"),
                meaning("A place of dwelling.")));
  dictionary.insert(pair<word, meaning>(word("keyboard"),
                meaning("An input device.")));
  dictionary.insert(pair<word, meaning>(word("programming"),
          meaning("The act of writing a program.")));
  dictionary.insert(pair<word, meaning>(word("STL"),
                meaning("Standard Template Library")));
  // введення слова та знаходження його значення
  char str[80];
```

```
 cout << "Enter word: ";
 cin >> str;
 map<word, meaning>::iterator p;
 p = dictionary.find(word(str));
 if(p != dictionary.end())
  cout << "Definition: " <<  p->second.get();
 else
  cout << "Word not in dictionary.\n";
 return 0;
}
```

Завдання 8.11 Приклад використання класу *set* бібліотеки STL

Две коллекции

Близнецы Саша и Паша собирают коллекцию марок, причем каждый собирает свою коллекцию. Близнецы - они всегда стараются иметь все в одинаковом количестве, поэтому в их коллекциях всегда равное количество марок. Поддерживать такое равновесие очень сложно, поэтому иногда Саше или Пете приходится покупать марки, которые уже есть в их коллекции. А что, спрашивается, делать, например, Саше, если у Пети появилась новая редкая марка? Вторую такую не достать, купить другой раритет сложно... Вот и приходится покупать первую попавшуюся!

Интересно, а сколько марок в коллекциях Пети и Саши совпадают? Причем интересно это количество без учета повторов! Напишите такую программу для Саши и Пети. Может быть, посмотрев на результаты, они станут собирать одну общую коллекцию?

В первой строке содержится одно целое число N - количество марок в одной коллекции. Далее две строки содержат по N целых чисел - индексы марок в каталоге, которым пользуются Петя и Саша. Известно, что индексы - это целые положительные числа, не превышающие 1000000, а количество марок $N \leq 10000$.

Выведите количество совпадающих индексов марок в коллекциях. Учтите, что учитываются только разные индексы марок.

Лістинг 8.11.

```
#include <iostream>
#include <set>
#include <algorithm>
using namespace std;

int n; int x;
set<int> a;
set<int> b;
set<int>::iterator it;

int main()
{
  cin>>n;
  for(int i=0;i<n;++i)
  {
     cin>>x;
     a.insert(x);
  }
  for(int i=0;i<n;++i)
  {
     cin>>x;
     b.insert(x);
  }
  int c=0;
  for(it = a.begin();it!= a.end();++it)
     {
        if  (b.find(*it)!=b.end()) c++;
     }
  cout<<c<<endl;

return 0;
}
```

Пример входных данных	Результат
6 1 5 7 5 7 10 1 7 5 5 3 5	3

САМОСТІЙНА РОБОТА №9
ОБРОБКА ВИНЯТКОВИХ СИТУАЦІЙ

Мета роботи

Навчитись обробляти виняткові ситуації засобами мови С++.

Завдання 9.1 Клас Exception

Для завдання 3.1 створити клас Exception, що дозволяє обробляти виняткові ситуації; помилки при відкритті файлу, помилки арифметичних операцій (ділення на 0), помилки при виділенні динамічної пам'яті.

Лістинг 9.1.

```
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;

class Exception{
public:
  Exception(int error_code);
};
Exception::Exception(int error_code)
{
switch(error_code){
  case 1: cout<<"Error in division 0 \n"; break;
  case 2: cout<<"Error allocation of dynamic memory\n"; break;
  case 3: cout<<"Error couldn't open file\n"; break;
  default: break;
  }
};

class DynArray
{
  unsigned int *arr;
  int n;
```

```
public:
  DynArray(int count);
  DynArray(const DynArray &obj);
  ~DynArray();

  void show();
   void toFile();
  void fromFile();
  DynArray& append(const DynArray &obj);
};


DynArray::DynArray(int count)
{
  n = count;
   try
  {
            arr = new unsigned int[n];
            if (!arr) throw Exception(2);
  }
  catch(...){}
            for (int i=0; i<n; i++)
  try {
            if (i!=0) arr[i] =2*i+1/i;
                      else { arr[i] = i; throw Exception(1);}
            }
  catch(...){}
  }

DynArray::DynArray(const DynArray &obj)
{
  n = obj.n;
  try
  {
            arr = new unsigned int[n];
            if (!arr) throw Exception(2);
  }
```

```
  catch(...){}
  for (int i=0; i<n; i++)
            arr[i] = obj.arr[i];

}

DynArray::~DynArray()
{
  //delete [] arr;
}

void DynArray::show()
{
  for (int i=0; i<n; i++)
  cout << arr[i] << " ";
  cout << endl;
}

DynArray& DynArray::append(const DynArray &obj)
{
  unsigned int *tmp;
  try
  {
            tmp = new unsigned int[n + obj.n];
            if(!tmp) throw Exception(2); //throw std::bad_alloc();?
  }
  catch(...){}
  int i;
  for (i=0; i<n; i++)
            tmp[i] = arr[i];
  for (i=0; i<obj.n; i++)
            tmp[n+i] = obj.arr[i];
  delete [] arr;
  arr = tmp;
  n += obj.n;
  return *this;
}
```

```
void DynArray::toFile() {
  ofstream ofs("1.txt");
  if (ofs.is_open()) {
            ofs << n << " ";
            for (int i=0; i<n; i++)
                      ofs << arr[i] << " ";
  }
  ofs.close();
}

void DynArray::fromFile() {
  ifstream ifs;
  ifs.open("1.txt");
  if (ifs.fail()) throw Exception(3);
  else {
            ifs >> n;
            arr = new unsigned int[n];
            for (int i=0; i<n; i++)
                      ifs >> arr[i];
  }
  ifs.close();
}

int main() {
  DynArray obj1(5);
  DynArray obj2(3);
  obj1.show();
  obj2.show();

  obj1.append(obj2);
  obj1.toFile();
  obj1.fromFile();
  obj1.show();
  cin.get();
  return 0;
}
```

САМОСТІЙНА РОБОТА №10
РОБОТА З QDEBUG В QT

Мета роботи

Навчитись виконувати тестування програм за допомогою класу QDebug, а також використовуючи вбудований відладчик в Qt Creator.

Теоретичні відомості

Під час розробки програм часто виникають проблеми з виявленням помилок (bug, помилка або баг). Розробникам доводиться тратити чималу частину робочого часу на те, щоб знайти і усунути їх. До засобів , що допомагають знизити їх кількість, можна віднести:

− надання початкового коду для перегляду іншими розробниками (code review);

− створення класів для автоматизованих тестів.

Помилки можна мінімізувати, але, в кожному разі, повністю їх уникнути не вдасться, і якщо у вашу програму раптом закрався підступний баг, то найпершим засобом, що допомагає у нелегкій праці його пошуку, буде *відладчик*. Роль відладчика полягає в наданні оболонки, в якій можна відслідковувати зміну даних під час виконання програми, завдяки чому можна дізнатися, чому створена вами програма веде себе не так, як ви це задумували.

Завдяки платформонезалежності Qt розробник може для налагодження своїх програм використовувати будь-який з вподобаних відладчиків, наприклад GDB або відладчик, вбудований в Microsoft Visual Studio. Якщо ви ще серйозно не стикалися з процесом налагодження програм, то рекомендую почати з відладчика без графічного користувача інтерфейсу, оскільки подача команд в діалоговому режимі допоможе вам зрозуміти роботу відладчика як такого і в майбутньому гідно оцінити відладчики, що володіють графічним інтерфейсом. Тому докладніше розберемо відладчік GDB, доступний як для Windows, так для Linux і Mac OS X.

Відладчик GDB (GNU Debugger). GDB − це самий звичайний засіб для налагодження програм в ОС Unix. Робота з цим відладчиком зазвичай здійснюється з командного рядка, хоча можна скористатися і оболонками, які надають можливість роботи з цим відладчиком в інтерактивному режимі, ось деякі з них: XXGDB, DDD і KDBG.

Мабуть, найбільш ідеальним середовищем для роботи з відладчиком в інтерактивному режимі є IDE, в цьому випадку все необхідне знаходиться "під рукою". Якщо ви не збираєтеся працювати з відладчиком безпосередньо і віддасте перевагу використовувати IDE, то подальше опис GDB можете пропустити. Але якщо ви хочете розібратися в процесі налагодження, то давайте створимо програму, свідомо містить проблемний код.

Лістинг 10.1.

```
void bug()
{
  int n = 3;
  int* pn = &n;
  // помилка!
  delete pn;
}
int main()
{
  bug();
  return 0;
}
```

В листінге з функції main () здійснюється виклик функції bug (), в якій створюється і ініціалізується змінна n, а після присвоєння вказівника pn її адреси виклик оператора delete виконується спроба вивільнення пам'яті, що використовується зміною n. Оскільки пам'ять не була виділена динамічно, ця операція неминуче приведе до помилки.

Відкомпілюємо програму з параметром -g, щоб у виконуваний файл було включено інформацію, необхідну для відладчика:

```
g++ -g bug.cpp –o bug
```

В даному випадку не був створений традиційний для Qt pro-файл та був виконаний виклик компілятора напряму. У випадках же з pro-файлами Qt, для того, щоб отримати виконуваний файл із включеною відладочною інформацією, змінна pro-файла CONFIG

повинна містити значення debug або debug_and_release.

Відладчик можна запустити наступним чином, вказавши назву програми, призначеної для налагодження:

gdb bug.exe

Окрім назви програми, в GDB можна додатково передавати і core-файл, згенерований операційною системою після аварійного завершення програми. Це дуже зручно, так як можна не завантажувати програму на виконання в відладчик, а знайти проблемне місце за допомогою core-файлу. На жаль, ОС Windows не генерує подібних файлів, тому надалі будуть описані прийоми роботи з відладчиком GDB, що застосовуються на обох ОС (Windows та Linux).

Після цього відладчик відобразить рядок запрошення такого вигляду:

(gdb)

Отже, давайте запустимо саму програму під відладчиком. Для цього потрібно ввести команду run:

(gdb) run

В результаті ми отримаємо повідомлення, що сигналізують про некоректне завершення програми. Для того щоб розібратися у проблемі, потрібно переглянути стек програми. Це робиться за допомогою команди where:

(gdb) where

Виведення відладчика буде приблизно таким:

#0 0x77f767ce in _libmsvcrt_a_iname () ...
#6 0x00401305 in operator delete(void*) ()
#7 0x004012ae in bug() () at bug.cpp:5
#8 0x004012df in main () at bug.cpp:10

Зауважте, що функція main() викликала у десятому рядку

функцію bug(),а виклик п'ятого рядка цієї функції створив проблему. За допомогою команди up можна піднятися по стеку програми на певну кількість рівнів. Давайте піднімемося на один рівень, це буде відповідати функції bug():

```
(gdb) up 1
```

Відладчик покаже наступне:

```
#7 0x004012ae in bug() () at bug.cpp:5
5 delete pn;
```

Для того щоб дізнатися значення будь-якої локальної змінної функції, потрібно подати команду print. Давайте виконаємо це для змінної n:

```
(gdb) print n
```

У відповідь відладчик покаже її значення:

```
$1 = 3
```

Встановлення контрольних точок (break points) здійснюється у відладчику за допомогою команди break. Встановимо нашу точку у функції bug() та виконаємо перезапуск нашої програми:

```
(gdb) break bug
(gdb) run
```

Відладчик зупиниться на заданій нами контрольній точці та покаже наступне:

```
Breakpoint 1, bug() () at bug.cpp:3
3 int n = 3;
```

Для того щоб перейти на наступний рядок, скористаємося командою next. Ця команда виконує код порядково без переходу всередину тіла функції:

```
(gdb) next
4 int* pn = &n;
```

З цього видно, що налагоджувач перейшов з третього рядка на четвертий. Якщо знадобиться виконати рядки коду, включаючи рядок всередині тіла функції, то для цього потрібно було б скористатися командою step. Наприклад, можна встановити контрольну точку у функції main()і за допомогою команди step увійти всередину функції bug(). В таблиці 10.1 зведені найбільш часто використовувані команди відладчика.

Таблиця 10.1 – Деякі команди відладчика GDB

Команда	Опис
1	2
quit	Вихід з відладчика
help	Виведення довідкової інформації. Якщо додатковим параметром вказана яка-небудь команда, то виводиться повна довідкова інформація по цій команді
run	Запуск програми
attach	Приєднання відладчика до запущеного процесу зі зазначеним ідентифікатором
detach	Від'єднання відладчика від приєднаного процесу
break	Встановлення контрольної точки. Виклик команди без параметра встановить точку наступної виконуваної інструкції. Як параметр можна передавати назва функції, номер рядку і зсув. Якщо потрібно, можна вказати назву конкретного вихідного файлу у вигляді <назва файлу>: <номер рядка> або<назва файлу>: <назва функції>
tbreak	Аналогічна команді break з тією лише різницею, що контрольна точка буде видалена після її досягнення

Продовження табл. 10.1

1	2
clear	Видалення контрольної точки. Виклик команди без параметра видалить контрольну точку наступної виконуваної інструкції. Як параметр можна передавати назву функції, номер рядку та зсув. Якщо потрібно, то можна вказати назву конкретного вихідного файлу у вигляді <назву файлу>: <номер рядку> або <назва файлу>: <назва функції>
delete	Видалення всіх контрольних точок
disable	Відключення всіх контрольних точок
enable	Включення всіх контрольних точок
continue	Виконання наступного рядка вихідного коду програми. Додатковим параметром можна задати кількість виконуваних рядків
next	Виконання наступного рядка вихідного коду програми. Додатковим параметром можна задати кількість виконуваних рядків
step	Виконання наступного рядка вихідного коду програми. Додатковим параметром можна задати кількість виконуваних рядків. На відміну від next, при виконанні функції відбувається вхід в неї і зупинка
until	Продовження виконання програми до виходу з функції

Інші методи налагодження. Одним із стандартних прийомів налагодження є вставка в вихідний код операторів виведення, що дозволяє побачити значення змінних і порівняти їх з очікуваними значеннями. Такий спосіб налагодження часто використовується розробниками, оскільки нічого не варто помістити ці оператори або оформити їх у вигляді окремого дамп-методу.

В Qt прикладом такого підходу є метод QObject::dumpObjectInfo(), який виводить на екран метаінформацію об'єкта.

Qt надає макроси та функції для налагодження, за допомогою яких можна вбудовувати в саму програму різного роду перевірки і

виведення тестових повідомлень.

В заголовочному файлі QtGlobal містяться визначення двох макросів Q_ASSERT () і Q_CHECK_PTR ():

– Q_ASSERT () приймає в якості аргументу значення бульового типу та виводить попередження, якщо це значення не дорівнює true;

– Q_CHECK_PTR () приймає покажчик та виводить попередження, якщо переданий покажчик дорівнює 0, а це означає, що або покажчик ні ініціалізований, або операція по виділенню пам'яті пройшла невдало.

Qt надає глобальні функції qDebug (), qWarning () і qFatal (), які також визначені в заголовочному файлі QtGlobal. Їх застосування подібне на функцію printf ().

Як і в printf (), в ці функції передаються форматований рядок та різні параметри. У Microsoft Visual Studio виведення цих функцій виконується в вікно відладчика, а в ОС Linux – в стандартний потік виведення помилок. Виклик функції qFatal() після виведення повідомлення відразу завершує роботу всієї програми.

Якщо буде потрібно перенаправити потік виведення повідомлення, потрібно створити та встановити свою власну функцію для управління виведенням. Встановлюється вона за допомогою функції qInstallMsgHandler(). Цій функції як аргумент передається адреса на функцію, керуючу повідомленнями і має наступний прототип:

```
void fct(QtMsgType type, const char *msg);
```

На місці fct повинно стояти назва функції. Перший аргумент являє собою тип повідомлення, що приймає одне із значень перерахування QtMsgType: QtDebugMsg, QtWarningMsg або QtFatalMsg. Другий аргумент – це покажчик на саме повідомлення.

Для полегшення процесу налагодження рекомендується привласнювати всім об'єктам назви. Таким чином, ці об'єкти можна буде завжди знайти, викликавши метод Object::objectName(). Це дозволить у процесі роботи програми скористатися методом QObject::dumpObjectInfo(), що дозволяє відобразити внутрішню інформацію об'єкта.

Також для налагодження можна скористатися установкою фільтра подій для об'єкта класу QCoreApplication, у цьому випадку

даний фільтр буде найперший об'єктом, який отримує та обробляє події всіх об'єктів за стосунку.

Найпростіший спосіб операції виведення у Qt – це використання об'єкта класу QDebug. Цей об'єкт дуже нагадує стандартний об'єкт потоку виведення у C++ cout.

Наприклад, вивести повідомлення у відладчику або в консолі за допомогою функції qDebug() можна наступним чином:

```
qDebug() << "Test";
```

Ця функція створює об'єкт класу потоку QDebug, передаючи у його конструктор згаданий раніше аргумент QtDebugMsg. Можна було б, звичайно, поступити і так:

```
QDebug(QtDebugMsg) << "Test";
```

Але, як ви бачите, попередній рядок виглядає більш компактно, тому рекомендую користуватися саме цим рядком.

Важливо розуміти, що виведення інформації за допомогою функції qDebug() відбувається для налагоджувальних та релізних схем. Якщо виведення інформації повино бути присутнє тільки у налагоджувальній версії програми, а у релізної версії воно повино бути відсутнє, то можна реалізувати макрос подібному наступному:

```
#if defined(QT_DEBUG)
  #define QDEBUG(X) qDebug() << X;
#else
  #define QDEBUG(X) ;
#endif
```

Та використовувати замість функції qDebug() наступний запис:

```
QDEBUG("Test1" << 123 << "Test2" << 456);
```

Порядок виконання лабораторної роботи

1. Ознайомитися з теоретичними відомостями щодо роботою в Qt.

2. Виконати тестування програм з попередніх самостійних робіт за допомогою класу QDebug, а також використовуючи вбудований відладчик в Qt Creator.

3. Виконати аналіз отриманих результатів.

ЛІТЕРАТУРА

1. Язык программирования С++: Учебный курс [Текст] / С.В. Глушков, А.В. Коваль, С.В. Смирнов. – Харьков: Фолио, М.: Изд. АСТ, 2001. – 500 с.

2. Дейтел, Х.М. Как программировать на С++.[Текст] / Х.М. Дейтел, П.Дж. Дейтел – М.: ЗАО "Изд. БИНОМ", 2001. – 1152 с.

3. Павловская, Т.А. С/С++. Программирование на языке высокого уровня. [Текст] / Т.А. Павловская – СПб.: Питер, 2003. – 461 с.

4. Павловская, Т.А. Структурное программирование: Учеб. пособие [Текст] / Т.А. Павловская, Ю.А. Щупак. – СПб.: Питер, 2002. – 240 с.

5. Топп, У. Структуры данных в С++. [Текст] / Топп, У., Форд У. – М.: БИНОМ, 1999. – 816 с.

6. Фридман, А.Л. Основы ООП на языке С++. [Текст] / А.Л. Фридман – М.: Горячая линия, – Телеком, 2001. – 232 с.

7. Шилдт, Г. Искусство программирования на С++. [Текст] / Г. Шилдт. – СПб.: БХВ – Петербург, 2006. – 496 с.

8. Лафоре, Р. Объектно-ориентированное программирование в С++. [Текст] / Р. Лафоре. – СПб.: Питер, 2003. – 928 с.

9. Лесневский, А.С. Объектно-ориентированное программирование для начинающих [Текст] / А.С. Лесневский. – М: Бином, 2005. – 232 с.

10. Савитч, У. Язык С++. Курс объектно-ориентированного программирования. [Текст] / Савитч, У. – М.: "Вильямс", 2001. – 704 с.

11. Шаммас, К. Основы С++ и объектно-ориентированного программирования [Текст] / К. Шаммас. – Диалектика, 1996. – 448 с.

12. Кормен, Т. Алгоритмы: построение и анализ / [Текст] Т. Кормен, Ч. Лейзерсан, Р. Ривест. – М.: МУНМО, 2001. – 960 с.

13. Кубенский, А.А. Структуры и алгоритмы обработки данных: объектно-ориентироанный подход и реализация на С++ [Текст] /

А.А. Кубенский. – СПб: БХВ-Петербург, 2004. – 464 с.

14. Элдтер, Дж. С++: Библиотека программиста [Текст] / Дж. Элдтер. – СПб.: Питер, 2000. – 320 с.

15. Браунси Кэн Основные концепции структур данных и реализация в С++ [Текст] / Кэн Браунси. – М.: Изд. Дом «Вильямс», 2002. – 320 с.