Лабораторна робота № 3. Класи, члени класу, робота з класами

Мета роботи: – вивчення класів, конструкторів і деструкторів, дружніх функцій і дружніх класів, вбудованих функцій, статичних і константних членів класу.

Теоретичний матеріал лекції, [1, розділ 2, 3], [2, розділ 12], [3, розділ . 10, 11, 12].

1. Короткі теоретичні відомості

Дані класу можуть бути:

- звичайними, без кваліфікаторів (`int x`);
- статичними (`static int x`);
- константними (`const int x`).

Статичні поля оголошуються за допомогою слова `static` і мають бути визначені поза класом на глобальному рівні оголошень (незалежно від їх розміщення у класі). Статичні дані зберігаються разом з глобальними, окремо від об'єкта класу.

Статичні дані є спільними для всіх об'єктів цього класу та успадкованих від нього об'єктів. За допомогою статичних полів можна передавати інформацію всім об'єктам цього класу. Статичні поля існують навіть тоді, коли не оголошений жоден об'єкт цього класу.

```
class A {
public:
static int x;     // оголошення статичного елемента даних
          int y;
};
int A::x=5;       // визначення статичного елемента даних
```

Статичні поля можуть бути ініціалізовані також конструктором у його тілі (всередині блоку {}), але не у списку ініціалізації (після символу ':'). Особливістю такого способу ініціалізації є те, що при кожному створенні нового об'єкта змінюватимуться статичні поля у всіх об'єктів класу.

Константні дані оголошуються за допомогою слова `const`. Ініціалізуються константні дані після двокрапки у списку ініціалізації конструктора. Константні дані не можна змінити у програмі. У класі також можна визначити символьні константи перелічимого типу:

```
class A {
enum (DAY=30, MOHTH=3, YEAR=2015};
…
};
```

1.1. Методи класу

Методи (функції-члени), оголошені у протоколі класу, можуть бути:

- членами класу;
- друзями класу.

Методи та друзі мають доступ до усіх частин класу. Методи допускають перевантаження. Перевантажені методи мають однакові імена, але повинні відрізнятися кількістю параметрів або їх типами. Методи викликаються у межах класу безпосередньо, за їх іменами. Поза класом вони викликаються за допомогою об'єкта класу, посилання або вказівника на клас.

Якщо метод повертає об'єкт, посилання або вказівник на об'єкт, то можливий ланцюжковий виклик методів.

Кожний нестатичний метод класу має доступ до об'єкта, з якого він викликаний, через вказівник `this`. За допомогою цього вказівника можна отримати доступ до всіх елементів класу(даних та методів):

```
class Base {
int x;
public:
```

```
void print () { cout << x << ' ' << (*this).x << ' ' << this->x << endl; }
};
```

Розіменований вказівник `this` використовується для повернення методом класу посилання на об'єкт свого класу:

```
class Base {
int x;
public:
Base & Copy(const Base& a) { x=a.x; return *this; }
};
```

1.2. Статичні та константні методи

Статичний метод оголошується за допомогою слова `static`, яке записується перед заголовком функції. Статичний метод може бути викликаний незалежно від існування об'єкта класу. У цьому випадку він викликається з іменем класу, наприклад `Base::f1()` з врахуванням рівнів захисту елементів класу.

Статичному методу не передається вказівник `this`, тому він не може звертатися до нестатичних елементів класу безпосередньо, за їх іменами. Однак, статичний метод може звертатися до нестатичних елементів класу через посередник - об'єкт, посилання або вказівник на об'єкт класу:

```
class A {
   int x;
public:
   A(int x=0) { this->x=x; }
   static void print() {
   A a(7);
     cout << a.x << endl;
   }
};
void main() {
   A::print();
}
```

Константний метод оголошується за допомогою слова `const`, яке записується після його заголовка. Якщо константний метод визначається за межами класу, то слово `const` повинно зберігатися в кінці заголовка функції. Константний метод не може змінювати дані класу. Статичний метод не може бути константним та навпаки.

```
// Програма не компілюється!
#include <iostream>
using namespace std;

class Demo {
  int i;
public:
  int geti() const {
    return i; // ok
  }

  void seti(int x) const {
    i = x; // помилка
  }
};

int main()
{
  Demo ob;

  ob.seti(1900);
  cout << ob.geti();
```

```
  return 0;
}
```

Іноді виникає необхідність модифікувати той або інший член класу за допомогою константної функції-члена, зберігаючи у недоторканості іншу частину. Для цього призначене ключове слово `mutable`. Воно відміняє атрибут `const`. Тобто член, оголошений з атрибутом `mutable`, може модифікуватися константною функцією-членом. У прикладі змінна і оголошена з атрибутом `mutable`, тому функція `seti()` може її модифікувати.

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

class Demo {
  mutable int i;
  int j;
public:
  int geti() const {
    return i; // все вірно
  }

  void seti(int x) const {
    i = x; // тепер все вірно
  }

// Ця функція не компілюється, так як j не mytable
//  void setj(int x) const {
//    j = x; // помилка
//  }
};

int main() {
  Demo ob;
  ob.seti(1900);
  cout << ob.geti();
  return 0;
}
```

1900

1.3. Вказівник на дані класу

У програмі може оголошуватися *вказівник на дані класу*:

```
тип ім'я_класу::* ім'я_вказівника=&імя_класу::ім'я_поля;
```

Наприклад:

```
class Base {
   public:
      int x;
      const int y;
      static int z;
      Base(int x1=0, int y1=0) : y(y1) { x=x1; }
};
int Base::z=3;
int Base::*p1 = &Base::x;
```

Для використання вказівника використовуються операції `.*, ->*`, наприклад

```
Base a;
a.*p1=5;   // a.x=5
Base *q=new Base;
q->*p1=5;
```

1.4. Вказівник на метод класу

Вказівник на метод класу можна використовувати для безпосереднього виклику методу або як аргумент чи тип результату іншої функції.

Оголошення та ініціалізація вказівника на нестатичний метод класу:

```
тип_функції (ім'я_класу::*ім'я_вказівника) (типи параметрів) =
&ім'я_класу::ім'я_методу;
```

Вказівник використовується за допомогою операції `.*` або `->*`, наприклад

```
class Base {
   int x;
public:
   Base(int x) {x=y;}
   int GetX() { return x; }
};
int (Base::*p) () = &Base::GetX;
void main() {
   Base a(5);
   Cout << (a.*p)() << endl;  // або a.GetX();
   Base *q=&a;
   Cout << (q->*p)() << endl;
}
```

1.5. Конструктори і деструктори

Конструктори розрізняють за призначенням:

- ініціалізації;
- копіювання;
- перетворення типів.

1.6. Конструктор ініціалізації

Перед використання обєктів їх необхідно проініціалізувати. У С++ автоматична ініціалізація обєктів здійснюється конструктором ініціалізації. *Конструктор* – це особлива функція, член класу, ім'я якої співпадає з іменем класу. Конструктори не повертають значення.

Конструктори ініціалізації викликаються:

- при створенні нового об'єкту у сегменті даних, стеку, динамічній пам'яті та ініціалізації його полів;

- для створення локального об'єкта у виразі:

```
тип_класу(список параметрів конструктора);
```

Приклади створення об'єктів, коли викликається конструктор ініціалізації:

```
A a(1,2);
A *p=new A(1,2);
A& q=*new A(1,2);
```

Якщо конструктор має тільки один аргкмент, то для ініціалізації об'єкту `obj` можна застосувати вираз `obj(x)` або `obj=x`. При використання виразу `obj=x` неявно створюється функція перетворення `obj(x)`. Іноді таке перетворення є небажаним. У таких випадках використовується ключове слово `explicit`, яке допускає тільки один спосіб виклику конструктора `obj(x)`.

```
// явний (explicit) конструктор
#include <iostream>
using namespace std;

class myclass {
  int a;
public:
// myclass(int x) { a = x; }
  explicit myclass (int x) { a=x; }  // явний конструктор без перетворень
  int geta() { return a; }
};
int main() {
//  myclass ob = 4; // автоматично перетворюється у вираз myclass(4)
```

```
  myclass ob(4);  // конструктор без перетворення
  cout << ob.geta();
  return 0;
}
```
4

Звичайно члени класу ініціалізуються всередині конструкторів.

// ініціалізація членів класу всередині конструктора
```
#include <iostream>
using namespace std;

class MyClass {
  int numA;
  int numB;
public:
// ініціалізація зі звичайним синтаксисом
  MyClass(int x, int y) {
    numA = x;
    numB = y;
  }
  int getNumA() { return numA; }
  int getNumB() { return numB; }
};

int main() {
  MyClass ob1(7, 9), ob2(5, 2);
  cout << "Значення в об'єкті ob1 є " << ob1.getNumB() <<
          " і " << ob1.getNumA() << endl;
  cout << "Значення в об'єкті ob2 є " << ob2.getNumB() <<
          " і " << ob2.getNumA() << endl;
  return 0;
}
```
Значення в об'єкті ob1 є 9 і 7
Значення в об'єкті ob2 є 2 і 5

Але такий не можна застосувати коли дані-члени оголошені константними, коли необхідно проініціалізувати посилання, і якщо у класі не передбачений конструктор за замовчуванням. У цих випадках використовується альтернативний спосіб ініціалізації членів класу при створенні об’єкту:

```
    Конструктор(список_аргументів) : член1(ініціалізатор),
                                     . . .
                                     членN(ініціалізатор)
    { //тіло конструктора }
```

// ініціалізація константних членів
```
#include <iostream>
using namespace std;

class MyClass {
  const int numA; // константний член
  const int numB; // константний член
public:
  // ініціалізація  numA і numB використовуючи список ініціалізації
  MyClass(int x, int y) : numA(x), numB(y) { }
  int getNumA() { return numA; }
  int getNumB() { return numB; }
};

int main() {
  MyClass ob1(7, 9), ob2(5, 2);
  cout << "Значення в ob1 є " << ob1.getNumB() <<
          " і " << ob1.getNumA() << endl;
  cout << "Значення в ob2 є " << ob2.getNumB() <<
          " і " << ob2.getNumA() << endl;
```

```
  return 0;
}
```
Значення в ob1 є 9 і 7
Значення в ob2 є 2 і 5

Конструктори із списком ініціалізації використовуються, якщо клас має тип, для якого не передбачений конструктор за замовчуванням.

```
// клас, який не має конструктора за замовчуванням
// Програма не компілюється!
#include <iostream>
using namespace std;

class IntPair {
public:
  int a;
  int b;
  IntPair(int i, int j) : a(i), b(j) { }
};

class MyClass {
  IntPair nums; // Клас IntPair не має конструктора без параметрів!
public:
   MyClass(int x, int y) {
    nums.a = x;
    nums.b = y;
  }
  int getNumA() { return nums.a; }
  int getNumB() { return nums.b; }
};

int main() {
  MyClass ob1(7, 9), ob2(5, 2);
  cout << "Значення в ob1 є " << ob1.getNumB() <<
          " і " << ob1.getNumA() << endl;
  cout << "Значення в ob2 є " << ob2.getNumB() <<
          " і " << ob2.getNumA() << endl;
  return 0;
}
```

```
// конструктор зі списком ініціалізації
#include <iostream>
using namespace std;

class IntPair {
public:
  int a;
  int b;
  IntPair(int i, int j) : a(i), b(j) { }
};

class MyClass {
  IntPair nums; // now OK
public:
  // Застосування списка ініціалізації
  // відкриває доступ до членів класу (також при відсутності source коду класу)
  MyClass(int x, int y) : nums(x,y) { }
  int getNumA() { return nums.a; }
  int getNumB() { return nums.b; }
};

int main()
{
  MyClass ob1(7, 9), ob2(5, 2);
```

```
  cout << "Значення в ob1 є " << ob1.getNumB() <<
          " і " << ob1.getNumA() << endl;
  cout << "Значення в ob2 є " << ob2.getNumB() <<
          " і " << ob2.getNumA() << endl;
  return 0;
}
```

Значення в ob1 є 9 і 7
Значення в ob2 є 2 і 5

1.7. Конструктор копіювання

Конструктор копіювання:

- має один параметр з посиланням на тип класу: `A&` або `const A&`. Може мати більше одного параметра, які задаються за замовчуванням;
- якщо не визначений у класі, то конструктор копіювання генерується автоматично і виконує порозрядне копіювання об'єктів;
- виконує копіювання усіх видів полів, зокрема константних та з типом посилання;
- якщо поля класу мають тип вказівника або посилання на будь-який тип, то конструктор копіювання повинен бути визначений користувачем.

Конструктор копіювання викликається при:

- створенні нового об'єкту та його ініціалізації вже існуючим об'єктом цього ж класу (новий об'єкт може бути створений у сегменті даних, стеку, динамічній пам'яті, може бути глобальним або локальним);
- передачі об'єктів через список параметрів функції "за значенням" (не через вказівник або посилання);
- поверненні функціям локальних об'єктів класу "за значенням " (не через вказівник або посилання).

Приклад оголошення та визначення конструктора копіювання:

```
clas A {
     const int x;
     int& y;
     int z;
// визначення конструктора ініціалізації
    A(int x1=0, int y1=0, int z1=0) : x(x1), y(y1) { z=z1; }
// оголошення конструктора копіювання
    A(const A&);
};
// визначення конструктора копіювання
A::A(const A& a) : x(a.x), y(a.y)  { z=a.z; }
```

Приклади створення об'єктів, коли викликається конструктор копіювання:

```
A a;
A b(a);          // A b=a;
A *p=new A(a);
A& q=*new A(a);
```

Якщо у протоколі класу є поля з типом вказівника або посилання на будь-який тип, то для такого класу необхідно визначити власний конструктор, який здійснює глибоке копіювання:

```
class A {
   int *p;
   public:
A(int x) { p=new int(x); }          // звичайний конструктор
A(const A& a) { p=new int(*a.p); } // конструктор копіювання
~A(){ delete p; }
int  Get(void)  { return *p; }
void Set(int x) { *p=x; }
};
void main() {
   Int x=5;
   A a(x);
   A b(a);
```

```
cout << a.Get() << ” ” << b.Get() << endl;
}
```

Конструктор копіювання закріплює за вказівником `p` нову область динамічної пам’яті, тому `a.p != b.p`.

1.8. Конструктор перетворення типів

Конструктор перетворення типів призначений для перетворення значення заданого типу до типу класу. Має один параметр з типом, який вимагає перетворення, або декілька параметрів, значення яких задаються за замовчуванням. Наприклад:

```
clas A {
    int x;
public:
// конструктори перетворення типів
    A(int x) { this->x=x; }
    A(double x) { this->x=int(x); }
};
```

Конструктор перетворення типів викликається у таких випадках:

- для ініціалізації об’єкта класу значенням, тип якого є відмінним від типу цього класу;
- для передавання параметрів у функцію, коли формальний параметр має класовий тип, а фактичний - інший тип;
- для повернення значення функцією, коли функція має класовий тип, а вираз оператора `return` має інший тип.

1.9. Деструктор

Деструктор призначений для звільнення пам’яті, відведеної під поля об’єкту. Деструктор не перевантажується і не успадковується. Деструктор викликається автоматично, якщо об’єкт виходить із області досяжності програми. Пам'ять, відведена у стеку для локального об’єкта з класом пам’яті “автоматичний” (`auto`), буде звільнена під час завершення роботи функції. Пам'ять статичного локального об’єкту буде звільнена перед завершенням роботи функції `main()`. Якщо локальний об’єкт розміщено в області динамічної пам’яті, то при завершенні роботи функції деструктор не викликається. Всю незвільнену динамічну пам'ять буде автоматично звільнено після завершення роботи програми.

Деструктор можна викликати явно:

```
Class A {
    Int *p;
Public:
A(int x) { p=new int(x); }
~A() { delete p; }
…
};
Void main() {
A a(5);
….
b->!A();
a.!A();
}
```

Запитання.

1. Призначення дружньої функції і як вона описується в класі. Чи успадковується дружня функція.
2. Що таке дружній клас і який він має доступ до базового класу.
3. Що таке вбудовувана функція. В чому її недоліки і переваги.
4. Як використовуються статичні члени класу (дані і методи).
4. Що таке вказівники на дані і функції класу, для чого вони використовуються.

5. Константні члени класу (дані і методи). Атрибут `mutable`.

6. Що таке вказівник `this` і коли він використовується.

7. Що таке конструктор і деструктор. Коли вони викликаються.

8. Для чого і як конструкторам передаються параметри. Як ініціалізувати дані за допомогою конструктора.

9. Конструктор ініціалізації. Особливості використання конструкторів ініціалізації.

10. Що таке конструктор копіювання і коли він викликається.

11. Що таке конструктор перетворення типів і коли він викликається.

12. Оголошення, автоматичний та явний виклик деструктора.

Завдання.

1. Задано неповне визначення класу

```
class strtype {
   char *p;
   int len;
public:
   char *getstring() { return p; }
   int getlength() { return len; }
```

Добавити в це визначення два конструктори. В першому не має бути параметрів. Він має виділяти 255 байтів пам'яті оператором *new*, ініціалізувати цю пам'ять нульовою стрічкою і встановити змінну *len* рівною 255. В другому конструкторі має бути два параметри. Перший – це стрічка, яка використовується при ініціалізації, другий – число байтів, які необхідно виділити. Показати в програмі роботу цих конструкторів.

2. Конструктор класу А виділяє в динамічній пам'яті масив int[80]. Добавити в клас конструктор копіювання. Створити об'єкт класу А і зробити його копію В. Проініціалізувати масиви обох об'єктів різними числами і вивести їх значення.

3. Виконати завдання згідно номеру студента у журналі групи.

3.1. Створити клас ДАТИ з полями у закритій частині: день (1-31), місяць (1-12), рік (ціле число). Клас має конструктор, методи встановлення дня, місяця і року, методи отримання значень дня місяця року, а також методи виведення за шаблонами "12 лютого 2015" і 12.02.2015. Методи встановлення полів класу повинні перевіряти коректність параметрів, що задаються.

3.2. Створити клас МАТРИЦІ, який у закритій частині містить вказівник на `int`, кількість рядків, стовпців та змінну стану. Визначити конструктор без параметрів, конструктор з одним параметром, та конструктор з двома параметрами, деструктор. Визначити метод для повернення значення елемента за індексами `[i][j]`. Визначити функцію виведення матриці. Визначити функції додавання, віднімання та множення матриць. Визначити функції множення матриці на число. Перевірити роботу цього класу. У випадку нестачі пам'яті, невідповідності розмірностей, виходу за межі масиву встановлювати код помилки у змінній стану.

3.3. Створити клас ЧАСУ з полями у закритій частині: година (0-23), хвилини (0-59), секунди (0-59). Клас має конструктор, методи встановлення часу, методи отримання годин, хвилин, і секунд, а також два методи виведення за шаблонами "16 годин 18 хвилин 3 секунди" і "4 p.m. 18 хвилин 3 секунди". Методи встановлення полів класу повинні перевіряти коректність параметрів, що задаються.

3.4. Створити клас ПРЯМОКУТНИК. У закритій частині визначити поля - висоту і ширину. Метод класу обчислюють площу, периметр, встановлюють поля даних і повертають їхні значення. У методах встановлення значень полів класу необхідно перевіряти коректність заданих параметрів. Визначити функцію виведення елементів класу.

3.5. Створити клас ОДНОЗВЯЗНИЙ СПИСОК. Методи класу додають елемент до списку, вилучають елементи зі списку, сортують список, відображають елементи списку від початку до кінця. Вилучити заданий елемент списку.

3.6. Створити клас ДВОЗВЯЗНИЙ СПИСОК. Методи класу додають елемент до списку,

вилучають елементи зі списку, сортують список, відображають елементи списку від початку до кінця. Знайти у списку заданий елемент.

3.7. Створити клас КАЛЕНДАР. У закритій частині визначити дані - день, місяць, рік. Визначити необхідні конструктори, деструктори та методи. Методи класу встановлюють та зчитують значення полів даних, визначають назву тижня за заданою датою, виводять результат на екран. Ввести дату та визначити назву дня свого дня народження.

3.8. Створити клас ДАТА з полями у закритій частині: день (1-31), місяць (-12), рік (ціле число). Клас має конструктор, методи встановлення дня, місяця і року, методи читання дня, місяці і року. У методах встановлення полів класу необхідно перевіряти коректність заданих параметрів. Визначити метод, який збільшує значення дати на 1 день.

3.9. Створити клас ЧАС з полями у закритій частині: година(0-23), хвилини (0-59), секунди (0-59). Клас має конструктор, методи встановлення часу, методи читання години, хвилини і секунди. У методах встановлення полів класу необхідно перевіряти коректність параметрів, що задаються. Розробити метод для збільшення значення часу на 1 секунду, 1 хвилину, 1 годину.

3.10. Створити клас КВАДРАТ з полями у закритій частині: координати головної діагоналі. Методи класу обчислюють довжину сторони квадрату, площу, периметр, встановлюють значення полів і повертають їх значення. Методи встановлення полів класу повинні перевіряти коректність параметрів, що задаються.

3.11. Створити клас СТЕК. Розробити методи класу для введення елемента у стек та вилучення елемента зі стека. Розробити функцію, яка визначає кількість елементів у стеку.

3.12. Розробити клас квадратна матриця. У закритій частині визначити дані: порядок матриці та вказівник на її початок в області динамічної пам’яті. Клас має конструктор та деструктор. Розробити методи класу, які виконують такі операції: встановлення та виведення значень елементів матриці, визначення сліду матриці (суми елементів головної діагоналі), суми елементів вище та нижче головної діагоналі.

3.13. Розробити клас МНОЖИНА ЦІЛИХ ЧИСЕЛ. У закритій частині визначити вказівник на цілий тип (на елементи множини). Визначити конструктори, деструктор та методи створення множини, виведення вмісту множини, об’єднання, різниці та перетину множин.

3.14. Розробити клас РЯДОК СИМВОЛІВ. У закритій частині визначити вказівник на символьний тип (на початок рядка). Визначити конструктори, деструктор та методи введення-виведення рядка, конкатенації, порівняння рядків, перевірки входження підрядка у рядок.

3.15. Розробити клас СТУДЕНТ. У закритій частині визначити дані - прізвище, номер залікової книжки, оцінки з предметів. Визначити конструктори, деструктор та методи встановлення і читання значень полів даних, знаходження середнього балу, визначення кількості незадовільних оцінок.

3.16. Розробити клас КНИГА. У закритій частині визначити дані - автор, назва, видавництво, рік видання. Визначити конструктори, деструктор та методи встановлення і читання значень полів даних, визначення відповідності книги пошуковим критеріям.

3.17. Цифровий лічильник - це змінна з обмеженим діапазоном, яка скидається у початкове значення, коли її цілочисельне значення досягає визначного максимуму. Опишіть клас такого ЛІЧИЛЬНИКА. Забезпечте можливість встановлення максимального і мінімального значень, збільшення значень лічильника на 1, повернення поточного значення.

3.18. Створити клас для виконання арифметичних операцій над ЦІЛИМИ ЧИСЛАМИ у двійковій системі числення. Числа задаються як стрічка символів в області динамічної пам’яті. Визначити конструктори, деструктор, методи виконання операцій, введення чисел та виведення результату.

3.19. Створити клас для виконання арифметичних операцій над ЦІЛИМИ ЧИСЛАМИ у шістнадцятковій системі числення. Числа задаються як стрічка символів в області динамічної пам’яті. Визначити конструктори, деструктор, методи виконання операцій, введення чисел та виведення результату.

3.20. Розробити клас, який виконує статистичну обробку ТЕКСТОВОГО ФАЙЛУ - підрахунок кількості символів, слів, речень. Визначити необхідні конструктори, деструктор та методи роботи з файлом.

3.21. Розробити клас для роботи зі СЛОВНИКОМ, який складається з масиву слів та їх перекладу іншою мовою. Визначити конструктори, деструктор та методи для додавання нових слів, пошуку слів, злиття двох словників без повторень елементів.

3.22. Розробити клас для роботи з КАТОТЕКОЮ КНИГ. Клас містить дані про назви книги: автор, назва книги, видавництво, рік видання. Реалізувати методи додавання даних про книгу до картотеки, витирання з картотеки. Пошук книг за прізвищем автора, назвою книги, роком видання.

3.23. Розробити клас для роботи з ПРАЙС АРКУШЕМ комп’ютерної фірми, у якому вказано дані про марку комп’ютера, тип процесора, частоту процесора, об’єм оперативної та дискової пам’яті, характеристики відеокарти, ціну комп’ютера.

3.24. Створити клас для створення ДІЛОВОГО ЗАПИСНИКА з планом роботи на тиждень. Визначити конструктори, деструктор, методи роботи із записником - додавання нових записів, корегування та витирання записів.

3.25. Створити клас для роботи з ЧЕРГОЮ мешканців міста, які потребують покращення житлових умов. У закритій частині класу визначити поля даних - номер черги, прізвище та ініціали, склад сім’ї, дата поставлення у чергу. У відкритій частині класу визначити методи для поставлення та вилучення з черги, пошуку за номером черги, прізвищу, дані.

3.26. Розробити клас для роботи з ЧЕРГОЮ на біржі праці. У закритій частині класу визначити необхідні дані: прізвище, паспортні дані, освіта, професія, дата поставлення на облік, бажаний вид зайнятості, бажана зарплата. Визначити методи для роботи з цими даними - додавання, вилучення, корегування записів, виведення статистики.

2. Завдання для самостійної роботи № 2

```
/* 1. chapter 12, listing 1 – чередування специфікацій доступу public, private в класі */
#include <iostream>
#include <cstring>
using namespace std;

class employee {
  char name[80]; // private – закриті по замовчуванню
public:
  void putname(char *n); // public – відкриті члени
  void getname(char *n);
private:
  double wage; // знов, private
public:
  void putwage(double w); // знов public
  double getwage();
};

void employee::putname(char *n) {
  strcpy(name, n);
}

void employee::getname(char *n) {
  strcpy(n, name);
}

void employee::putwage(double w) {
  wage = w;
}

double employee::getwage() {
  return wage;
}

int main() {
  employee ted;
  char name[80];

  ted.putname("Ted Jones");
  ted.putwage(75000);

  ted.getname(name);
  cout << name << " makes $";
  cout << ted.getwage() << " per year.";

  return 0;
}
>Ted Jones makes $75000 per year.

/* 2. listing 3 – доступ до відкритих членів класу */
#include <iostream>
using namespace std;

class myclass {
public:
  int i, j, k; // доступні з любої точки програми
};

int main() {
  myclass a, b;
```

```
  a.i = 100; // доступ до i, j, k є
  a.j = 4;
  a.k = a.i * a.j;

  b.k = 12; // a.k i b.k належать різним об'єктам
  cout << a.k << " " << b.k;

  return 0;
}
```
>**400 12**

```
/* 3. listing 4 - використання структури як класу */
#include <iostream>
#include <cstring>
using namespace std;

struct mystr {
  void buildstr(char *s);  // відкритий член
  void showstr();
private:
  char str[255];           // закритий член
};

void mystr::buildstr(char *s) {
  if(!*s) *str = '\0'; // ініціалізація рядка
  else strcat(str, s);
}

void mystr::showstr() {
  cout << str << "\n";
}

int main() {
  mystr s;
  s.buildstr(""); // ініціалізація
  s.buildstr("Привіт ");
  s.buildstr("всім!");
  s.showstr();
  return 0;
}
```
>**Привіт всім!**

```
/* listing 6 – union, об'єднання в C++ можуть містити не тільки дані, але і
   функції */
#include <iostream>
using namespace std;

union swap_byte {
  void swap();
  void set_byte(unsigned short i);
  void show_word();

  unsigned short u;
  unsigned char c[2];
};

void swap_byte::swap() {
  unsigned char t;

  t = c[0];
  c[0] = c[1];
  c[1] = t;
}
```

```
void swap_byte::show_word() {
  cout << u;
}

void swap_byte::set_byte(unsigned short i) {
  u = i;
}

int main() {
  swap_byte b;

  b.set_byte(1);
  b.swap();
  b.show_word();

  return 0;
}
```
>**256**

/* 4. listing 7 – union, об'єднання без імен */
```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
  // визначення безіменного union
  union {
    long l;
    double d;  char s[10];
  } ;

  // тепер, відкритий прямий доступ до елементів об'єднання без оператора "."
  l=1;
  cout << l << " ";
  d=2.22;
  cout << d << " ";
  strcpy(s,"123456789a");
  cout << s << "\n";
  return 0;
}
```
>**1 2.22 123456789a**

/* 5. listing 8 - за допомогою ключового слова friend можна надати звичайній функції доступ до закритих членів класу */
```
#include <iostream>
using namespace std;

class myclass {
  int a, b;
public:
  friend int sum(myclass x);
  void set_ab(int i, int j);
};

void myclass::set_ab(int i, int j) {
  a = i;
  b = j;
}

// Увага: sum() не є функцією членом класу
int sum(myclass x) {
  /* Так як() є дружньою по відношенню до класу myclass, вона має прямий доступ до
a і b */
```

```
  return x.a + x.b;
}

int main() {
  myclass n;
  n.set_ab(3, 4);
  cout << sum(n);
  return 0;
}
>7
```

```
/* 6. listing 9 – дружня функція по відношенню до двох класів */
#include <iostream>
using namespace std;

const int IDLE = 0;
const int INUSE = 1;

class C2;
// неповне оголошення, так як в класі C1 дружня функція idle має аргумент C2

class C1 {
  int status;
// дорівнює IDLE якщо екран вільний, і INUSE, якщо екран зайнятий
public:
  void set_status(int state);
  friend int idle(C1 a, C2 b);
};

class C2 {
  int status;
// дорівнює IDLE якщо екран зайнятий, і INUSE якщо екран вільний
public:
  void set_status(int state);
  friend int idle(C1 a, C2 b);
};

void C1::set_status(int state) {
  status = state;
}

void C2::set_status(int state) {
  status = state;
}

int idle(C1 a, C2 b) {
  if(a.status || b.status) return 0;
  else return 1;
}

int main() {
  C1 x;
  C2 y;

  x.set_status(IDLE);
  y.set_status(IDLE);

  if(idle(x, y)) cout << "Екран вільний.\n";
  else cout << "Зайнятий.\n";

  x.set_status(INUSE);

  if(idle(x, y)) cout << " Екран вільний.\n";
  else cout << " Зайнятий.\n";
```

```
  return 0;
}
>Екран вільний.
Зайнятий.

/* 7. listing 10 – дружня функція може бути членом іншого класу */
#include <iostream>
using namespace std;

const int IDLE = 0;
const int INUSE = 1;

class C2;  // forward declaration – неповне описання

class C1 {
  int status;
// дорівнює IDLE, якщо екран зайнятий, і INUSE, якщо екран вільний
  // ...
public:
  void set_status(int state);
  int idle(C2 b);  // тепер функція idle – член класу C1
};

class C2 {
  int status;
// дорівнює IDLE, якщо екран зайнятий, і INUSE, якщо відьний
  // ...
public:
  void set_status(int state);
  friend int C1::idle(C2 b);  // дружня функція idle

};

void C1::set_status(int state) {
  status = state;
}

void C2::set_status(int state) {
  status = state;
}

// Функція idle() є членом класу C1, але другом класу C2
int C1::idle(C2 b) {
  if(status || b.status) return 0;
  else return 1;
}

int main() {
  C1 x;
  C2 y;
  x.set_status(IDLE);
  y.set_status(IDLE);
  if(x.idle(y)) cout << x.idle(y) << "Екран вільний.\n";
  else cout << "Зайнятий.\n";
  x.set_status(INUSE);
  if(x.idle(y)) cout << x.idle(y) << " Екран вільний.\n";
  else cout << "Зайнятий.\n";
  return 0;
}
>Екран вільний.
Зайнятий.

/* 8. listing 11 – дружні класи */
```

```
// Один клас може бути дружнім по відношенню до іншого. У цьому випадку
// дружній клас і всі його члени-функції мають доступ до закритих членів,
// визначених у другому класі.
#include <iostream>
using namespace std;

class TwoValues {
  int a;
  int b;
public:
  TwoValues(int i, int j) { a = i; b = j; }
  friend class Min;    // дружній клас
};

class Min {
public:
  int min(TwoValues x);
};

int Min::min(TwoValues x) {
  return x.a < x.b ? x.a : x.b;
}

int main() {
  TwoValues ob(10, 20);
  Min m;
  cout << m.min(ob);
  return 0;
}
```
>**10**

/* 9. listing 12 – функції, які підставляються, а не викликаються. Ключове слово inline */

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

inline int max(int a, int b) {
  return a>b ? a : b;
}
int main() {
  cout << max(10, 20);           // cout << (10>20) ? 10 : 20);
  cout << " " << max(99, 88);    // cout << ” ” << (99>88 ? 99 : 88);
  return 0;
}
```
>**20 99**

/* 10. listing 13 – програма з точки зору компілятора після підставлення */

```
#include <iostream>
using namespace std;
int main() {
  cout << (10>20 ? 10 : 20);
  cout << " " << (99>88 ? 99 : 88);
  return 0;
}
```
>**20 99**

/* 11. listing 14 – підставляємі функції, можуть бути членами класу */

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

class myclass {
  int a, b;
public:
  void init(int i, int j);
```

```
  void show();
};

// Створення inline функції.
inline void myclass::init(int i, int j) {
  a = i;
  b = j;
}
// Створити іншу inline функцію.
inline void myclass::show() {
  cout << a << " " << b << "\n";
}

int main() {
  myclass x;
  x.init(10, 20);
  x.show();
  return 0;
}
>10 20
```

/* 12. listing 15 – короткі функції можна визначити всередині класу і вони перетворюються в inline функції*/

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

class myclass {
  int a, b;
public:
  // функції автоматично inline
  void init(int i, int j) { a=i; b=j; }
  void show() { cout << a << " " << b << "\n"; }
};

int main() {
  myclass x;
  x.init(10, 20);
  x.show();
  return 0;
}
>10 20
```

/* 13. listing 17 – конструктори з параметрами */

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

class myclass {
  int a, b;
public:
  myclass(int i, int j) {a=i; b=j;}
  void show() {cout << a << " " << b;}
};

int main() {
  myclass ob(3, 5);
  ob.show();
  return 0;
}
>3 5
```

/* 14. listing 20 – конструктор з параметрами */

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

```
const int IN = 1;
const int CHECKED_OUT = 0;

class book {
  char author[40];
  char title[40];
  int status;
public:
  book(char *n, char *t, int s);
  int get_status() {return status;}
  void set_status(int s) {status = s;}
  void show();
};

book::book(char *n, char *t, int s) {
  strcpy(author, n);
  strcpy(title, t);
  status = s;
}

void book::show() {
  cout << title << " by " << author;
  cout << " is ";
  if(status==IN) cout << "in.\n";
  else cout << "out.\n";
}

int main() {
  book b1("Twain", "Tom Sawyer", IN);
  book b2("Melville", "Moby Dick", CHECKED_OUT);

  b1.show();
  b2.show();

  return 0;
}
```

>**Tom Sawyer by Twain is in.**
Moby Dick by Melville is out.

/* 15. listing 21 – конструктор з одним параметром можна ініціалізувати оператором *obj* */

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

class X {
  int a;
public:
  X(int j) { a = j; }
  int geta() { return a; }
};

int main() {
  X ob = 99; // передати параметру j значення 99
//  X ob = X(99);
  cout << ob.geta(); // вивести на екран 99
  return 0;
}
```

>**99**

/* 16. listing 23 – статичні змінні-члени класу мають тільки один екземпляр цієї змінної для всіх об'єктів цього класу. Оголошення статичної змінної-члена в класі не означає її визначення (виділення пам'яті). Для виділення пам'яті її потрібно визначити поза класом, тобто глобально. */

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

class shared {
  static int a;  // статична змінна-член
  int b;
public:
  void set(int i, int j) {a=i; b=j;}
  void show();
} ;

int shared::a; // визначення змінної поза класом, глобально

void shared::show() {
  cout << "Це статична змінна a: " << a << " і нестатична b: " << b << "\n";
}

int main() {
  shared x, y;

  x.set(1, 1); // присвоїти a значення 1
  cout << "Obj x. ";
  x.show();

  cout << "Obj y. ";
  y.set(2, 2); // змінити значення a на 2
  y.show();


  cout << "Obj x. ";
  x.set(3,3);
  x.show(); /* Тут одночасно змінюється значення змінних-членів
               об'єктів x і y,
               оскільки змінна a використовується обома об'єктами. */

  cout << "Obj y. ";
  y.show();

  return 0;
}
```

```
>Obj x. Це статична змінна a: 1 і нестатична: 1
Obj y. Це статична змінна a: 2 і нестатична b: 2
Obj x. Це статична змінна a: 3 і нестатична b: 3
Obj y. Це статична змінна a: 3 і нестатична b: 2
```

```
/* 17. listing 24 – статична змінна-член створюється до створення першого об'єкта
класу */
#include <iostream>
using namespace std;

class shared {
public:
  static int a;
} ;

int shared::a; // визначення змінної a

int main() {
  shared::a = 99; // ініціалізація перед створенням об'єкта
  cout << "Початкове значення змінної a: " << shared::a;
  cout << "\n";
  shared x;  // створення об'єкту
  cout << "Значення змінної як члена x.a: " << x.a;
  return 0;
```

```
}
>Початкове значення змінної: 99
Значення змінної як члена x.a: 99

/* 18. listing 25 – статична змінна, як індикатор використання ресурсів усіма об'єктами класу */
#include <iostream>
using namespace std;

class cl {
  static int resource;
public:
  int get_resource();
  void free_resource() {resource = 0;}
};

int cl::resource; // визначення ресурсу

int cl::get_resource() {
  if(resource) return 0; // ресурс зайнятий
  else {
    resource = 1;
    return 1;  // ресурс наданий об'єкту
  }
}

int main() {
  cl ob1, ob2;
  if(ob1.get_resource()) cout << "ob1 володіє ресурсом\n";
  if(!ob2.get_resource()) cout << "ob2 доступ заборонений\n";
  ob1.free_resource();  // Звільнення ресурсу
  if(ob2.get_resource())
    cout << "ob2 може використовувати ресурс\n";

  return 0;
}
>ob1 володіє ресурсом
ob2 доступ заборонеий
ob2 може використовувати ресурс

/* 20. listing 26 – визначення кількості існуючих об'єктів конкретного класу за
   допомогою статичної змінної */
#include <iostream>
using namespace std;

class Counter {
public:
  static int count;
  Counter() { count++; }
  ~Counter() { count--; }
};
int Counter::count;

void f();

int main(void) {
  Counter o1;
  cout << "Існуючі об'єкти: ";
  cout << Counter::count << "\n";

  Counter o2;
  cout << " Існуючі об'єкти: ";
  cout << Counter::count << "\n";
```

```
  f();
  cout << " Існуючі об'єкти: ";
  cout << Counter::count << "\n";

  return 0;
}

void f() {
  Counter temp;
  cout << " Існуючі об'єкти:: ";
  cout << Counter::count << "\n";
  // temp знищується після повернення з функції f()
}
```
>Існуючі об'єкти: 1
Існуючі об'єкти: 2
Існуючі об'єкти: 3
Існуючі об'єкти: 2

/* 21. listing 27 – статичні функції-члени мають доступ тільки до інших статичних членів класу. Приклад надання одного ресурсу різним об'єктам. */

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

class cl {
  static int resource;
public:
  static int get_resource();
  void free_resource() { resource = 0; }
};

int cl::resource; // визначення ресурсу

int cl::get_resource() {
  if(resource) return 0; // ресурс зайнятий
  else {
    resource = 1;
    return 1;  // ресурс наданий іншому об'єкту
  }
}

int main() {
  cl ob1, ob2;

  /* Функція get_resource() є статичною, тому її можна викликати незалежно від
любого обєкту */
  if(cl::get_resource()) cout << "ob1 має ресурс\n";
  if(!cl::get_resource()) cout << "ob2 доступ заборонений\n";
  ob1.free_resource();

// може викликатися використовуючи об'єктний синтаксис
  if(ob2.get_resource())
     cout << "ob2 тепер може використовувати ресурс\n";

  return 0;
}
```
>ob1 має ресурс
ob2 доступ заборонений
ob2 тепер може використовувати ресурс

/* 22. listing 28 – статичні функції члени можуть ініціалізовувати закриті статичні змінні-члени до створення реальних об'єктів */

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

```
class static_type {
  static int i;
public:
  static void init(int x) {i = x;}
  void show() {cout << i;}
};

int static_type::i; // визначення змінної i

int main() {
  // ініціалізація статичної змінної перед створенням об'єкта
  static_type::init(100);

  static_type x;
  x.show(); // displays 100

  return 0;
}
>100
```

/* 23. listing 29 – конструктори локальних об'єктів викликаються послідовно при їх створенні, а деструктори викликаються у зворотному порядку. Конструктори глобальних об'єктів виконуються до функції main(), а деструктори після закінчення роботи main()*/

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

class myclass {
public:
  int who;
  myclass(int id);
  ~myclass();
} glob_ob1(1), glob_ob2(2);

myclass::myclass(int id) {
  cout << "Ініціалізація " << id << "\n";
  who = id;
}

myclass::~myclass() {
  cout << "Знищення " << who << "\n";
}

int main() {
  myclass local_ob1(3);

  cout << "Цей рядок не буде першим.\n";

  myclass local_ob2(4);

  return 0;
}
>Ініціалізація 1
Ініціалізація 2
Ініціалізація 3
Цей рядок не буде першим.
Ініціалізація 4
Знищення 4
Знищення 3
Знищення 2
Знищення 1
```

/* 24. listing 32 – локальні класи оголошуються всередині функцій */

```
#include <iostream>
```

```
using namespace std;

void f();

int main() {
  f();
  // клас myclass з цього місця не видно
  return 0;
}

void f() {
  class myclass {
    int i;
  public:
    void put_i(int n) { i=n; }
    int get_i() { return i; }
  } ob;

  ob.put_i(10);
  cout << ob.get_i();
}
>10

/* 25. listing 33 – передача об'єктів функціям по значенню*/
#include <iostream>
using namespace std;

class myclass {
  int i;
public:
  myclass(int n);
  ~myclass();
  void set_i(int n) { i=n; }
  int get_i() { return i; }
};

myclass::myclass(int n) {
  i = n;
  cout << "Створення " << i << "\n";
}

myclass::~myclass() {
  cout << "Знищення " << i << "\n";
}

void f(myclass ob);

int main() {
  myclass o(1);

  f(o);
  cout << "Змінна i в функції main: ";
  cout << o.get_i() << "\n";

  return 0;
}

void f(myclass ob) {
  ob.set_i(2);

  cout << "Це локальна змінна i: " << ob.get_i();
  cout << "\n";
}
//  конструктор викликався один раз, а деструктор два рази. При створенні копії
```

аргумента звичайний конструктор не викликається. Замість нього викликається конструктор копіювання.

>**Створення 1**
Це локальна змінна i: 2
Знищення 2
Змінна i в функції main: 1
Знищення 1

```
/* 26. listing 34 – повернення функціями об’єктів викликаючому модулю */
// Returning objects from a function.
#include <iostream>
using namespace std;

class myclass {
  int i;
public:
  void set_i(int n) { i=n; }
  int get_i() { return i; }
};

myclass f();  // повернення об’єкта типу myclass

int main() {
  myclass o;
  o = f();
  cout << o.get_i() << "\n";
  return 0;
}

myclass f() {
  myclass x;
  x.set_i(1);
  return x;
}
```

>**1**

```
/* 27. listing 35 – присвоєння об’єктів однакового типу один одному */
// Присвоєння об’єктів.
#include <iostream>
using namespace std;

class myclass {
  int i;
public:
  void set_i(int n) { i=n; }
  int get_i() { return i; }
};

int main() {
  myclass ob1, ob2;
  ob1.set_i(99);
  ob2 = ob1; // присвоєння даних об’єкта ob1 об’єкту ob2
  cout << "Змінна i з об’єкта ob2': " << ob2.get_i();
  return 0;
}
```

>**Змінна i з об'єкта ob2: 99**

```
/* 28. listing 36 (timer.cpp) -  робота з часом */
#include <iostream>
#include <ctime>
using namespace std;
//clock_t clock(void);
```

```
class timer {
  double start, stop;
public:
    timer() {
    start = (double) clock() / CLOCKS_PER_SEC;
    cout << "Конструктор " << start << endl;
  } ; // constructor

  ~timer() {
     stop = (double) clock() / CLOCKS_PER_SEC;
     cout << "Деструктор " << stop << "\n";
  }; // destructor
};

int main() {
  timer ob;
  char c;

  // delay ...
  cout << "Натисніть любу клавішу і ENTER: ";
  cin >> c;

  return 0;
}
```

/* 29. listing 37 (swap.cpp) – родова функція. Переставлення значень змінних незалежно від типу */

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

void swap(void *&a, void *&b) {
  void *temp=a;
  a=b;
  b=temp;
}

int main() {
  int *i, *j;
  i = new int;   *i=5;
  j = new int;   *j=20;

  cout << " *i=" << *i << " *j=" << *j << endl;
  swap(i,j);
  cout << " *i=" << *i << " *j=" << *j << endl;
  delete i;
  delete j;

  double *x, *y;
  x = new double;   *x=4.0;
  y = new double;   *y=10.0;

  cout << " *x=" << *x << " *y=" << *y << endl;
  swap(x,y);
  cout << " *x=" << *x << " *y=" << *y << endl;
  delete x;
  delete y;

  return 0;
}
```

```
/.swap
*i=5 *j=20
*i=20 *i=5
*x=4 *y=10
```

***x=10 *y=4**

```
/* 30. 38 (bits.cpp) – використання об’єднання для виведення бітів в байтах */
#include <iostream>
using namespace std;

union bits {
  bits(double n);       // об’єднання може містити члени-функції
  void show_bits();
  double d;
  unsigned char c[sizeof(double)];
};

bits::bits(double n) {
  d = n;
}

void bits::show_bits() {
  int i, j;

  for(j = sizeof(double)-1; j>=0; j--) {
    cout << "Біти в байті " << j << ": ";
    for(i = 128; i; i >>= 1)   // 128 -> 1000'0000
      if(i & c[j]) cout << "1";
      else cout << "0";
    cout << "\n";
  }
}

int main() {
  double i;
  cout << "Введіть число double xxxx.xxxx : ";
  cin >> i;
  bits ob(i);
  ob.show_bits();
  return 0;
}
```

/.bits
Введіть число xxxx.xxxx double: 2012.1234
Біти в байті 7: 01000000
Біти в байті 6: 10011111
Біти в байті 5: 01110000
Біти в байті 4: 01111110
Біти в байті 3: 01011100
Біти в байті 2: 10010001
Біти в байті 1: 11010001
Біти в байті 0: 01001110

/* 31. 39 (bits_union.cpp)– використання неіменованого об’єднання для виведення байтів */

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

int main() {
  union {
    unsigned char bytes[8];
    double value;      // розмір double 8 байтів
  };
  int i;

  value = 859345.324;  // звернення до елементу неіменованого об’єднання
  // виведення байтів числа типу double
  for(i=0; i<8; i++)
    cout << (int) bytes[i] << " ";
```

```
  return 0;
}
./bits_union
248 83 277 165 162 57 42 65
```