Доколин Георгий ИУ7-22Б

Отчет по Проектно-технологической практике (тестированию, отладке и профилированию ПО)
Задание №3.4 в рамках вычислительного практикума.
Представление в памяти структур и объединений.

Оглавление

Введение

Цель работы: изучение представления в памяти структур и объединений при помощи отладчика **gdb**.

Локальные переменные

```
#include <stdio.h>

int main(void)
{
   int num_1 = 10;
   int num_2 = -13;
   char ch_1 = 78;3
   char ch_2 = -92;
   long long int lldnum_1 = 6789;
   long long int lldnum_2 = -5464;
   unsigned int unum_1 = 567823;
   unsigned int unum_2 = -427956;
   return 0;
}
```

Подсчитаем суммарный размер всех переменных, а потом проверим сумму при помощи команды sizeof. Все переменные занимают 34 байта, значит для дампа можем воспользоваться командой x /34xb &num_1 (&num_1 нужно для того, чтобы показать отладчкику с какого места нужно проводить дамп, а переменная num_1 храниться первой).

Все переменные:

```
(gdb) x /34xb &num_1
0x7fffffffdc90: 0x0a    0x00    0x00    0x00    0xf3
0xff    0xff    0xff
0x7fffffffdc98: 0x0f    0xaa    0x08    0x00    0x4c
0x78    0xf9    0xff
0x7fffffffdca0: 0x85    0x1a    0x00    0x00    0x00
0x00    0x00    0x00
0x7fffffffdca8: 0xa8    0xea    0xff    0xff    0xff
0xff    0xff    0xff
0x7fffffffdcb0: 0x01    0x00
```

Пояснение дампа:

```
(gdb) x /34xb &num_1
0x7fffffffdc90: num_1 = [0x0a    0x00    0x00    0x00]
num_2 = [0xf3    0xff    0xff    0xff]
0x7fffffffdc98: ch_1 = [0x0f]    ch_2 = [0xaa]
lldnum_1 = [0x08    0x00    0x4c    0x78    0xf9    0xff
0x7fffffffdca0: 0x85    0x1a]    lldnum_2 = [0x00
0x00    0x00    0x00    0x00    0x00
0x7fffffffdca8: 0xa8    0xea]    unum_1 = [0xff    0xff
0xff    0xff]    unum_2 = [0xff    0xff
0x7fffffffdcb0: 0x01    0x00]
```

Сведения о переменных:

Имя	Размер	Адрес
num_1	4	0x7fffffffdc90
num_2	4	0x7fffffffdc94
ch_1	1	0x7fffffffdc98
ch_2	1	0x7fffffffdc99
lldnum_1	8	0x7fffffffdc9a
lldnum_2	8	0x7fffffffdca2
unum_1	4	0x7fffffffdcaa
unum_2	4	0x7fffffffdcae

Выводы:

1. Переменные располагаются в порядке их объявления.
2. Переменные размещаются по адресам, номера которых кратны размеру его же типа.

Структуры

```
#include <stdio.h>

struct data
{
   double double_num_1;
   char ch_1;
   int num_1;
   long long int lldnum_1;
   unsigned long long ullnum_1;
};

int main(void)
{

   struct data data_1 = {586, 32, -729, 67892442, 4.72};
   return 0;
}
```

Для определения размера структуры воспользуемся командой sizeof(). Вся структура занимает 32 байта. Значит, что для дампа всей структуры нужно воспользоваться командой x /32xb &data_1.

Дамп всей структуры:

```
0x7fffffffdc90: 0x00    0x00    0x00    0x00    0x00
0x50    0x82    0x40
0x7fffffffdc98: 0x20    0x00    0x00    0x00    0x27
0xfd    0xff    0xff
0x7fffffffdca0: 0xda    0xf4    0x0b    0x04    0x00
0x00    0x00    0x00
0x7fffffffdca8: 0x04    0x00    0x00    0x00    0x00
0x00    0x00    0x00
```

Пояснение дампа:

1 – 8-ой байт – поле вещественного числа double_num_1.

9-ый байт – поле символа ch_1.

10 – 12-ый байт – выравнивающие байты для записи дальнейших переменных.

13 – 16-ый байт – поле целого числа num_1.

17 – 24-ый байт – поле целого числа, типа long long, lldnum_1.

25 – 32-ой байт - поле беззнакового целого числа ullnum_1.

Имя	Размер	Адрес
data_1. double_num_1	8	0x7fffffffdc90
data_1. ch_1	1	0x7fffffffdc98
data_1.num_1	4	0x7fffffffdc9c
data_1.lldnum_1	8	0x7fffffffdca0
data_1. ullnum_1	8	0x7fffffffdca8

Ответ на пункт 3:

Адрес поля зависит от его размера, поскольку адрес переменной в компьютере должен быть кратет размеру самого поля.

Ответ на пункт 4:

Адрес структурной переменной: 0x7fffffffdc90.

На знаечние адреса влияет поле переменной double_num_1.

Ответ на пункт 5:

```
#include <stdio.h>

#pragma pack(push, 1)
struct data
{
   double double_num_1;
   char ch_1;
   int num_1;
   long long int lldnum_1;
   unsigned long long ullnum_1;
};
#pragma pack(pop)

int main(void)
{
   struct data data_1 = {586, 32, -729, 67892442, 4.72};
   return 0;
}
```

Для определения размера структуры воспользуемся командой sizeof(). Вся структура занимает 29 байт. Значит, что для дампа всей структуры нужно воспользоваться командой x /29xb &data_1.

Дамп упакованной структуры:

```
0x7fffffffdc90: 0x00    0x00    0x00    0x00    0x00
0x50    0x82    0x40
0x7fffffffdc98: 0x20    0x27    0xfd    0xff    0xff
0xda    0xf4    0x0b
0x7fffffffdca0: 0x04    0x00    0x00    0x00    0x00
0x04    0x00    0x00
0x7fffffffdca8: 0x00    0x00    0x00    0x00    0x00
```

Пояснение дампа упакованной структуры:

1 – 8-ой байт – поле вещественного числа double_num_1.

9-ый байт – поле символа ch_1.

10 – 13-ый байт – поле целого числа num_1.

14 – 21-ый байт – поле целого числа, типа long long, lldnum_1.

22 – 29-ой байт - поле беззнакового целого числа ullnum_1.

Имя	Размер	Адрес
data_1. double_num_1	8	0x7fffffffdc90
data_1. ch_1	1	0x7fffffffdc98
data_1.num_1	4	0x7fffffffdc99
data_1.lldnum_1	8	0x7fffffffdc9d
data_1. ullnum_1	8	0x7fffffffdca5

Ответ на пункт 6:

Наиболее компактная упаковка структуры:

```
#include <stdio.h>
struct data
{
   long long int lldnum_1;
   unsigned long long ullnum_1;
   double double_num_1;
   int num_1;
   char ch_1;
};

int main(void)
{
   struct data data_1 = {58656783, 32, -729.6, 678, 4};
   return 0;
}
```

Узнаем размер структуры при помощи команды sizeof(), структура занимает 32 байта, однако последние 2 байта занимает выравннивание, поэтому итоговый объем структуры состовляет 30 байта.

Дамп структуры:

```
(gdb) x /32xb &data_1
0x7fffffffdc90: 0x0f    0x08    0x7f    0x03    0x00
0x00    0x00    0x00
0x7fffffffdc98: 0x20    0x00    0x00    0x00    0x00
0x00    0x00    0x00
0x7fffffffdca0: 0xcd    0xcc    0xcc    0xcc    0xcc
0xcc    0x86    0xc0
0x7fffffffdca8: 0xa6    0x02    0x00    0x00    0x04
0x55    0x00    0x00
```

Доказательство: элемент с минимальным размером расположен в конце, поэтому там будет максимальное количество байтов выравнивания, а значит будет задействовано минимальное количество памяти.

Ответ на 7 пункт: Завершающее выравнивание есть в размере 2 байт, более их нет, так как все остальные поля занимают место полностью, без необходимости выравнивания.

Объединение

```
#include <stdio.h>
union data
{
   long long int lldnum_1;
   unsigned int ullnum_1;
   double double_num_1;
   int num_1;
   char ch_1;
};
int main(void)
{
   union data data_1;
   data_1.num_1 = 58683;
   return 0;
}
```

Определим размер объединения при помощи нахождения максимального размера элемента. В данном примере такими объектами являются поля типа double и long long int с размерами 8 байт, значит команда для дампа будет выглядеть так: x/8xb &data_1.

Дамп памяти объединения:

```
0x7fffffffdca8: 0x3b    0xe5    0x00    0x00    0x55
0x55    0x00    0x00
```

Поле num_1 располагается в первых четырех байтах.

Дамп памяти объединения, после присвоения полю double_num_1 значения 4.67.

```
0x7fffffffdca8: 0xae    0x47    0xe1    0x7a    0x14
0xae    0x12    0x40
```

Теперь все 8 байт отвечают за поле double_num_1, в котором было заменено значение.