|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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

ОПРЕДЕЛЕНИЕ ОСОБЕННОСТЕЙ РАСПОЛОЖЕНИЯ ОБЪЕКТОВ ПРОГРАММЫ В ПАМЯТИ

|  |  |
| --- | --- |
| Обучающийся | Проскуряков И.С. |
| Шифр | 19К0414 |
| Направление подготовки | 01.03.02  Прикладная математика и информатика |
| Группа | КМБО-02-19 |

|  |
| --- |
|  |

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

Москва 2019

**Содержание:**

[Тема, цель работы, задание 1](#_Toc26915680)

[Тема: особенности расположения объектов программы в памяти. 1](#_Toc26915681)

[Цель: определение закономерностей размещения объектов программы в оперативной памяти. 1](#_Toc26915682)

[Задание: 1](#_Toc26915683)

[Уточнение задания: 1](#_Toc26915684)

[Ход работы 2](#_Toc26915685)

[Задача 1 2](#_Toc26915686)

[Задача 2 3](#_Toc26915687)

[Задача 3 4](#_Toc26915688)

[Вывод: 5](#_Toc26915689)

[Листинг: 6](#_Toc26915690)

Тема, цель работы, задание

Тема: особенности расположения объектов программы в памяти.

Цель: определение закономерностей размещения объектов программы в оперативной памяти.

Задание:

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).

Уточнение задания:

Структуры должны описывать объекты какой-либо предметной области, все поля структур не должны быть одного типа.

Программы, требуемые для решения задач, должны без ошибок работать на ОС Linux (компиляторы g++/clang) и ОС Windows (компиляторы minGW/msvc). Каждая задача решается и для ОС Linux, и для ОС Windows.

Ход работы

Для решения поставленных задач была написана программа на языке программирования C++ стандарта C++14. Далее представлено описание этого решения.

Задача 1

**Гипотеза:** переменные в памяти располагаются «по порядку»: наименьший абсолютный адрес имеют глобальные переменные, затем идут динамические переменные и переменные стека.

Для решения первой задачи были созданы функции: variables, compare\_var, print\_arr\_of\_var.

Функция variable (тип void) вызывается при запуске программы из функции main и является «телом» для решения первой задачи. В ней создаются переменные local\_auto\_val и dynamic\_val и инициализируются начальными значениями 34 и 35 соответственно для последующей их однозначной идентификации.

|  |  |  |
| --- | --- | --- |
| Тип | Имя | Значение |
| int | local\_auto\_val | 34 |
| int\* | dynamic\_val | 35 |

Затем, при вызове функции print\_arr\_of\_var, вызывается функция compare\_var, в которую в качестве параметров передаются указатели на переменную global\_var(глобальная переменная типа int, инициализирована значением 33), local\_auto\_val и dynamic\_val.

В функции compare\_var(int\*\*) создаётся массив указателей arr, который инициализируется принятыми в качестве параметров функции указателями. Функция сортирует массив arr по возрастанию указателей, в нём находящихся (используется bubble sort), затем возвращает отсортированный массив.

Функция print\_arr\_of\_var(void) принимает массив указателей arr и его размер s. Внутри функции цикл for проходит по «первому уровню» массива и за счёт значений, находящихся по указателям, определяет, какой переменной принадлежит указатель, записанный в массив: global val, local\_auto\_val, dynamic\_val. Т.к. указатели уже отсортированы по возрастанию, вывод функции даёт возможность определить порядок, в котором следуют ячейки памяти, выделенные ОС для конкретных переменных в ходе работы программы.

При запуске данной программы на OC Linux Mint 19.2(Unix OC) было обнаружено, что наименьший абсолютный адрес всегда имеет глобальная переменная, затем по возрастанию располагается динамическая переменная, и наибольший абсолютный адрес имеет автоматическая переменная.

При запуске программы на OC Windows(7, 10) было обнаружено отсутствие каких-либо закономерностей в размещении переменных в памяти.

Задача 2

**Гипотеза:** элементы массивов располагаются в памяти «друг за другом» - последовательно.

Для решения этой задачи были созданы функции arrays(void), print\_\_(void).

Функция arrays является «телом» для решения второй задачи. Здесь создаются и инициализируются одномерный и двумерный статические массивы и одномерный и двумерный динамические массивы, как показано в таблице (для инициализации была использована глобальная переменная size\_arr(тип const short), проинициализированная значением 2):

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Размер | Имя | Значение |
| int[] | 2 | static\_one\_dim\_arr | {1,2} |
| int[][] | 2 x 2 | static\_two\_dim\_arr | {5,6}, {7,8} |
| int\* | 2 | dynamic\_one\_dim\_arr | {0, 1} |
| int\*\* | 2 x 2 | dynamic\_two\_dim\_arr | {8, 9}, {9, 10} |

Далее в функции выводятся указатели на элементы каждого из массивов. Запустив данную программу на OC Windows и Unix OC(версии те же, что и в задаче 1), обнаруживалась следующая закономерность: в статических массивах и одномерном динамическом элементы следуют в памяти друг за другом, занимая без «пробелов» определённый пласт в памяти. Двумерный динамический массив отличается следующим: первый и второй вложенные массивы находятся в разных частях памяти, из-за чего последний элемент первого вложенного массива и первый элемент второго вложенного массива идут не «друг за другом», а с некоторым интервалом.

Функция print\_\_ используется исключительно для создания более читаемого вывода в консоль и не несёт прикладной нагрузки. Далее о ней упоминаться не будет.

Задача 3

**Гипотеза:** память, занимаемая объектом структуры с автоматическим выравниваем, не будет совпадать с суммой размеров полей структуры по отдельности. Объект структуры, выравненной вручную, будет совпадать с суммой размеров полей структуры. С помощью изменения порядка следования полей удастся уменьшить количество памяти, занимаемое структурой с автоматическим выравниванием полей.

Для решения этой задачи были созданы функция structs(void), структуры City\_district1 и City\_district2. Состав структур полностью идентичен вплоть до порядка следования их полей. Отличие: на структуре City\_district2 было использовано ручное выравнивание с помощью команды предпроцессора #pragma pack; в City\_district1 использовано выравнивание полей по умолчанию (делает компилятор). Структура City\_struct представлена в таблице:

|  |  |  |  |
| --- | --- | --- | --- |
| Тип поля | Размер  (предположительный) | Название | Назначение |
| char[] | 60 байт | name | Название района города |
| int | 4 байта | count\_inhabitans | Количество жителей |
| char[] | 21 байт | local\_boss | Имя главы местной администрации |
| float | 4 байта | middle\_age | Средний возраст жителей |

Функция structs является «телом» решения данной задачи. Здесь создаются объекты South\_East и South\_West – структур City\_district1 и 2 соответственно. Здесь же в переменной res(тип int) подсчитывается количество памяти, необходимое для хранения полей полей структур по отдельности. Далее выводятся фактические размеры объектов South\_East и South\_West, значение переменной res. Отсюда мы видим, что South\_East занимает на 3 байта больше, чем структура без выравнивания и сумма по отдельности хранящихся её полей. Выводя указатели на поля структур, мы видим, что поля выравненной автоматически структуры(объект South\_East) идут не совсем последовательно: после массива char[21](boss\_name) до ячейки со следующим полем пропущено 3 байта – это объясняется автоматическим выравниванием компилятора.

Изменение порядка следования полей в структурах не привело к изменению их размера – при автоматическом выравнивании размер структуры будет кратен самому «большому» по памяти типу из тех, которыми проинициализированы её поля.

Вывод:

Гипотезы, выдвинутые в начале каждой из задач, либо подтвердились не в полном объёме, либо не подтвердились: в задаче 1 гипотеза верна только при работе с Unix OC; в задаче 2 обнаружилось, что элементы многомерных динамических массивов располагаются не совсем последовательно: вложенные массивы могут находиться в разных частях памяти. Наконец, в задаче 3 не удалось путём изменения порядка следования полей в автоматически выравниваемой структуре уменьшить её размер.

Всё вышеизложенное говорит о специфике каждого конкретного случая и необходимости при работе с памятью учитывать особенности каждого конкретного проекта и среды (операционной системы), для которой он предназначен.

Листинг:

#include <iostream>

using namespace std;

int global\_val = 33;

const short size\_arr = 2;

struct City\_district1{

char name[60];

int count\_inhabitants;

char local\_boss[21];

float middle\_age;

};

#pragma pack(push, 1)

struct City\_district2{

char name[60];

int count\_inhabitants;

char local\_boss[21];

float middle\_age;

};

#pragma pack(pop)

void print\_\_(){

for(int i = 0; i < 12; i++){

cout << "\_\_";

}

cout << '\n';

}

void print\_arr\_of\_var(int \*\*arr, short s){

for(int i = 0; i < s; ++i){

switch (arr[i][0]){

case (33):

cout << "global variable (";

cout << arr[i] << ")";

break;

case (34):

cout << "local auto variable (";

cout << arr[i] << ")";

break;

case (35):

cout << "dynamic variable (";

cout << arr[i] << ")";

break;

default:

break;

}

i == s - 1 ? cout << "" : cout << ", ";

}

cout << '\n';

print\_\_();

}

int\*\* compare\_var(int \*a, int \*b, int \*c){

const short size = 3;

int \*\*arr = new int\*[size];

arr[0] = a;

arr[1] = b;

arr[2] = c;

for(int i = 0; i < size; ++i){

for(int j = 0; j < size - i - 1; ++j){

if(arr[j] - arr[j + 1] > 0){

swap(arr[j], arr[j + 1]);

}

}

}

return arr;

}

void variables(){

int local\_auto\_val = 34;

int \*dynamic\_val = new int;

\*dynamic\_val = 35;

print\_arr\_of\_var(compare\_var(&global\_val, &local\_auto\_val, dynamic\_val), 3);

}

void arrays(){

int static\_one\_dim\_arr[size\_arr]{1,2};

int \*dynamic\_one\_dim\_arr = new int[size\_arr];

for(int i = 0; i < size\_arr; ++i){

dynamic\_one\_dim\_arr[i] = i;

}

int static\_two\_dim\_arr[size\_arr][size\_arr]{{5,6},{7,8}};

int \*\*dynamic\_two\_dim\_arr = new int\*[size\_arr];

for(int i = 0; i < size\_arr; ++i){

dynamic\_two\_dim\_arr[i] = new int[size\_arr];

}

for(int i = 0; i < size\_arr; ++i){

for(int j = 0; j < size\_arr; ++j){

dynamic\_two\_dim\_arr[i][j] = i + j + 8;

}

}

cout << "static one-dimensional array (";

cout << &(static\_one\_dim\_arr[0]) << ")" << endl;

for(int j = 0; j < size\_arr; j++){

cout << &(static\_one\_dim\_arr[j]) << " ";

}

cout << '\n';

cout << "dynamic one-dimensional array (";

cout << (dynamic\_one\_dim\_arr) << ")" << endl;

for(int j = 0; j < size\_arr; j++){

cout << &(dynamic\_one\_dim\_arr[j]) << " ";

}

cout << '\n';

cout << "static two-dimensional array (";

cout << &(static\_two\_dim\_arr[0][0]) << ")" << endl;

for(int j = 0; j < size\_arr; j++){

for(int k = 0; k < size\_arr; k++){

cout << &(static\_two\_dim\_arr[j][k]) << " ";

}

}

cout << '\n';

cout << "dynamic two-dimensional array (";

cout << (dynamic\_two\_dim\_arr[0]) << ")" << endl;

for(int j = 0; j < size\_arr; j++){

for(int k = 0; k < size\_arr; k++){

cout << &(dynamic\_two\_dim\_arr[j][k]) << " ";

}

}

cout << '\n';

print\_\_();

}

void structs(){

int res = 60 + 21 + sizeof(int) + sizeof(float);

City\_district1 South\_East;

City\_district2 South\_West;

cout << "Sum of all elements in structs: " << res << endl;

cout << "Without pushing: " << sizeof(South\_East) << endl;

cout << "With pushing: " << sizeof(South\_West) << endl;

cout << "Struct without #pragma: " << endl;

cout << &(South\_East.name) << ", " << &(South\_East.count\_inhabitants)

<< ", " << &(South\_East.local\_boss) << ", "<<

&(South\_East.middle\_age) << endl;

cout << "Struct with #pragma: " << endl;

cout << &(South\_West.name) << ", " << &(South\_West.count\_inhabitants)

<< ", " << &(South\_West.local\_boss) << ", "<<

&(South\_West.middle\_age) << endl;

print\_\_();

}

int main() {

variables();

arrays();

structs();

return 0;

}