Балтийский государственный технический университет  
«ВОЕНМЕХ» им. Д. Ф. Устинова

Кафедра О7 «Информационные системы и программная инженерия»

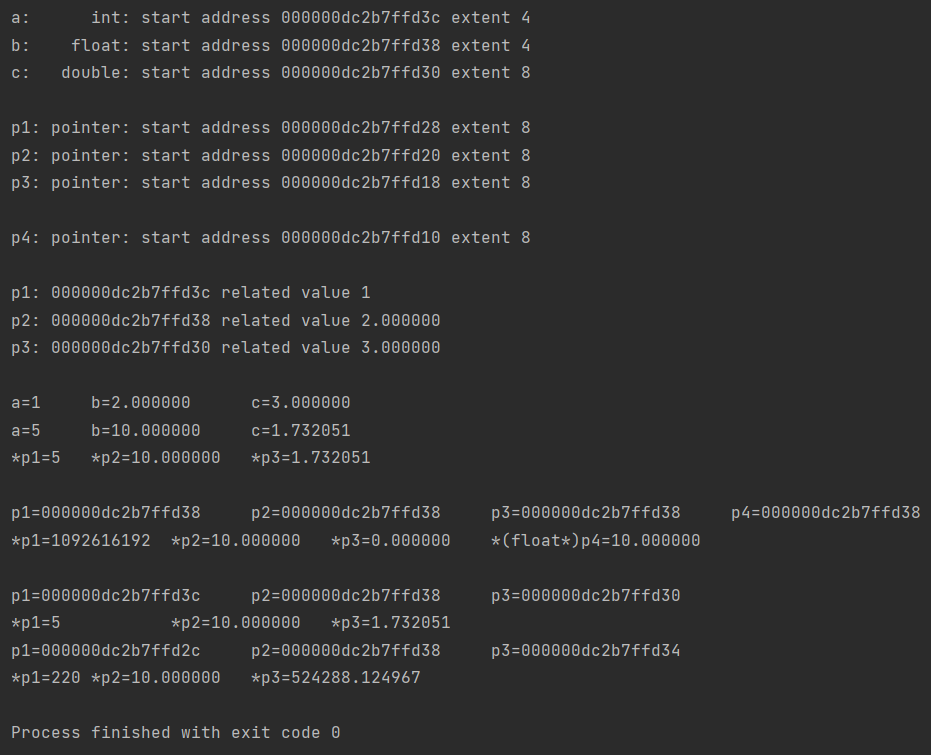
**Практическая работа №3**по дисциплине «Информатика: Основы программирования»  
на тему «Указатели»

Выполнил:  
Студент УсовД.А.  
Группа Е123Б  
  
Преподаватель:  
Лестенко Н.А.

Санкт-Петербург  
2022 г.

**Задание 1.**

Проанализировать текст представленной программы и выдаваемые программой результаты. Объяснить, почему результаты именно такие.

Результаты работы программы:

Текст программы:

#include<stdlib.h>

#include<stdio.h>

#include<math.h>

int main()

{

/\* «Обычные» переменные \*/

int a = 1;

float b = 2;

double c = 3;

/\* Указатели \*/

int \*p1 = &a;

float \*p2 = &b;

double \*p3 = &c;

void \*p4 = NULL;

/\* Адреса «обычных» переменных и размер выделяемой памяти \*/

printf("a: int: start address %p extent %d\n",&a,sizeof(a));

printf("b: float: start address %p extent %d\n",&b,sizeof(b));

printf("c: double: start address %p extent %d\n\n",&c,sizeof(c));

/\* Адреса указателей и размер выделяемой памяти \*/

printf("p1: pointer: start address %p extent %d\n",&p1,sizeof(p1));

printf("p2: pointer: start address %p extent %d\n",&p2,sizeof(p2));

printf("p3: pointer: start address %p extent %d\n\n",&p3,sizeof(p3));

printf("p4: pointer: start address %p extent %d\n\n",&p4,sizeof(p4));

/\* Значения, на которые ссылаются указатели \*/

printf("p1: %p related value %d\n",p1,\*p1);

printf("p2: %p related value %f\n",p2,\*p2);

printf("p3: %p related value %lf\n\n",p3,\*p3);

Размещение переменных в памяти:



Вывод: размещение элементов памяти зависит от операционной системы. В данном случае в ОС Windows 11 используется обратный порядок записи.

/\* Использование указателей в выражениях \*/

printf("a=%d\tb=%f\tc=%lf\n",a,b,c);

\*p1 = 5;

\*p2 = \*p2 \* \*p1;

\*p3 = sqrt(\*p3);

printf("a=%d\tb=%f\tc=%lf\n",a,b,c);

printf("\*p1=%d\t\*p2=%f\t\*p3=%lf\n\n",\*p1,\*p2,\*p3);

/\* Присваивание указателей \*/

p1 = (int\*)p2;

p3 = (double\*)p2;

p4 = p2;

printf("p1=%p\tp2=%p\tp3=%p\tp4=%p\n",p1,p2,p3,p4);

printf("\*p1=%d\t\*p2=%f\t\*p3=%f\t\*(float\*)p4=%f\n\n",\*p1,\*p2,\*p3,\*(float\*)p4);



Вывод: указателям можно присвоить адреса на переменные других типов, однако стоит понимать, что значение может представляться

по-другому.

Именно поэтому при получении значения указателя (\*p1) мы получили не 10, а 109261619210 = 4120000016, так как представление чисел типа

double отличается от типа int. Разыменование позволяет выполнять операции над значениями, находящимися по адресам указателей.

/\* Изменение значений указателей \*/

p1++;

p3--;

printf("p1=%p\tp2=%p\tp3=%p\n",p1,p2,p3);

printf("\*p1=%d\t\t\*p2=%f\t\*p3=%lf\n",\*p1,\*p2,\*p3);



Вывод: значения вернулись к прежним, так как мы сместили указатели на изначальные адреса (указатель p1 сместился на 4 байт, так

как это указатель на int, p3 — на 8 байт, так как это указатель на double).

p1 -= 4;

p3 = (double\*)&a - 1;

printf("p1=%p\tp2=%p\tp3=%p\n",p1,p2,p3);

printf("\*p1=%d\t\*p2=%f\t\*p3=%lf\n",\*p1,\*p2,\*p3);

return 0;

}



Вывод: при смещении указателя p1 влево на 16 байт (4\*sizeof(int)) значение под ним является частью значения p1. Это значение при выводе будет выглядеть так: 3276410. p3 является указателем на тип double, поэтому мы после присваивания к адресу a сместились на 8 байт влево (1\*sizeof(double)).

Выводы: Порядок записи переменных в памяти зависит от ОС. От типа указателя зависит смещение, указателю можно присвоить указатель другого типа или вывести значение указателя на void с помощью явного приведения

**Задание 2.**

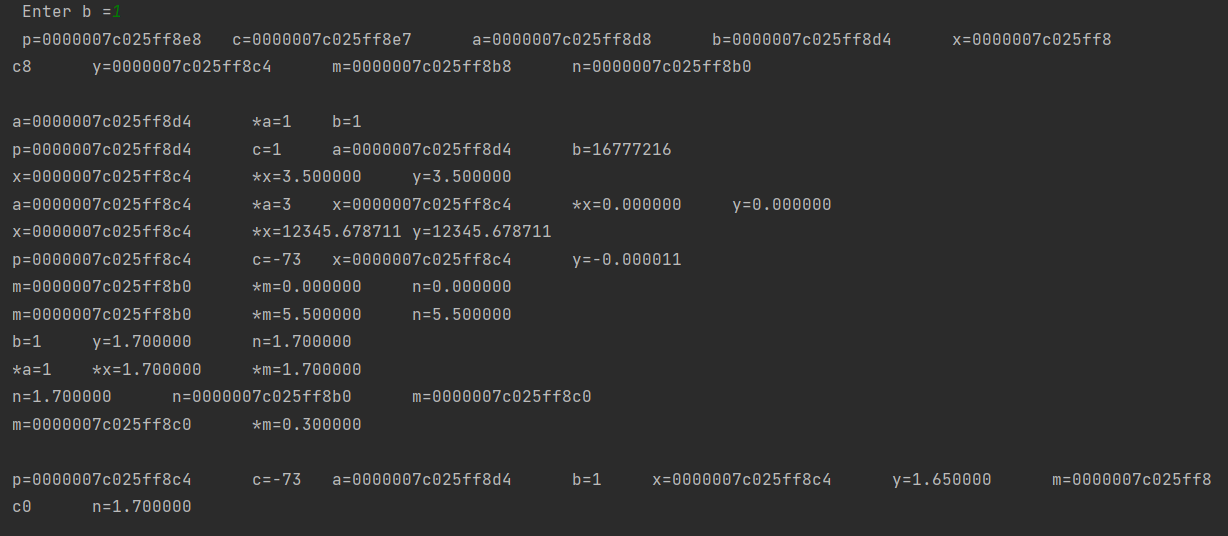
Операционная система Windows 11 x64, среда разработки Clion

Проанализировать текст представленной программы, найти в нем синтаксические ошибки и исправить их, в начало программы добавить вывод на экран адресов всех переменных и объемов занимаемой ими памяти, а в конец – значений всех переменных, проанализировать полученные результаты и объяснить, почему они именно такие. Заменить инструкцию «m+=2;» инструкцией «m++;», проанализировать результат.

Текст измененной программы:

#include <stdio.h>  
  
int main() {  
 char \*p, c;  
 int \*a, b;  
 float \*x, y = 3.5f;  
 double \*m, n;  
 a = &b;  
 printf(" Enter b = ");  
 scanf("%d", a);  
 printf("p=%p\tc=%p\ta=%p\tb=%p\tx=%p\ty=%p\tm=%p\tn=%p\n\n", &p, &c, &a, &b, &x, &y, &m, &n);  
 printf("a=%p\t\*a=%d\tb=%d\n", a, \*a, b);  
 p = (char \*) a;  
 c = \*p;  
 \*p = \*(p + 3);  
 \*(p + 3) = c;  
 printf("p=%p\tc=%d\ta=%p\tb=%d\n", p, c, a, b);  
 x = &y;  
 printf("x=%p\t\*x=%f\ty=%f\n", x, \*x, y);  
 a = (int \*) x;  
 \*a = (int) \*x;  
 printf("a=%p\t\*a=%d\tx=%p\t\*x=%f\ty=%f\n", a, \*a, x, \*x, y);  
 a = &b;  
 y = 12345.6789f;  
 printf("x=%p\t\*x=%f\ty=%f\n", x, \*x, y);  
 p = (char \*) x;  
 c = \*p;  
 \*p = \*(p + 3);  
 \*(p + 3) = c;  
 printf("p=%p\tc=%d\tx=%p\ty=%f\n", p, c, x, y);  
 m = &n;  
 printf("m=%p\t\*m=%lf\tn=%lf\n", m, \*m, n);  
 n = 5.5;  
 printf("m=%p\t\*m=%lf\tn=%lf\n", m, \*m, n);  
 b = n = y = 1.7;  
 printf("b=%d\ty=%f\tn=%lf\n", b, y, n);  
 printf("\*a=%d\t\*x=%f\t\*m=%lf\n", \*a, \*x, \*m);  
 m++;  
 printf("n=%lf\tn=%p\tm=%p\n", n, &n, m);  
 \*m = (float) \*a - n + (int) \*x;  
 printf("m=%p\t\*m=%lf\n\n", m, \*m);  
 printf("p=%p\tc=%d\ta=%p\tb=%d\tx=%p\ty=%f\tm=%p\tn=%f\n\n", p, c, a, b, x, y, m, n);  
 return 0;  
}

Результат работы программы (первый вариант «m+=2;»):

**Размещение переменных в памяти:





p = a;

c = \*p;

\*p = \*(p+3);

\*(p+3) = c;

printf("p=%p\tc=%d\ta=%p\tb=%d\n", p, c, a, b);

Здесь указателю p присваивается значение указателя a, но так как указатель char на 1 байт, то p будет указывать только на этот байт. При этом мы меняем местами значения по адресам f8d4 и f8d7 с помощью переменной c, это удалось сделать, так смещение у p равно размеру char (1 байт), в переменной b получилось число 100000016 = 1677721610.

**

**

x = &y;

printf("x=%p\t\*x=%f\ty=%f\n", x, \*x, y);

a = x;

\*a = \*x;

pintf("a=%p\t\*a=%d\tx=%p\t\*x=%f\ty=%f\n", a, \*a, x, \*x, y);

Здесь указателю x даётся адрес переменной y, а к указателю a присваивается значение указателя x, преобразованное в int. В результате изменения значения переменной y (\*a = \*x) в ней оказалось число 3, что в стандарте ieee754 будет очень маленьким числом, поэтому при выводе \*x (значения под указателем на float) получается 0. Напротив, при выводе \*a (значения под указателем на int) получается число, обрезанное до целого (3.5 (int) = 3), так получилось при присваивании \*a = \*x.

**

**

a = &b;

y = 12345.6789f;

printf("x=%p\t\*x=%f\ty=%f\n", x, \*x, y);

p = (char \*) x;

c = \*p;

\*p = \*(p + 3);

\*(p + 3) = c;

printf("p=%p\tc=%d\tx=%p\ty=%f\n", p, c, x, y);

Здесь присваивается указателю p значение указателя x с преобразованием в char, далее идёт обмен значениями по адресам p и p+3 (f8c4 и f8c7 соответственно), в результате при выводе переменной y получается очень маленькое число. т.к. смещенный порядок числа стал меньше (110 < 140).**

**

m = &n;

printf("m=%p\t\*m=%lf\tn=%lf\n", m, \*m, n);

n = 5.5;

printf("m=%p\t\*m=%lf\tn=%lf\n", m, \*m, n);

b = n = y = 1.7;

printf("b=%d\ty=%f\tn=%lf\n", b, y, n);

printf("\*a=%d\t\*x=%f\t\*m=%lf\n", \*a, \*x, \*m);

m++;

printf("n=%lf\tn=%p\tm=%p\n", n, &n, m);

\*m = (float) \*a - n + (int) \*x;

printf("m=%p\t\*m=%lf\n\n", m, \*m);

printf("p=%p\tc=%d\ta=%p\tb=%d\tx=%p\ty=%f\tm=%p\tn=%f\n\n", p, c, a, b, x, y, m, n);

return 0;

При m+=2 указатель был смещён на 16 байт (2\*sizeof (double)), первые 4 ячейки не являются какой-либо переменной, а вторые 4 ячейки — это переменная y. Потом вычисляется значение:

\*m = (float) \*a – n + (int) \*x:

\*m = 1. - 1.7 + 1 = 0.3

При выводе y отображается 1.650000, так как на порядок у float отводится меньше, чем у double, в итоге сам порядок в представлении float стал больше, и число изменилось (0.300000 => 1.650000).

**

**

После замены инструкции «m+=2;» инструкцией «m++;» указатель стал ссылаться на себя. При попытке изменить значение под указателем выводится ошибка, так как это значение (адрес) будет находиться в другой, недоступной для компилятора, области памяти. (segmentation fault – ошибка доступа к памяти).

**

**

**Задание 3.**

Объявить по две переменные типов *char*, *int* и *double*, а также указатель на *char.* Вывести на экран размеры и адреса всех переменных, начертить схему расположения переменных в памяти. Поменять порядок объявления переменных, запустить программу повторно, проанализировать, что изменилось. Задать переменной типа *int* такое значение, чтобы значение каждого байта было уникальным, использовать для этого шестнадцатеричную константу. Записать адрес этой переменной в указатель на *char* и с его помощью вывести на экран содержимое каждого байта (тоже в шестнадцатеричной системе счисления). Проанализировать, прямой или обратный порядок расположения байт при записи числа применяется в используемой системе.

Повторить выполнение этого задания на другой платформе. Сравнить результаты работы программы на разных платформах, сделать выводы.

Операционная система Windows 11 x64, среда разработки Clion

Текст программы (первый порядок объявления переменных):

#include <stdio.h>

int main(){

char c1,c2;

int i1,i2;

double d1,d2;

char\* pC;

printf("c1, address = %p, size = %llu\n",&c1,sizeof(c1));

printf("c2, address = %p, size = %llu\n",&c2,sizeof(c2));

printf("i1, address = %p, size = %llu\n",&i1,sizeof(i1));

printf("i2, address = %p, size = %llu\n",&i2,sizeof(i2));

printf("d1, address = %p, size = %llu\n",&d1,sizeof(d1));

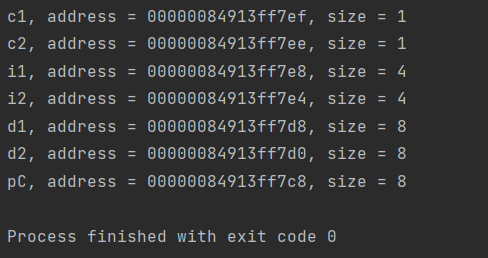
printf("d2, address = %p, size = %llu\n",&d2,sizeof(d2));

printf("pC, address = %p, size = %llu\n",&pC,sizeof(pC));

return 0;

}

Результаты работы программы:

**

Размещение переменных в памяти



Текст программы (второй порядок объявления переменных):

#include <stdio.h>

int main() {

char c1;

double d1;

char \*pC;

int i1;

double d2;

char c2;

int i2 = 0x7f5f3a22;

printf("c1, address = %p, size = %d\n", &c1, sizeof(c1));

printf("c2, address = %p, size = %d\n", &c2, sizeof(c2));

printf("i1, address = %p, size = %d\n", &i1, sizeof(i1));

printf("i2, address = %p, size = %d\n", &i2, sizeof(i2));

printf("d1, address = %p, size = %d\n", &d1, sizeof(d1));

printf("d2, address = %p, size = %d\n", &d2, sizeof(d2));

printf("pC, address = %p, size = %d\n\n", &pC, sizeof(pC));

printf("i2 value = %x\n", i2);

printf("i2 byte structure:\n\n");

for (pC = (char \*) &i2; pC < (char \*) &i2 + sizeof(int); pC++) {

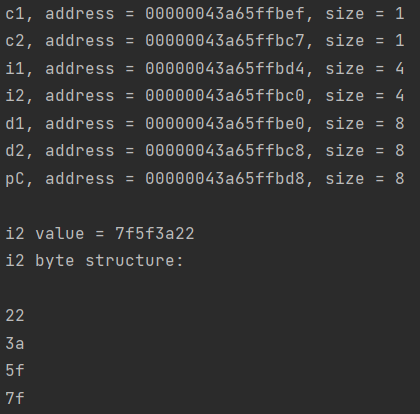
printf("%x\n", \*pC);

}

return 0;

}

Результаты работы программы:

**

Размещение переменных в памяти:



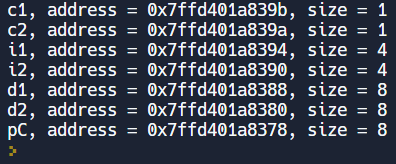


*Выводы:* память под переменные выделяется в том порядке, в котором они были объявлены, выделение памяти идёт от большего адреса к меньшему.

Порядок следования байт при записи числаобратный.

1. Онлайн-IDE Replit https://replit.com/languages/c

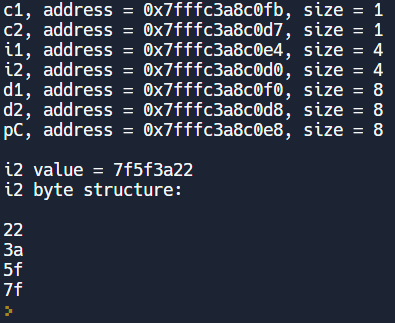
Результаты работы программы при первом порядке объявления переменных:

**

Размещение переменных в памяти:



Результаты работы программы при втором порядке объявления переменных:

**

Размещение переменных в памяти:

**



*Выводы:* память под переменные выделяется в том порядке, в котором они были объявлены, выделение памяти идёт от большего адреса к меньшему.

Порядок следования байт при записи числа обратный.

*Выводы:* Распределение памяти под переменные зависит от компилятора (gcc или clang) и ОС (Windows или Linux), но в обоих

системах порядок следования байт при записи числа является обратным.