Введение в современные компьютерные технологии

Практикум 6

Дрюк Андрей

Немнюгин Сергей Андреевич

Санкт-Петербургский государственный университет

2022

Динамическое выделение памяти операторы new, delete, delete[]

Динамическое выделение памяти — это способ запроса памяти из операционной системы запущенными программами по мере необходимости.

```
// создаем вне нашей программы переменную int
// new возвращает указатель на адрес переменной

int* px = new int(5);
cout << "px = " << px << endl;
cout <<"*px = " << *px << endl;
delete px;
```

```
 // создаем вне нашей программы массив на 10 элементов
    // new возвращает указатель на адрес переменной
    int size = 5;
    int* myArray = new int[size];
    cout << "px = " << px << endl;
    for (int i = 0; i < size; ++i) {
        cout << "&myArray[" << i << "] = " <<
              &myArray[i] << endl;
    }
    delete[] myArray;
```

Динамическое выделение памяти операторы new, delete, delete[]

Динамическое выделение памяти — это способ запроса памяти из операционной системы запущенными программами по мере необходимости.

```
px = 000001B35224DE10
*px = 5
```

```
myArray = 00000161A98019B0
&myArray[0] = 00000161A98019B0
&myArray[1] = 00000161A98019B4
&myArray[2] = 00000161A98019B8
&myArray[3] = 00000161A98019BC
&myArray[4] = 00000161A98019C0
```

```
myArray[0] = -842150451
myArray[1] = -842150451
myArray[2] = -842150451
myArray[3] = -842150451
myArray[4] = -842150451
```

Динамическое выделение памяти операторы new, delete, delete[]

```
#include <iostream>
#include <ctime>
using namespace std;
void FillDynamicArray(int* arr, int size)
{
    for (int i = 0; i < size; ++i) {
        arr[i] = rand() % 10;
    }
}
void PrintDynamicArray(int* arr, int size)
{
    for (int i = 0; i < size; ++i) {
        cout << arr[i] << endl;
    }
}
```

```
int main()
{
    srand(time(NULL));
    int mySize = 10;
    int* myDArray = new int[mySize];
    FillDynamicArray(myDArray, mySize);
    PrintDynamicArray(myDArray, mySize);

    delete[] myDArray;
}
```

Работа с двумерными динамическими массивами

```
int rows = 4;
int columns = 5;

int** myArray = new int*[rows];
for (int i = 0; i <rows; ++i) {
   myArray[i] = new int[columns];
   for (int j = 0; j < columns; ++j) {
      myArray[i][j] = rand() % 10;
   }
}
for (int i = 0; i < 10; ++i) {
   delete[] myArray[i];
}
delete[] myArray;
```

Копирование динамического массива
Проблема

```
int main()
{
    int size = 4;
    int* a = new int[size];
    FillWithOnes(a, size);
    int* b = a;
    cout << "b[0] = " << b[0] << endl;
    FillWithZeros(a, size);
    cout << "b[0] = " << b[0] << endl;
}
```

Копирование динамического массива

```
void copyArray(int* arr1, int *arr2, int size) {
      for (int i = 0; i < size; ++i) {
      arr2[i] = arr1[i];
      }
}
```

```
int size = 4;
int* a = new int[size];
FillWithOnes(a, size);
int* b = new int[size];
copyArray(a, b, size);
cout << "b[0] = " << b[0] << endl;
FillWithZeros(a, size);
cout << "b[1] = " << b[0] << endl;
```

Задание

1. Создать новый проект DynamicArray
2. Создать переменные size – для одномерного массива, rows and columns для двумерного
3. Создать функции вывода на экран этих массивов
4. Создать функции заполнения этих массивов, двумерный массив должен быть напечатан как матрица rows x columns

Утечка памяти

```
int size = 5;



 //Утечка памяти, открыть диспетчер задач!
    for (int i = 0; i < size*10000000; ++i)
{
        char* myArray1 = new char;
        //delete myArray1;
}
```

Введение в ООП

1. **Недостатки структурного программирования:**

a) **Неограниченность доступа функций к глобальным переменным**

b) **Отделение данных от функций плохо отображает реальную картину мира (разделение на данные/свойство и функции/поведение)**

2. Идея ООП - объединить данные и действия над этими данными в единое целое – объект.

Введение в ООП

1. ***Класс***-универсальный, комплексный тип данных, состоящий из набора полей и методов.

```
class Figure{
//данные, поля, состояние
public:
  int x;
  int y;

public:
// методы
setters
  void setX(int x1) {
      x = x1;
   }
  void setY(int y) {
      this->y = y;
  }

//getters
   int getX() {
       return x;
   }
   int getY() {
      return y;
   }
};
```

Объект

2.***Объект***-сущность в адресном пространстве, появляющаяся при создании экземпляра класса.

Попробуем создать объект:

```
int main()
{
    Figure figure;
    cout << figure.x << endl;
}
```

Конструктор

Чтобы настроить, как класс инициализирует свои члены или вызывать функции при создании объекта класса, определите *конструктор*. Конструкторы имеют имена, совпадающие с именами классов, и не имеют возвращаемых значений. Можно определить столько перегруженных конструкторов, сколько необходимо для настройки инициализации различными способами. Как правило, конструкторы имеют открытый доступ, поэтому код за пределами определения класса или иерархии наследования может создавать объекты класса.

Конструктор

```
public:
//constructor
   Figure() {
      x = 0;
      y = 0;
   }
```

```
int main()
{
   Figure figure;
   cout << figure.x << endl;
   cout << figure.y << endl;
}
```

```
Figure(int x1, int  y1) {
   x = x1;
   y = y1;
}
```

```
int main()
{
   Figure figure(10, 11);
   cout << figure.x << endl;
   cout << figure.y << endl;
}
```

Наследование

В классе Figure создадим метод:

```
virtual void printInfo() {
    cout << "=========PrintInfo in Figure============" << endl;
    cout << "Figure x = " << getX() << " y = " << getY() << endl;
}
```

```
int main()
{
    Figure* figure = new Figure(10, 11);
    figure->printInfo();
    delete figure;
}
```

Наследование

```
class Circle: public Figure{
public:
   double r;

public:
   Circle() {
      x = 0;
      y = 0;
      r = 1;
   }
   Circle(int x1, int y1, double r) {
      x = x1;
      y = y1;
      this->r = r;
   }
};
```

Задание

В классе Circle написать реализацию метода getArea(), а также свою реализацию метода printInfo()

Полиморфизм

В объектном программировании под полиморфизмом подразумевается наличие кода, написанного с использованием ссылок (указателей), имеющих тип базового класса иерархии. При этом такой код должен правильно работать для любого объекта, являющегося экземпляром класса из данной иерархии, независимо от того, где этот класс расположен в иерархии. Такой код и называется полиморфным. При написании полиморфного кода заранее неизвестно, для объектов какого типа он будет работать — один и тот же метод будет исполняться по-разному в зависимости от типа объекта.

Полиморфизм

```
int main()
{
    Figure* figure = new Figure(10, 11);
    Circle* circle = new Circle(10, 11, 5);
    Figure* circle1 = new Circle(0, 0, 5);
    figure->printInfo(); //метод figure
    circle->printInfo(); //метод у circle
    circle1->printInfo(); //метод у circle
    delete figure;
    delete circle;
    delete circle1;
}
```

Задание “Класс комплексных чисел” Task5.

1. Поля: Вещественная и мнимая часть
2. Методы: сложение, вычитание, умножение, деление, print(). Задать конструкторы

Все методы, кроме print() возвращают новое комплексное число