|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |
| Институт кибернетики | | |
| Кафедра программного обеспечения систем радиоэлектронной аппаратуры | | |

ЛАБОРАТОРНАЯ РАБОТА

ОСОБЕННОСТИ РАСПОЛОЖЕНИЯ ОБЪЕКТОВ ПРОГРАММЫ В ПАМЯТИ

|  |  |
| --- | --- |
| Обучающийся | Дугин И. А. |
| Шифр | 19К0198 |
| Направление подготовки | 01.03.02  Прикладная математика и информатика |
| Группа | КМБО-02-19 |

Научный руководитель­­­­­ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Г. Москва

**Тема**: особенности расположения объектов программы в памяти.

**Цель**: определение закономерностей размещения объектов программы в оперативной памяти.

**Задачи**:

1) определить, как расположены в памяти *глобальные, автоматические* и *динамические*переменные;

2) определить принцип расположения в памяти *одномерного статического массива*, *одномерного динамического массива*, *двумерного статического массива* и *двумерного динамического массива*;

3) определить, как расположены в памяти поля объектов структур (с различным упорядочиванием полей) для структур *с выравниванием*и*без выравнивания*. Сравнить размеры объектов структур с выравниванием и без (для каждой структуры вычислять размер непосредственно как сумму размеров полей и как *sizeof*всей структуры).

**Инструменты работы:** На ОС Windows будем использоваться GNU GCC Compiler, на ОС Linux g++.

**ПРАКТИЧЕСКАЯ ЧАСТЬ**

Для решения данной лабораторной работы использовались:

* В ОС Windows используется IDE Сode Blocks 2019, компилятор GNU GCC Compiler, разрядностью x32.
* В OC Linux используется текстовый редактор vim, компилятор g++, разрядность 64 бит.

**ЗАДАНИЕ 1**

Определить, как расположены в памяти глобальные, автоматические и динамические переменные.

**Ход работы:**

1. сначала в программе происходит создание переменных трёх типов:
   * глобальной;
   * автоматической;
   * динамической.
2. Затем происходит вывод адреса переменных трёх типов. (т.е. адреса ячеек, начиная с которых хранятся те или иные переменные).
3. Затем, чтобы избежать потерю памяти, происходит удаление динамических переменных.

|  |  |  |
| --- | --- | --- |
|  | Windows | Linux |
| Запуск 1  Global  Automatic  Dynamic | 0x4c6008  0x28ff0c  0x36af70 | 0x601174  0x7fff04d353ec  0x7fff04d353d8 |
| Запуск 2  Global  Automatic  Dynamic | 0x4c6008  0x28ff0c8  0x60af70 | 0x601178  0x7ffee9dd04a8  0x7ffee9dd0490 |
| Запуск 3  Global  Automatic  Dynamic | 0x4c6008  0x28ff04  0x7baf70 | 0x60117c  0x7fffb09b6584  0x7fffb09b6568 |

**Вывод 1:**

На OC Windows динамическая/глобальная переменная находится ближе к нулевому байту, затем идет глобальная/динамическая, затем автоматическая. Также, глобальные и автоматические переменные при каждом новом запуске практически не меняют свой адрес, что нельзя в точности сказать про OC Linux.

В OC Linux ближе к нулевому байту хранится глобальная переменная, затем - динамическая, автоматическая.

**ЗАДАНИЕ 2**

Определить принцип расположения в памяти одномерного статического массива, одномерного динамического массива, двумерного статического массива и двумерного динамического массива.

**Ход работы:**

Происходит создание массивов четырёх типов:

* одномерный статический (размер n);
* двумерный статический (размер n\*n);
* одномерный динамический (размер n);
* двумерный динамический (размер n\*n).

Затем происходит вывод адресов элементов массивов (т.е. адреса ячеек, начиная с которых хранятся те или иные переменные), таким образом происходит анализ взаимного расположения переменных друг относительно друга.

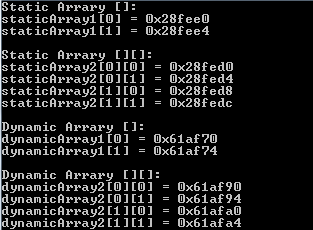
Затем, чтобы избежать потерю памяти, происходит удаление динамических массивов.

|  |  |  |
| --- | --- | --- |
| **Одномерный статический** | **Windows** | **Linux** |
| staticArray1[0] | 0x28fee0 | 0x7fffe2237e80 |
| staticArray1[1] | 0x28fee4 | 0x7fffe2237e84 |

|  |  |  |
| --- | --- | --- |
| **Одномерный динамический** | **Windows** | **Linux** |
| dynamicArray1[0] | 0x61af70 | 0xc57010 |
| dynamicArray1[1] | 0x61af74 | 0xc57014 |

|  |  |  |
| --- | --- | --- |
| **Двумерный статический** | **Windows** | **Linux** |
| staticArray2[0][0] | 0x28fed0 | 0x7fffe2237e70 |
| staticArray2[0][1] | 0x28fed4 | 0x7fffe2237e74 |
| staticArray2[1][0] | 0x28fed8 | 0x7fffe2237e78 |
| staticArray2[1][1] | 0x28fedc | 0x7fffe2237e7с |

|  |  |  |
| --- | --- | --- |
| **Двумерный динамический** | **Windows** | **Linux** |
| dynamicArray2[0][0] | 0x61af90 | 0xc57050 |
| dynamicArray2[0][1] | 0x61af94 | 0xc57054 |
| dynamicArray2[1][0] | 0x61afa0 | 0xc57070 |
| dynamicArray2[1][1] | 0x61afa4 | 0xc57074 |

****

**Вывод 2:**

На ОС Windows и строки, и элементы в строках статического массива располагаются последовательно. В динамическом строки располагаются в различных ячейках памяти, а элементы в строках располагаются последовательно. На OC Linux элементы массивов располагаются так же, как и на OC Windows, но ближе к нулю находятся динамические массивы, на OC Windows наоборот.

На OC Linux и на OC Windows элементы статического одномерного, двумерного массива и элементы динамического одномерного массива хранятся последовательно, в отличие от элементов динамического двумерного массива, которые располагаются с определённым интервалом.

**ЗАДАНИЕ 3**

Определить, как расположены в памяти поля объектов структур (с различным упорядочиванием полей) для структур с выравниванием и без выравнивания. Сравнить размеры объектов структур с выравниванием и без (для каждой структуры вычислять размер непосредственно как сумму размеров полей и как sizeof всей структуры).

**Ход работы:**

1. Для решения данного задания необходимо создать четыре структуры:
   * упорядоченная с выравниванием;
   * упорядоченная без выравнивания;
   * не упорядоченная с выравнивания;
   * не упорядоченная без выравнивания

с 4тырьмя переменными разных типов (bool, short, int, double).

1. Производим выравнивание в structFirst и structSecond.
2. Затем выводятся адреса переменных каждой из структур, их размер (по-отдельности), а также реальный размер структуры в целом.

**Windows:**

|  |  |
| --- | --- |
| Упорядоченная,  с выравнивания (structSecond) | Упорядоченная,  без выравниванием (structFourth) |
| **bool - 0x28fde0**  **short - 0x28fde2**  **int - 0x28fde4**  **double - 0x28fde8** | **bool - 0x28fdc1**  **short - 0x28fdc2**  **int - 0x28fdc4**  **double - 0x28fdc8** |
| Размер структуры: 16  Сумма полей: 15 | Размер структуры: 15  Сумма полей: 15 |

|  |  |
| --- | --- |
| Не упорядоченная,  с выравнивания (structFirst) | Не упорядоченная,  без выравниванием (structThird) |
| **bool - 0x28fdf0**  **int - 0x28fdf4**  **short - 0x28fdf8**  **double - 0x28fe00** | **bool - 0x28fdd8**  **int – 0x28fdd9**  **short - 0x28fddd**  **double - 0x28fdd0** |
| Размер структуры: 24  Сумма полей: 15 | Размер структуры: 15  Сумма полей: 15 |

**Linux:**

|  |  |
| --- | --- |
| Упорядоченная,  с выравнивания (structSecond) | Упорядоченная,  без выравниванием (structFourth) |
| **bool - 0x7fffb56e4360**  **short - 0x7fffb56e4364**  **int - 0x7fffb56e4364**  **double - 0x7fffb56e4368** | **bool - 0x7fffb56e4340**  **short - 0x7fffb56e4341**  **int - 0x7fffb56e4343**  **double - 0x7fffb56e4347** |
| Размер структуры: 16  Сумма полей: 15 | Размер структуры: 15  Сумма полей: 15 |

|  |  |
| --- | --- |
| Не упорядоченная,  с выравнивания (structFirst) | Не упорядоченная,  без выравниванием (structThird) |
| **bool - 0x7fffb56e4370**  **int - 0x7fffb56e4374**  **short - 0x7fffb56e4378**  **double - 0x7fffb56e4380** | **bool - 0x7fffb56e4358**  **int - 0x7fffb56e4359**  **short - 0x7fffb56e435d**  **double - 0x7fffb56e4350** |
| Размер структуры: 24  Сумма полей: 15 | Размер структуры: 15  Сумма полей: 15 |

**Вывод 3:**

Размер структуры без выравнивания равен сумме размеров каждой отдельной переменной, входящей в это структуру. Её размер будет меньше или равен упорядоченной структуре с выравниванием. Также, можно заметить, что упорядочивание элементов от меньшего к большему снижает размер использующейся памяти, а результаты на OC Windows и OC Linux не различаются.

**4 ВЫВОД**

На OC Windows глобальная переменная находится ближе к нулевому байту, затем идет динамическая, затем локальная. Также, можно заметить, что автоматическая\локальная и динамическая переменные имеют практически одинаковый адрес. В OC Linux: глобальная, локальная, динамическая.

На ОС Windows и строки, и элементы в строках статического массива располагаются последовательно. В динамическом строки располагаются в различных ячейках памяти, а элементы в строках располагаются последовательно. На OC Linux элементы массивов располагаются так же, как и на OC Windows, но ближе к нулю находятся динамические массивы.

На OC Windows элементы статического одномерного, двумерного массива и элементы динамического одномерного, двумерного массива хранятся последовательно.

На OC Linux элементы статического одномерного, двумерного массива и элементы динамического одномерного массива хранятся последовательно, в отличие от элементов динамического двумерного массива, которые располагаются с определённым интервалом.

Также можно заметить, что без выравнивания её размер будет равен сумме размеров каждой отдельной переменной, входящей в это структуру. Таким образом структуру можно заметно уменьшить по памяти.

**5 ЛИСТИНГ**

#include <iostream>

using namespace std;

/\*Объявление глобальных переменных\*/

int globalOne = 0;

int globalTwo = 0;

int globalThree = 0;

int main(){

/\*Объявление автоматических переменных\*/

int automaticOne = 0;

int automaticTwo = 0;

int automaticThree = 0;

/\*Объявление динамических переменных\*/

int \*dynamicOne = new int;

int \*dynamicTwo = new int;

int \*dynamicThree = new int;

/\*Вывод\*/

cout << "Global Variables:\n";

cout << "&GlobalOne = " << &globalOne << ";\n";

cout << "&GlobalTwo = " << &globalTwo << ";\n";

cout << "&GlobalThree = " << &globalThree << ";\n\n";

cout << "Automatic Variables:\n";

cout << "&AutomaticOne = " << &automaticOne << ";\n";

cout << "&AutomaticTwo = " << &automaticTwo << ";\n";

cout << "&AutomaticThree = " << &automaticThree << ";\n\n";

cout << "Dynamic Variables:\n";

cout << "&DynamicOne = " << dynamicOne << ";\n";

cout << "&DynamicTwo = " << dynamicTwo << ";\n";

cout << "&DynamicThree = " << dynamicThree << ";\n";

/\*Удаление динамических переменных\*/

delete dynamicOne;

delete dynamicTwo;

delete dynamicThree;

return 0;

}

#include <iostream>

#define n 2

using namespace std;

int main(){

/\*Объявляем одномерный статический массив\*/

int staticArray1[n];

/\*Объявляем двумерный статический массив\*/

int staticArray2[n][n];

/\*Объявляем одномерный динамический массив\*/

int \*dynamicArray1 = new int[n];

/\*Объявляем двумерный динамический массив\*/

int \*\*dynamicArray2 = new int\* [n];

for (int i = 0; i < n; i++)

dynamicArray2[i] = new int[n];

/\*Вывод\*/

cout << "Static Arrary []:\n";

for(int i = 0; i < n; i ++)

cout << "staticArray1[" << i << "] = " << &staticArray1[i] << "\n";

cout << "\nStatic Arrary [][]:\n";

for(int i = 0; i < n; i++)

for(int j = 0; j < n; j++)

cout << "staticArray2[" << i << "]" << "[" << j << "] = " << &staticArray2[i][j] << "\n";

cout << "\nDynamic Arrary []:\n";

for(int i = 0; i < n; i ++)

cout << "dynamicArray1[" << i << "] = " << &dynamicArray1[i] << "\n";

cout << "\nDynamic Arrary [][]:\n";

for(int i = 0; i < n; i++)

for(int j = 0; j < n; j++)

cout << "dynamicArray2[" << i << "]" << "[" << j << "] = " << &dynamicArray2[i][j] << "\n";

/\*Удаление динамических массивов\*/

for(int i = 0; i < n; i++)

delete [] dynamicArray2[i];

delete dynamicArray2;

delete[] dynamicArray1;

return 0;

}

#include <iostream>

using namespace std;

/\*Объявление структур\*/

/\*Не упорядоченная, с выравниванием\*/

struct StructFirst{

char Char;

int Int;

short Short;

double Double;

};

/\*Упорядоченная, с выравниванием\*/

struct StructSecond{

char Char;

short Short;

int Int;

double Double;

};

/\*Не упорядоченная, без выравнивания\*/

#pragma pack(push,1)

struct StructThird{

double Double;

char Char;

int Int;

short Short;

};

#pragma pack(pop)

/\*Упорядоченная, без выравнивания\*/

#pragma pack(push,1)

struct StructFourth{

char Char;

short Short;

int Int;

double Double;

};

#pragma pack(pop)

int main(){

/\*Вывод\*/

StructFirst structFirst;

cout << "Struct 1:\n";

cout << "Size - char: " << sizeof(structFirst.Char) << " - " << &structFirst.Char<< "\n";

cout << "Size - short: " << sizeof(structFirst.Short) << " - " << &structFirst.Short << "\n";

cout << "Size - int: " << sizeof(structFirst.Int) << " - " << &structFirst.Int << "\n";

cout << "Size - double: " << sizeof(structFirst.Double) << " - " << &structFirst.Double << "\n";

cout << "Real size of StructFirst: " << sizeof(structFirst) << "\n\n";

StructSecond structSecond;

cout << "Struct 2:\n";

cout << "Size - char: " << sizeof(structSecond.Char) << " - " << &structSecond.Char << "\n";

cout << "Size - short: " << sizeof(structSecond.Short) << " - " << &structSecond.Short << "\n";

cout << "Size - int: " << sizeof(structSecond.Int) << " - " << &structSecond.Int << "\n";

cout << "Size - double: " << sizeof(structSecond.Double) << " - " << &structSecond.Double << "\n";

cout << "Real size of StructSecond: " << sizeof(structSecond) << "\n\n";

StructThird structThird;

cout << "Struct 3:\n";

cout << "Size - char: " << sizeof(structThird.Char) << " - " << &structThird.Char << "\n";

cout << "Size - short: " << sizeof(structThird.Short) << " - " << &structThird.Short << "\n";

cout << "Size - int: " << sizeof(structThird.Int) << " - " << &structThird.Int << "\n";

cout << "Size - double: " << sizeof(structThird.Double) << " - " << &structThird.Double << "\n";

cout << "Real size of StructThird: " << sizeof(structThird) << "\n\n";

StructFourth structFourth;

cout << "Struct 4:\n";

cout << "Size - char: " << sizeof(structFourth.Char) << " - " << &structFourth.Char << "\n";

cout << "Size - short: " << sizeof(structFourth.Short) << " - " << &structFourth.Short << "\n";

cout << "Size - int: " << sizeof(structFourth.Int) << " - " << &structFourth.Int << "\n";

cout << "Size - double: " << sizeof(structFourth.Double) << " - " << &structFourth.Double << "\n";

cout << "Real size of StructFourth: " << sizeof(structFourth) << "\n\n";

return 0;

}