Введение в современные компьютерные технологии

Практикум 5

Дрюк Андрей

Немнюгин Сергей Андреевич

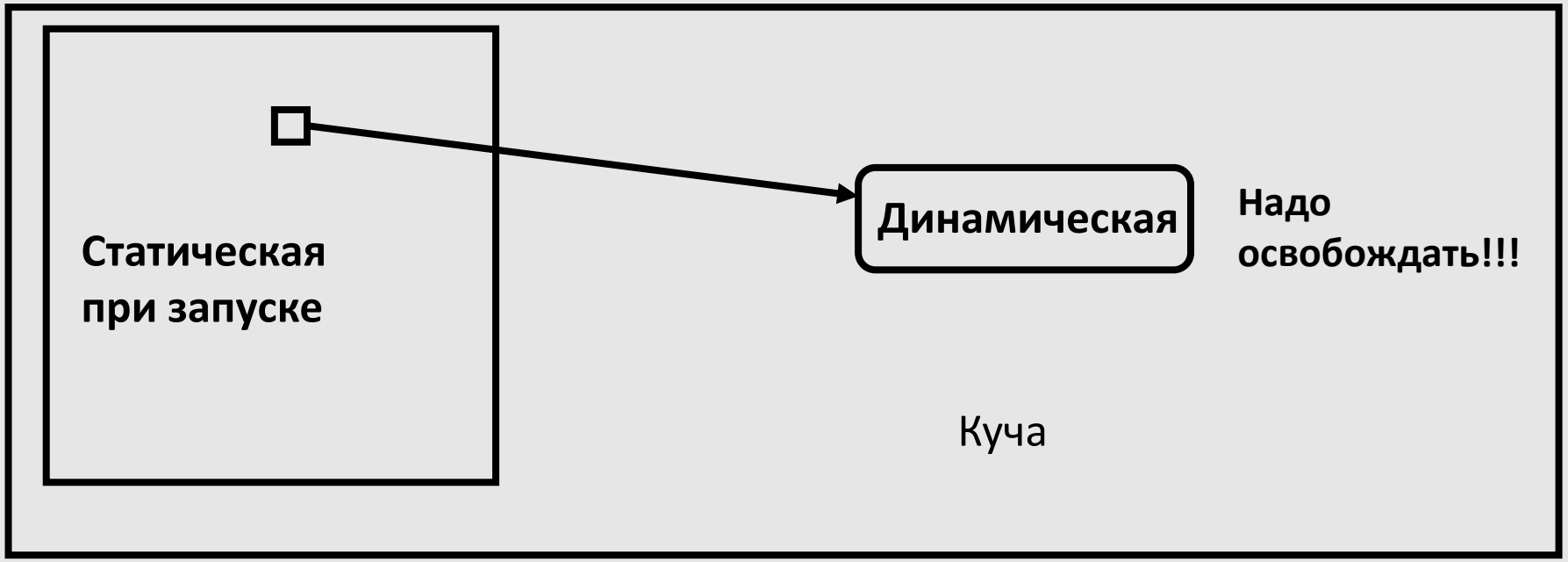
Санкт-Петербургский государственный университет

2022

Статическое и динамическое выделение памяти

Статическое выделение памяти выполняется для статических и глобальных переменных. Память выделяется один раз (при запуске программы) и сохраняется на протяжении работы всей программы.

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



Указатель

Указатель - это переменная, содержащая адрес другой переменной (объекта)

```
int a = 5;
int b = a;

int *pa = &a; //объявляем указатель типа int

cout << a << endl;
cout << pa << endl; // адрес в памяти
cout << *pa << endl; //разыменование указателя, получаем реальные данные
```

Передача по указателю

```
void Foo(int* pa) {// объявляем функцию, в качестве параметра
передаем указатель!!!
    (*pa)++; // разыменовываем указатель!!!!
}

int main()
{
    int a = 5;
    cout << "a before: " << a <<endl;
    Foo(&a);// Операция взятия адреса (передаем адрес
(указатель) в функцию)
    cout << "a after: " << a << endl;
}
```

Ссылка

Ссылка – переменная, которая хранит в себе адрес другой переменной

```
int a = 5;

int* pa = &a; //указатель
int& aRef = a; // ссылка, нет дополнительного оператора взятия адреса
// int& bRef; // ошибка, обязательно надо инициализировать!!!

cout << "a = " << a << endl;
cout << "pa = "<< pa << endl;
cout << "aRef = " << aRef << endl; // нет дополнительного разыменования
```

Передача по ссылке

```
void Foo(int& pa) {// объявляем
    pa++; // инкрементируем
}

int main()
{
    int a = 5;
    cout << "a before: " << a <<endl;
    Foo(a);//
    cout << "a after: " << a << endl;
}
```

```
void Foo(int* pa) {// объявляем функцию
   (*pa)++; // разыменовываем указатель!
}

int main()
{
    int a = 5;
    cout << "a before: " << a <<endl;
    Foo(&a);
    cout << "a after: « << a << endl;
}
```

Отличия ссылки от указателя

1. Указатели могут быть равными NULL, в то время как ссылка всегда указывает на определенный объект.
2. Работа с ссылками не требует разыменования!
3. Указатель может быть переназначен любое количество раз, в то время как ссылка после привязки не может быть перемещена на другую ячейку памяти.
4. Для ссылок отсутствует арифметика ссылок

```
int a = 5;
int *pa = &a;
cout << "pa = " << pa << " *pa =  " << *pa << endl;
pa++;
cout << "pa = " << pa << " *pa =  " << *pa << endl;
```

Статические массивы

Массив — это последовательность объектов одного и того же типа, которые занимают смежную область памяти.

Пример:

```
int a[4];
bool b[4];
for (int i = 0; i < 4; ++i) {
    cout << "address of a[" << i << "]: " << &a[i] << endl;
}

for (int i = 0; i < 4; ++i) {
    cout << "address of b[" << i << "]: " << &b[i] << endl;
}
```

Статические массивы

Int 4 байта

address of a[0]: 60E7EFF9A8
address of a[1]: 60E7EFF9AC
address of a[2]: 60E7EFF9B0
address of a[3]: 60E7EFF9B4

Bool 1 байта

address of b[0]: 60E7EFF9D4
address of b[1]: 60E7EFF9D5
address of b[2]: 60E7EFF9D6
address of b[3]: 60E7EFF9D7

Статические массивы

Массив — это последовательность объектов одного и того же типа, которые занимают смежную область памяти.

```
int a[4];// в элементах мусор
bool b[4];
cout <<"a = "<< a << endl;
cout << "b = " << b << endl;
```

a = 00000008904FF508

address of a[0]: 00000008904FF508

b = 00000008904FF534

address of b[0]: 00000008904FF534

Инициализация статические массивы

1. Нумерация массива с 0 !!!!!!! – для цикла for

```
int a[4] = { 1,2,3,4 };

int b[4];// поэлементная запись
b[0] = 1;
b[1] = 2;
b[2] = 3;
b[3] = 4;

int c[] = { 1,2,3,4,5,6,7 };//старая запись
int d[]{ 1,2,3,4,5,6,7 };// новая запись
int e[5]{}; // инициализируем 0
```

Инициализация статические массивы

```
const int size = 100; //без const недопустимо,

int newArray[size]{};
```

Передача массива в функцию.

1. Массивы передаются по указателю!!!

```
void fillArray(double array[], const int size) {

    for (int i = 0; i < size; ++i) {
        array[i] = 1.0*rand()/RAND_MAX;
    }

}

void fillArray(double* array, const int size) {

    for (int i = 0; i < size; ++i) {
        array[i] = 1.0 * rand() / RAND_MAX;
    }

}
```

Двумерный массив – массив массивов

```
const int ROWS = 2; //обязательно const
const int COLS = 2;
int a[ROWS][COLS]{};
```

```
cout << "a = " << a << endl;
cout << "a[0] = " << a[0] << endl;
cout << "a[1] = " << a[1] << endl;
cout << "a[0][0] = " << a[0][0] << endl;
cout << "a[0][1] = " << a[0][1] << endl;
cout << "a[1][0] = " << a[0][1] << endl;
cout << "a[1][1] = " << a[1][1] << endl;
```

```
a = 000000E8925CF878
a[0] = 000000E8925CF878
a[1] = 000000E8925CF880
a[0][0] = 0
a[0][1] = 0
a[1][0] = 0
a[1][1] = 0
```

Двумерный массив

```
cout << "a = " << a << endl;
cout << "a[0] = " << a[0] << endl;
cout << "a[1] = " << a[1] << endl;
cout << “&a[0][0] = " << &a[0][0] << endl;
cout << “&a[0][1] = " << &a[0][1] << endl;
cout << “&a[1][0] = " << &a[1][0] << endl;
cout << “&a[1][1] = " << &a[1][1] << endl;
```

```
a = 000000AFEDCFF688
a[0] = 000000AFEDCFF688
a[1] = 000000AFEDCFF690
&a[0][0] = 000000AFEDCFF688
&a[0][1] = 000000AFEDCFF68C
&a[1][0] = 000000AFEDCFF690
&a[1][1] = 000000AFEDCFF694
```

Передача двумерного массива

```
Так запрещено!!!!
void FillArray(int arr[][], const int ROWS, const int COLS) {
    for (int i = 0; i < ROWS; ++i) {
        for (int j = 0; j < COLS; ++j) {
            arr[i][j] = rand() % 10;
        }
    }
}
```

Решение:

1. Использовать в качестве параметров глобальные переменные

Передача двумерного массива

```
const int COLS = 4;

void FillArray(int arr[][COLS], const int
ROWS) {
    for (int i = 0; i < ROWS; ++i) {
        for (int j = 0; j < COLS; ++j) {
            arr[i][j] = rand() % 10;
        }
    }
}

int main()
{
    srand(time(NULL));
    const int ROWS = 3;
    int myArray[ROWS][COLS];
    FillArray(myArray, ROWS);
}
```

```
void FillArray(int arr[3][4])
void FillArray(int arr[ROWS][COLS]
```

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 < 10; ++i) {
   myArray[i] = new int[columns];
   for (int j = 0; j < 10; ++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[1] = " << 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

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

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 Circle{

};
```

```
class Circle {
    //Поля
public:
    double x;
    double y;
    double r;
}
```

```
class Circle {
    //Поля
public:
    double x;
    double y;
    double r;

public:
    //constructors
    Circle() {
        cout << "Standart Constructor: " << endl;
        x = 0; y = 0; r = 1;
    }
    Circle(double x1, double y1, double r1){
        cout << "Special Constructor " << endl;
        x = x1; y = y1; r = r1;
    }
//method
void show() {
        cout << "Круг показался в x = "<<x<<" y = " << y << endl;
   }
};
```

Объект

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

```
Circle circle;
Circle *circle1 = new Circle(1,2,10);
```

Вывод:

Standart Constructor:
Special Constructor