9-11 классы

Программирование
на C++

Презентация занятия

Статическая и динамическая память.

7 занятие

КОД БУДУЩЕГО

Минцифры России

инжиниум МГТУ им. Н.Э. Баумана

НАЦИОНАЛЬНЫЕ ПРОЕКТЫ РОССИИ ЦИФРОВАЯ ЭКОНОМИКА

20.35 УНИВЕРСИТЕТ

2020

Программирование
на C++

Теоретическая часть

Статическая и динамическая память.

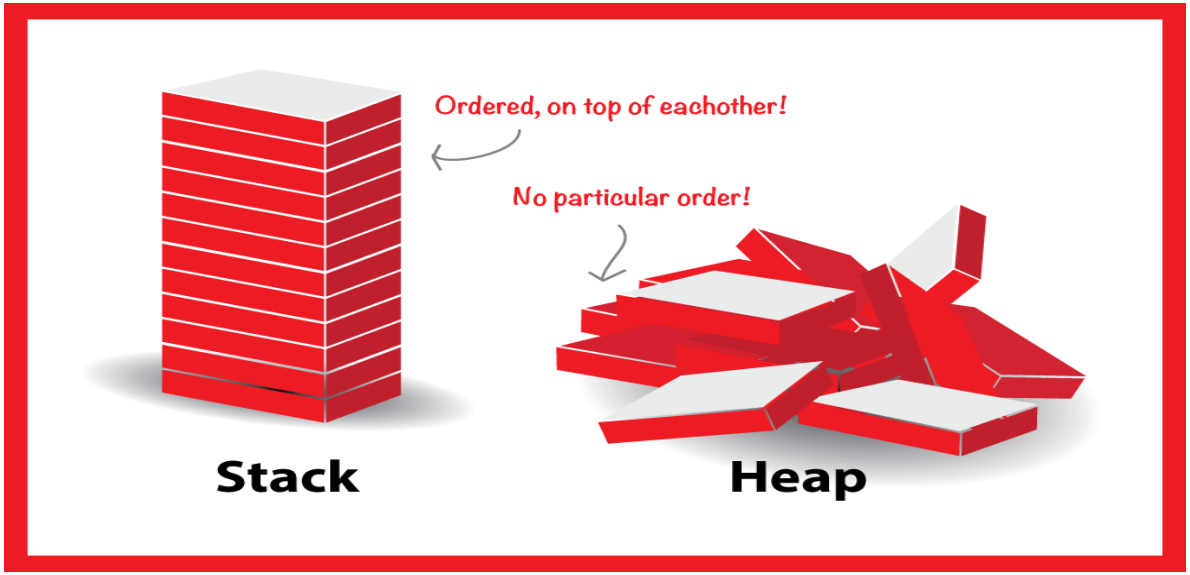
7 занятие

инжиниум®
МГТУ им. Н.Э. Баумана

2020

Тема: Статическая и динамическая память

В программах на языке C++, память разделена на 2 части:
Стек (stack): все локальные переменные размещаются в памяти стека.
Куча (heap): Неиспользованная программой память, которая может быть использована, когда программа **динамически** выделяет память.
Часто, вы не будете знать сколько памяти вам понадобится для хранения информации в определённых переменных и объём необходимой памяти будет определён при запуске программы. В таком случае вы можете выделить память во время выполнения программы с помощью кучи для переменных заданного типа используя оператор **new**, который возвращает адрес выделенной памяти.



Тема: Статическая и динамическая память

Выделенный адрес может быть сохранён в **указателе**, который в последствии может быть разадресован (разименован) для доступа к переменной (её значению)

```
int *p = new int;
*p = 5;
```

Имеем динамически выделенную память для целых чисел, а затем ей присвоено значение 5.
Указатель p хранится в стеке как локальная переменная и хранит выделенный в куче адрес. Значение 5 хранится по этому адресу в куче

Тема: Статическая и динамическая память

Стек

Для локальных переменных в **стеке**, управление памятью осуществляется автоматически. В **куче**, необходимо вручную управлять динамически выделенной памятью и использовать оператор **delete** для освобождения памяти, когда в ней больше нет необходимости.

```
delete pointer;
```

Это выражение освобождает память, на которую указывает pointer.

```
int *p = new int; // request memory
*p = 5; // store value

cout << *p << endl; // use value

delete p; // free up the memory
```

Тема: Статическая и динамическая память

Если забывать освобождать выделенную (динамически) память, то образуются утечки памяти, потому что эта память остаётся выделенной, пока программа не будет закрыта.
Оператор **delete** освобождает память, выделенную для переменной, но не удаляет сам указатель (который хранит только адрес объекта), т к сам указатель хранится в стеке.

Указатель NULL - это константа со значением равным нулю, которая определена в нескольких стандартных библиотеках.

Присваивание NULL указателю при его объявлении является хорошей практикой, в случае если у вас нет точного адреса для присваивания.

NULL

Тема: Статическая и динамическая память

Динамическая память также может быть выделена для массива

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int *p = NULL; // Pointer initialized with null
7       p = new int[20]; // Request memory
8       delete [] p; // Delete array pointed to by p
9
10      return 0;
11  }
```

Динамическое выделение памяти полезно во многих случаях, к примеру, когда вашей программе необходимо получить изображение, но она не знает его возможный размер и количество памяти, необходимое для его хранения

Тема: Статическая и динамическая память

Многомерные массивы

Многомерный массив хранит один и более массивов

Например, объявление двумерного массива целых чисел выглядит следующим образом.

```
int x[3][4];
```

Такую конструкцию удобно представлять себе в виде таблицы, состоящей из 3 строк и 4 столбцов. (Помните, что индексирование массивов начинается с 0)

	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

Тема: Статическая и динамическая память

Многомерные массивы могут быть инициализированы с использованием заключения внутри специальных скобок для каждой строки

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       cout << "Output:" << endl;
7       int x[2][3] = {{2, 3, 4}, {8, 9, 10}};
8       cout << x[0][2] << endl;
9
10      return (0);
11  }
```

```
Output:
4
```

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       cout << "Output:" << endl;
7       int **array;
8       int i;
9
10      array = new int* [2];
11      for (i = 0; i < 2; i++)
12      {
13          array[i] = new int[3];
14      }
15      array[0][2] = 3;
16      cout << array[0][2]<< endl;
17      return (0);
18  }
```

```
Output:
3
```

Тема: Статическая и динамическая память

Случайные числа (Псевдослучайные)

Возможность генерировать **случайные** числа очень полезна во многих ситуациях, включая создания игр, программ статического моделирования и подобных продуктов.

В стандартной библиотеке C++ определена функция **rand()**, которая возвращает псевдослучайное число. Для её использования необходимо с помощью директивы (инструкции) препроцессора подключить заголовочный файл **<cstdlib>**

```
1   #include <iostream>
2   #include <cstdlib>
3   using namespace std;
4
5   int main()
6   {
7       int i = 0;
8
9       cout << "Output:" << endl;
10      while (i < 10)
11      {
12          cout << rand() << endl;
13          i++;
14      }
15      return 0;
16  }
```

```
Output:
16807
282475249
1622650073
984943658
1144108930
470211272
101027544
1457850878
1458777923
2007237709
```

Тема: Статическая и динамическая память

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

int main()
{
    cout << "Output:" << endl;
    int **array;
    int i, j;

    array = new int* [2];
    for (i = 0; i < 2; i++)
    {
        array[i] = new int[3];
    }
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 3; j++)
        {
            array[i][j] = rand();
        }
    }
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 3; j++)
        {
            cout << array[i][j] << " ";
        }
        cout << endl;
    }
    return (0);
}
```

#include <cstdlib>

```
Output:
7 9 3
8 0 2
```

```
Output:
16807 282475249 1622650073
984943658 1144108930 470211272
```

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

int main()
{
    cout << "Output:" << endl;
    int **array;
    int i, j;

    array = new int* [2];
    for (i = 0; i < 2; i++)
    {
        array[i] = new int[3];
    }
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 3; j++)
        {
            array[i][j] = rand() % 10;
        }
    }
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 3; j++)
        {
            cout << array[i][j] << " ";
        }
        cout << endl;
    }
    return (0);
}
```

Программирование
на C++

Практическая часть

Статическая и динамическая память.

7 занятие

инжиниум®
МГТУ им. Н.Э. Баумана

2020

Тема: Статическая и динамическая память

Задание 1

Создайте матрицу (двумерный массив) 5 на 5, выделив для него память динамически. Заполните все элементы нулями и выведите на экран

*

Вынесете в отдельную функцию вывод матрицы (двумерного массива) на экран (стандартный вывод), выделение памяти для матрицы и присваивание случайных значений элементам матрицы

**

Напишите функцию сложения двух матриц (двумерных массивов)
Каждый элемент первого массива складывается с соответствующим элементом (имеющим такие же индексы) второго массива.

***

Написать функцию, контролирующую утечки памяти (удаляющую динамически выделенную память)

Тема: Статическая и динамическая память

```
/*
**  Выделение памяти для квадратной матрицы размером size на size
*/
int     **matrix_allocator(int **matr, const int size)
{
    int     counter = 0;

    matr = new int* [size];
    while (counter < size)
    {
        matr[counter] = new int [size];
        counter++;
    }
    return (matr);
}
```

Тема: Статическая и динамическая память

```
/*
**  Присваивание рандомных (случайных) чисел матрице, используя функцию rand
**  #include <csdtdlib> для rand()
**  Используем оператор '%' для ограничения диапазона рандомных (случайных
чисел)
*/
int     **matrix_initializer(int **matr, const int size)
{
    int     i = 0, j = 0;

    while (i < size)
    {
        j = 0;
        while (j < size)
        {
            matr[i][j] = rand() % 5;
            j++;
        }
        i++;
    }
    return (matr);
}
```

Тема: Статическая и динамическая память

```
/*
**  Вывод матрицы в стандартный вывод (печать на экран) размером size на size
*/
void    matrix_print(int **matr, const int size)
{
    int     i = 0, j = 0;

    cout << "Matrix " << size << " x " << size << ":" << endl;
    while (i < size)
    {
        j = 0;
        while (j < size)
        {
            cout << matr[i][j] << " ";
            j++;
        }
        i++;
        cout << endl;
    }
    cout << endl;
}
```

Тема: Статическая и динамическая память

```
/*
**  Суммирование двух матриц (с динамическим выделением памяти для
результата)
*/
int     **matrix_summer(int **matr_1, int **matr_2, const int size)
{
    int     **res;
    int     i = 0, j = 0;

    res = matrix_allocator(res, size);
    while(i < size)
    {
        j = 0;
        while(j < size)
        {
            res[i][j] = matr_1[i][j] + matr_2[i][j];
            j++;
        }
        i++;
    }
    return (res);
}
```

Тема: Статическая и динамическая память

```
/*
**  Удаление динамически выделенной памяти для матрицы размером size на size
*/
void    matrix_delete(int **matr, const int size)
{
    int     counter = 0;

    while (counter < size)
    {
        delete [] matr[counter];
        counter++;
    }
    delete [] matr;
}
```

Тема: Статическая и динамическая память

```
int main()
{
    //  Декларирование переменных для матриц и размера
    int             **matrix_1, **matrix_2, **matrix_sum;
    const int       size_of_matrix = 5;

    //  Выделение памяти для первой матрицы
    matrix_1 = matrix_allocator(matrix_1, size_of_matrix);
    //  Присваивание случайных чисел первой матрице
    matrix_1 = matrix_initializer(matrix_1, size_of_matrix);
    //  Вывод первой матрицы на экран
    cout << "First matrix:" << endl;
    matrix_print(matrix_1, size_of_matrix);

    //  Выделение памяти для второй матрицы
    matrix_2 = matrix_allocator(matrix_2, size_of_matrix);
    //  Присваивание случайных чисел второй матрице
    matrix_2 = matrix_initializer(matrix_2, size_of_matrix);
    //  Вывод второй матрицы на экран
    cout << "Second matrix:" << endl;
    matrix_print(matrix_2, size_of_matrix);

    //  Суммирование двух матриц
    matrix_sum = matrix_summer(matrix_1, matrix_2, size_of_matrix);
    //  Вывод рузультата на экран
    cout << "Result of sum:" << endl;
    matrix_print(matrix_sum, size_of_matrix);

    //  Освобождение ранее выделенной динамически памяти
    matrix_delete(matrix_1, size_of_matrix);
    matrix_delete(matrix_2, size_of_matrix);
    matrix_delete(matrix_sum, size_of_matrix);

    return 0;
}
```