ЛАБОРАТОРНА РОБОТА №2

Тема: Робота з лінійними списками. Конструктор і деструктор класу

Мета: Навчитись використовувати конструктори і деструктори класів, створювати класи для опису лінійних списків

КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

1. Конструктори та деструктори

Конструктор – це метод (функція-член класу), який викликається при створенні нового об'єкта класу. Конструктор являє собою метод класу, що полегшує програмам ініціалізацію атрибутів (елементів даних) класу. Він викликається автоматично кожен раз, коли створюється об'єкт класу. Кожен окремий екземпляр об'єкта повинен ініціалізуватись з використанням конструктора.

Конструктор має таке ж ім'я, як і клас. Конструктор не повертає результату (void писати не потрібно і не можна). Описується конструктор так:

```
class MyClass {  // клас з назвою MyClass
private:         // private можна не писати, бо елементи класу є
                 // закриті по замовчуванню
               …      …  // закриті елементи даних класу
public:        …            // відкриті елементи даних та методи класу
       MyClass();  //конструктор (оголошення конструктора)
                   // має таке ж ім'я, як і назва класу
};

MyClass::MyClass(); // реалізація конструктора
{   ….   }
```

Якщо клас є похідним від базового класу, або містить в собі об'єкти, які мають власні конструктори, то конструктор базового класу та конструктори об'єктів-членів класу викликаються автоматично перед викликом конструктора похідного класу. В тілі конструктора не можна використовувати оператор return. Конструктор не може бути віртуальним.

Деструктор – навпаки викликається при знищенні об'єкту, і тому в ньому необхідно звільнити пам'ять зарезервовану в конструкторі. Він викликається автоматично кожен раз, коли видаляється об'єкт класу. Для глобальних статичних змінних це відбувається при завершенні роботи програми, в порядку, зворотному до їх оголошення, для автоматичних — при виході з блоку, для динамічних — при виклику оператора **delete**.

Деструктор має таке ж ім'я, як і клас, за винятком того, що перед ним додається символ "~" (тильда). Деструктор не повертає результату (void писати не потрібно) і не отримує ніяких параметрів. Описується деструктор так:

```
class MyClass   // клас з назвою MyClass
{      …      …
public:       …
       MyClass(); // оголошення конструктор (має ім'я, як назва класу)
       ~MyClass(); // оголошення деструктора (в назві передує "тильда")
}
MyClass::~MyClass(); // реалізація деструктора
{   ….   }
```

Деструктор для базового класу виконується після деструктора похідного від нього класу. Деструктори для об'єктів-членів виконуються після деструктора для об'єкта, членами якого вони є. Деструктор може бути віртуальним.

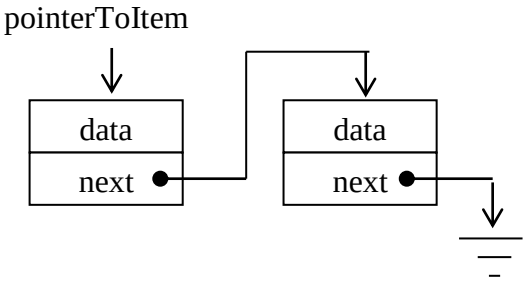
Динамічні об'єкти конструюються та знищуються з використанням операторів **new** та **delete**. Пам'ять для збереження динамічних об'єктів в C++ виділяється за допомогою оператора

new. Коли за допомогою оператора **new** створюється об'єкт класу, то неявно викликається конструктор цього класу, який ініціалізує елементи класу та виділяє під них пам'ять. Оператор **delete** для об'єкта неявно викликає його деструктор для звільнення пам'яті.

2. Лінійні списки

Для представлення взаємопов'язаних даних крім структур та масивів використовується поняття лінійного списку. Кожен елемент такого списку містить корисні дані та покажчик (адресу) на наступний елемент списку (або порожній покажчик **null** для останнього елемента списку). Оскільки довжина списку наперед невідома, то найкраще його елементи розміщувати у динамічній пам'яті, створюючи та знищуючи їх операторами **new** та **delete**. Враховуючи неоднорідний характер даних для кожного елемента списку (покажчик та елемент іншого типу) можна представляти його з допомогою покажчика на структуру:

```
struct Item {
       Item *next;
       SomeType data;
} * pointerToItem;
```



тут `Item` – структура елемента списку, `pointerToItem` — вказівник на список, `next` – атрибут, що вказує на наступний елемент списку, `data` — корисні дані певного типу `SomeType` (наприклад, ціле число або покажчик на якийсь складніший тип даних).

Основні операції, що виконуються над списками – це:

1) пошук у списку елемента із заданою властивістю;
2) визначення *i*-го елемента у списку;
3) внесення додаткового елемента до або після вказаного вузла;
4) вилучення певного елементу зі списку;
5) упорядкування лінійного списку за певною ознакою;
6) вивід списку на екран;
7) повне знищення списку.

Всі ці операції можуть бути реалізовані за допомогою зовнішніх функцій, проте найзручніше їх зробити методами класу List. Приклад такої реалізації наведений нижче:

```
class List {
  List *next;
  int data;
public:
  List(int newData, List *oldList = nullptr) {
      data = newData;
      next = oldList;
  }
  ~List();
  void print();
};
```

Клас `List` являє собою список (фактично перший елемент — "голову" списку — який може містити покажчики на наступні елементи списку). Якщо створити статичну змінну типу `List` то отримаємо список, що складається з одного елементу, покажчик на `List`, що рівний **nullptr** (не ініціалізований) являє собою порожній список (список, в якому немає ні одного елемента).

Кожен елемент списку містить поле `data` для зберігання даних та покажчик `next` на наступний об'єкт типу `List`. Ці дані є закритими (private), тому доступ до них можуть мати тільки методи класу `List`.

Конструктор `List(…)` ініціалізує поля `data` та `next` значеннями `newData` та `oldList`, заданими в його параметрах. Оскільки він дуже простий, то записаний у вигляді inline-функції, тобто в середині опису класу. Якщо об'єкт створюється в динамічній пам'яті з допомогою

операторами **new**, то цей конструктор викликається автоматично і крім ініціалізації забезпечує ще і виділення потрібної кількості пам'яті для зберігання `data` та `next`. При виклику конструктора, другий параметр (покажчик на наступний елемент списку) можна опускати, тоді йому присвоїться значення за замовчуванням **nullptr** (див. в описі конструктора: … `List *oldList = nullptr`), тобто створиться новий список, що складається тільки з одного елемента.

Враховуючи рекурсивну природу списку, операції над ним найзручніше реалізовувати з використанням рекурсивних функцій. Так деструктор `~List()` знищує об'єкт типу `List`, звільняє зайняту ним пам'ять, а також повинен знищити всі наступні елементи списку, на який вказує покажчик `next`, якщо він не **nullptr** (тобто поки не досягнуто "хвоста" списку):

```
List::~List()
{
   if (next)
      delete next;
}
```

Тут деструктор буде викликатись рекурсивно для кожного елемента списку. Використання умови `if(next)` є обов'язковим, щоб рекурсія завершилась на останньому елементі списку ("хвості"), а не стала нескінченною.

Аналогічно реалізований рекурсивний метод `void print()`, що виводить на екран послідовно всі значення поля списку `List`, розділені символом табуляції.

```
void List::print()
{
   std::cout << data <<'\t';
   if (next)
      next->print();
}
```

Вивід здійснюється починаючи від "голови" списку до його "хвоста".

Функція `main()` для робити зі списком `LIST` може мати такий вигляд:

```
void main(){
   List *myList = new List(1);     //створити список з елементом "1"
   myList = new List(2,myList);    //додати елемент "2"
   myList = new List(3,myList);    //додати елемент "3"
   myList->print();  //вивід списку на екран
   delete myList;    //знищення списку
}
```

Тут створюється список `myList`, що складається з одного елемента зі значенням поля `data=1`, потім в його "голову" додається ще два елементи (місять числа 2 та 3), далі весь список виводиться на екран і знищується.

Результати виконання програми (числа виводять від "голови" до "хвоста"):

```
3       2       1
```

ПОРЯДОК ВИКОНАННЯ РОБОТИ

1 Реалізувати клас згідно варіанту індивідуального завдання, що містить закриті дані, а саме два типа даних: числове значення та символьний рядок, який реалізований через вказівник на char (char *). Для спрощення задачі можна замінити char * на тип даних string (на оцінку «задовільно»).

2 Реалізувати методи:

- конструктор по замовчуванню;
- параметризований конструктор;
- конструктор копіювання;
- деструктор;
- input() – метод для запит у користувача даних та їх зчитування з клавіатури у поля класу;
- print() – константний метод виводу даних на екран;
- методи доступу до закритих даних.

3 У функції main() створити декілька екземплярів класу статично і динамічно (із введенням даних із клавіатури користувачем). У звіті продемонструвати дію всіх конструкторів і методів за допомогою знімків екрану.

4 *Реалізувати клас однозв'язного списку List, який міститиме об'єкти класу, розробленого згідно варіанту індивідуального завдання. Продемонструвати роботу списку, добавивши декілька елементів, після чого вивести на екран увесь список.

ВАРІАНТИ ЗАВДАНЬ

Варіант 1.

```
class Toy {
   char *ownerName;   // власник
   int old;     // для якого віку іграшка
public:
   Toy();
   Toy( char * ownerName, int old);
   Toy( const Toy& );
   void setOwnerName( char * ownerName);
   char * getOwnerName();
   void setOld( int old );
   int  getOld( );
   void print() const;
   void input ();
   ~Toy();
};
```

Варіант 2.

```
class Animal {
   char *species;  // порода, вид
   int old;        // вік тварини
public:
   Animal();
   Animal( char * species, int old);
   Animal( const Animal& );
   void setSpecies( char * species);
   char * getSpecies( );
   void setOld( int old );
   int  getOld();
   void print() const;
   void input ();
   ~Animal();
};
```

Варіант 3.

```
class House {
   char *type;  // тип
   int numberOfRooms;  //кількість кімнат
public:
   House();
   House(char * type, int numberOfRooms);
   House(  const House& );
   void setType( char * type );
   char * getType( );
   void setNumberOfRooms(
int numberOfRooms );
   int  getNumberOfRooms ( );
   void print() const;
   void input ();
   ~House();
};
```

Варіант 4.

```
class Plant {
   char *species;  // вид
   int height;     // висота рослини
public:
   Plant();
   Plant( char * species, int height );
   Plant(  const Plant& );
   void setSpecies( char * species);
   char * getSpecies( );
   void setHeight( int height );
   int  getHeight( );
   void print() const;
   void input ();
   ~Plant();
};
```

Варіант 5.

```
class City {
   char *name;  // назва міста
   int areal;   // площа міста
public:
   City();
   City( char * name , int areal );
   City( const City& );
   void setName( char * name);
   char * getName( );
   void setAreal( int areal );
   int  getAreal( );
   void print() const;
   void input ();
   ~City();
};
```

Варіант 6.

```
class Country {
   char *title;    // назва країни
   int population; // к-сть населення
public:
   Country();
   Country(char * title, int population);
   Country( const Country& );
   void setTitle( char * title);
   char * getTitle( );
   void setPopulation ( int population );
   int  getPopulation( );
   void print() const;
   void input ();
   ~Country();
};
```

Варіант 7.

```
class Airplane {
   char *model;   // модель літака
   int power;     // потужність літака
public:
   Airplane();
   Airplane( char * model, int power );
   Airplane( const Airplane& );
   void setModel( char * model);
   char * getModel( );
   void setPower( int power );
   int  getPower( );
   void print() const;
   void input ();
   ~Airplane();
};
```

Варіант 8.

```
class Food {
   char *type;    // тип їжі
   int calories;  // к-сть калорій
public:
   Food();
   Food( char * type, int calories );
   Food( const Food& );
   void setType( char * type);
   char * getType( );
   void setCalories( int calories );
   int  getCalories( );
   void print() const;
   void input ();
   ~Food();
};
```

Варіант 9.

```
class Vegetable {
   char *color;  // колір
   int price;    // ціна
public:
   Vegetable();
   Vegetable( char * color, int price );
   Vegetable( const Vegetable& );
   void setColor( char * color );
   char * getColor( );
   void setPrice( int price );
   int  getPrice( );
   void print() const;
   void input ();
   ~Vegetable();
};
```

Варіант 10.

```
class Furniture {
   char *room;  // для якої кімнати меблі
   int weight;  // вага
public:
   Furniture();
   Furniture( char * room, int weight );
   Furniture(  const Furniture& );
   void setRoom( char * room);
   char * getRoom( );
   void setWeight( int weight );
   int  getWeight( );
   void print() const;
   void input ();
   ~Furniture();
};
```

Варіант 11.

```
class River {
   char *name;  // назва річки
   int lenght;  // довжина річки
public:
   River();
   River( char * name, int lenght);
   River( const River& );
   void setName( char * );
   char * getName( );
   void setLenght( int lenght );
   int  getLenght( );
   void print() const;
   void input ();
   ~Flat();
};
```

Варіант 12.

```
class Computer {
   char *owner;    // власник
   int processor;  // к-сть процесорів
public:
   Computer();
   Computer( char* owner,int processor);
   Computer( const Computer& );
   void setOwner( char * owner);
   char * getOwner( );
   void setProcessor( int processor);
   int  getProcessor( );
   void print() const;
   void input ();
   ~Computer();
};
```

Варіант 13.

```
class Worker {
   char *surname;  // прізвище працівника
   int payment;    // оплата за годину
public:
   Worker();
   Worker( char * surname, int payment);
   Worker( const Worker& );
   void setSurname( char * surname);
   char * getSurname( );
   void setPayment( int payment);
   int  getPayment( );
   void print() const;
   void input ();
   ~Worker();
};
```

Варіант 14.

```
class Firm {
   char *type;  // тип фірми
   int size;// розмір (к-сть працівників)
public:
   Firm();
   Firm( char * type, int size);
   Firm( const Firm& );
   void setType( char * type);
   char * getType( );
   void setSize( int size);
   int  getSize( );
   void print() const;
   void input ();
   ~Firm();
};
```

Варіант 15

```
class Clothing
{
   char *color;  // колір
   int age;    // для якого віку
public:
   Clothing();
   Clothing( char * color, int age );
   Clothing( const Clothing& );
   void setColor( char * color);
   char * getColor();
   void setAge( int age);
   int  getAge();
   void print() const;
   void input();
   ~Clothing();
};
```

Варіант 16

```
class Car
{
   char *model;  // модель машини
   int cost;     // вартість
public:
   Car();
   Car( char * model, int cost);
   Car( const Car& );
   void setModel( char * model);
   char * getModel( );
   void setCost ( int  cost);
   int  getCost ();
   void print() const;
   void input();
   ~Car();
};
```

ОФОРМЛЕННЯ ЗВІТУ:

1. Титульний лист.
2. Мета роботи.
3. Короткі теоретичні відомості.
4. Текст завдання згідно варіанту.
5. Код програми.
6. Знімок вікна виконання програми.
7. Висновки.

Контрольні запитання

1. Що таке конструктор?
2. В яких випадках використовуються конструктори?
3. Яким чином здійснюється ініціалізація об’єктів класу?
4. Чи можна замість ініціалізації динамічного об’єкту використовувати оператор присвоювання йому іншого об’єкта того ж класу?
5. Як описується конструктор?
6. Який тип результату може повертати конструктор?
7. Чи може бути конструктор віртуальною функцією?
8. Вкажіть послідовність виклику конструкторів базового, похідного класу та конструкторів об’єктів-членів класу.
9. Як можна описати реалізацію (тіло) конструктора? Наведіть приклад.
10. Що таке деструктор?
11. В яких випадках використовуються деструктори?
12. Яким чином звільнюється пам’ять, яку займають динамічні об’єкти?
13. Коли викликаються деструктори для статичних, автоматичних та динамічних об’єктів?

14. Як описується деструктор?
15. Яке ім’я може мати деструктор?
16. Який тип результату може повертати деструктор?
17. Скільки параметрів може мати деструктор?
18. Чи може бути деструктор віртуальною функцією?
19. В якому порядку викликаються деструктори базового, похідного класу та об’єктів-членів класу.
20. Як можна описати реалізацію (тіло) деструктора? Наведіть приклад.
21. Які оператори використовуються для створення та знищення динамічних об’єктів?
22. Що відбувається при виклику оператора new для об’єкта?
23. Що відбувається при виклику оператора delete для об’єкта?
24. Що таке лінійний список?
25. Які типи даних використовуються для реалізації динамічного лінійного списку?
26. Які основні операції виконуються над списками?
27. Що містить у собі список, реалізований у вигляді класу?
28. Які методи повинен містити клас, що реалізує динамічний список?
29. Які дії повинен виконувати конструктор класу, що реалізує динамічний список?
30. Які дії повинен виконувати деструктор класу, що реалізує динамічний список?
31. Який вид функцій найзручніше використовувати для реалізації методів класу “Динамічний список”?
32. Наведіть приклад методу, що виводить на екран значення елементів динамічного списку.