Задание №6 в рамках вычислительного практикума. Выравнивание переменных. Представление в памяти структур

Студент: Краснов Леонид
Группа: ИУ7-21Б

Оглавление

Цель

Изучить как располагаются в памяти локальные переменные.
Изучить как в памяти представлены структуры.

Расположение локальных переменных в памяти

1. Описать несколько локальных переменных разных типов.

Код программы

```c
#include <stdio.h>


int main(void)
{
    char ch = 'a';
    int i = 7;
    float f = 1.32;
    double d = 3.143343;
    long long ll = 23432423;
    unsigned u = 3221;
    return 0;
}
```

2. На дампе памяти показать, как переменные располагаются в памяти

Нахождение размеров объявленных переменных

```
(gdb) b 12
Breakpoint 2 at 0xaaaaaaaaa0798: file main.c, line 12.
(gdb) run
Starting program: /home/x13eav1sx/Рабочий стол/practice6/a.out

Breakpoint 2, main () at main.c:12
12          return 0;
(gdb) p sizeof(ch)
$1 = 1
(gdb) p sizeof(i)
$2 = 4
(gdb) p sizeof(f)
$7 = 4
(gdb) p sizeof(d)
$3 = 8
(gdb) p sizeof(ll)
$5 = 8
(gdb) p sizeof(u)
$6 = 4
```

Дамп памяти

```
(gdb) x /1xb &ch
0xffffffffec2f: 0x61
(gdb) x /4xb &i
0xffffffffec28: 0x07    0x00    0x00    0x00
(gdb) x /4xb &f
0xffffffffec24: 0xc3    0xf5    0xa8    0x3f
(gdb) x /8xb &d
0xffffffffec18: 0x5d    0xe2    0xc8    0x03    0x91    0x25    0x09    0x40
(gdb) x /8xb &ll
```

```
0xfffffffffec10: 0xe7    0x8c    0x65    0x01    0x00    0x00    0x00    0x00
(gdb) x /4xb &u
0xfffffffffec0c: 0x95    0x0c    0x00    0x00
(gdb)
```

Расположение переменных

Судя по адресам переменных видно, что они располагаются друг за другом:

0xfffffffffec2f	char ch = 'a';	gdb) p sizeof(ch) $1 = 1
0xfffffffffec28	int i = 7;	(gdb) p sizeof(i) $2 = 4
0xfffffffffec24	float f = 1.32;	(gdb) p sizeof(f) $7 = 4
0xfffffffffec18	double d = 3.143343;	(gdb) p sizeof(d) $3 = 8
0xfffffffffec10	long long ll = 23432423;	(gdb) p sizeof(ll) $5 = 8
0xfffffffffec0c	unsigned u = 3221;	(gdb) p sizeof(u) $6 = 4

То есть:

```
0xfffffffffec0c + 4 = 0xfffffffffec10
0xfffffffffec10 + 8 = 0xfffffffffec18
0xfffffffffec18 + 8 = 0xFFFFFFFFEC20
Далее произошло выравнивание, так как пропущено 4 байта
0xFFFFFFFFEC20 + 4 = 0xfffffffffec24
0xfffffffffec24 + 4 = 0xfffffffffec28
0xfffffffffec28 + 4 = 0xFFFFFFFFEC2C
Далее опять произошло выравнивание, так как пропущено 3 байта
0xFFFFFFFFEC2C + 3 = 0xfffffffffec2f
```

3. Составить таблицу, в которой указывается имя переменной, ее размер, значение адреса, по которому располагается переменная.

Таблица расположения переменных

Имя переменной	Размер	Адрес
ch	1	0xfffffffffec2f
i	4	0xfffffffffec28
f	4	0xfffffffffec24
d	8	0xfffffffffec18
ll	8	0xfffffffffec10
u	4	0xfffffffffec0c

4. Проанализировать зависимость значения адреса переменной от её размера

Значение адреса переменной напрямую зависит от ее размера. Если выравнивание выключено, то адрес переменной отличается от адреса соседней переменной на свой размер, если же выравнивание включено, то значение адреса переменной отличается от соседней на свой размер + количество байт, которые ушли на выравнивание.

Представление структур в памяти

1. Описать структуру, содержащую несколько полей разного типа.

Код программы

```c
#include <stdio.h>


struct data
{
    char ch;
    int i;
    unsigned u;
    float f;
    double d;
    long long ll;
};

int main(void)
{
    struct data dat;
    dat.ch = 'a';
    dat.i = -7;
    dat.u = 12;
    dat.f = 2.43;
    dat.d = 3.1432;
    dat.ll = 2423423;
    return 0;
}
```

2. Показать дамп памяти, который содержит эту структуру. На дампе показать расположение каждого поля структуры.

Дамп памяти, в которой находится структура.

```
(gdb) b 23
Breakpoint 1 at 0x798: file main.c, line 23.
(gdb) run
Starting program: /home/x13eav1sx/Рабочий стол/practice6/a.out

Breakpoint 1, main () at main.c:23
23          return 0;
(gdb) p sizeof(dat)
$1 = 32
(gdb) x /32xb &dat
0xffffffffec10: 0x61    0xec    0xff    0xff    0xf9    0xff    0xff    0xff
0xffffffffec18: 0x0c    0x00    0x00    0x00    0x1f    0x85    0x1b    0x40
0xffffffffec20: 0x30    0x4c    0xa6    0x0a    0x46    0x25    0x09    0x40
0xffffffffec28: 0x7f    0xfa    0x24    0x00    0x00    0x00    0x00    0x00
```

Дамп памяти полей структуры.

```
(gdb) x /1xb &ch
No symbol "ch" in current context.
(gdb) x /1xb &dat.ch
0xffffffffec10: 0x61
(gdb) x /4xb &dat.i
0xffffffffec14: 0xf9    0xff    0xff    0xff
(gdb) x /4xb &dat.u
0xffffffffec18: 0x0c    0x00    0x00    0x00
(gdb) x /4xb &dat.f
0xffffffffec1c: 0x1f    0x85    0x1b    0x40
(gdb) x /8xb &dat.d
```

```
0xfffffffffec20: 0x30    0x4c    0xa6    0x0a    0x46    0x25    0x09    0x40
(gdb) x /8xb &dat.ll
0xfffffffffec28: 0x7f    0xfa    0x24    0x00    0x00    0x00    0x00    0x00
(gdb)
```

3. Составить таблицу, в которой содержится имя поля, его размер, значение адреса, по которому располагается поле. Проанализировать зависимость значения адреса поля от его размера.

Таблица расположения полей структуры в памяти

Имя поля	Размер	Адрес
ch	1	0xfffffffffec10
i	4	0xfffffffffec14
u	4	0xfffffffffec18
f	4	0xfffffffffec1c
d	8	0xfffffffffec20
ll	8	0xfffffffffec28

Зависимость расположения поля от его размера

Видно, что поля располагаются в памяти друг за другом, где каждый следующий адрес – предыдущий адрес + размер типа + (если есть) байты для выравнивания.

4. По какому адресу располагается переменная структурного типа? Какое поле структуры повлияло на значение этого адреса?

Адрес переменной структурного типа

```
(gdb) p sizeof(dat)
$1 = 32
(gdb) x /32xb &dat
0xfffffffffec10: 0x61    0xec    0xff    0xff    0xf9    0xff    0xff    0xff
0xfffffffffec18: 0x0c    0x00    0x00    0x00    0x1f    0x85    0x1b    0x40
0xfffffffffec20: 0x30    0x4c    0xa6    0x0a    0x46    0x25    0x09    0x40
0xfffffffffec28: 0x7f    0xfa    0x24    0x00    0x00    0x00    0x00    0x00
```

Здесь видно, что структура располагается по адресу 0xfffffffffec10 такой-же адрес имеет первое поле структуры:

ch	1	0xfffffffffec10

5. Упаковать структуру и выполнить предыдущие пункты для упакованной структуры.

Код программы, описывающий структуру и упаковывающий ее.

```c
#include <stdio.h>


#pragma pack(1)
struct data
{
    char ch;
    int i;
    unsigned u;
    float f;
```

```
    double d;
    long long ll;
};

int main(void)
{
    struct data dat;
    dat.ch = 'a';
    dat.i = -7;
    dat.u = 12;
    dat.f = 2.43;
    dat.d = 3.1432;
    dat.ll = 2423423;
    return 0;
}
```

Дамп памяти, который содержит эту структуру

```
(gdb) x /29xb &dat
0xfffffffffec10: 0x61    0xf9    0xff    0xff    0xff    0x0c    0x00    0x00
0xfffffffffec18: 0x00    0x1f    0x85    0x1b    0x40    0x30    0x4c    0xa6
0xfffffffffec20: 0x0a    0x46    0x25    0x09    0x40    0x7f    0xfa    0x24
0xfffffffffec28: 0x00    0x00    0x00    0x00    0x00
```

Расположение каждого поля структуры

```
(gdb) x /1xb &dat.ch
0xfffffffffec10: 0x61
(gdb) x /4xb &dat.i
0xfffffffffec11: 0xf9    0xff    0xff    0xff
(gdb) x /4xb &dat.u
0xfffffffffec15: 0x0c    0x00    0x00    0x00
(gdb) x /4xb &dat.f
0xfffffffffec19: 0x1f    0x85    0x1b    0x40
(gdb) x /8xb &dat.d
0xfffffffffec1d: 0x30    0x4c    0xa6    0x0a    0x46    0x25    0x09    0x40
(gdb) x /8xb &dat.ll
0xfffffffffec25: 0x7f    0xfa    0x24    0x00    0x00    0x00    0x00    0x00
(gdb)
```

Таблица, содержащая имя поля, его размер и адрес

имя поля	размер	адрес
ch	1	0xfffffffffec10
i	4	0xfffffffffec11
u	4	0xfffffffffec15
f	4	0xfffffffffec19
d	8	0xfffffffffec1d
ll	8	0xfffffffffec25

Зависимость расположения поля от его размера

Видно, что поля располагаются в памяти строго друг за другом, где каждый следующий адрес – предыдущий адрес + размер типа

Адрес переменной структурного типа

```
(gdb) p sizeof dat
$2 = 29
(gdb) x /29xb &dat
0xfffffffffec10: 0x61    0xf9    0xff    0xff    0xff    0x0c    0x00    0x00
0xfffffffffec18: 0x00    0x1f    0x85    0x1b    0x40    0x30    0x4c    0xa6
```

```
0xfffffffffec20: 0x0a    0x46    0x25    0x09    0x40    0x7f    0xfa    0x24
0xfffffffffec28: 0x00    0x00    0x00    0x00    0x00
```

Здесь видно, что структура располагается по адресу 0xfffffffffec10 такой-же адрес имеет первое поле структуры:

ch	1	0xfffffffffec10

6. Переставляя поля Вашей структуры, добейтесь, чтобы занимаемое структурой место стало минимальным. Упаковка для выполнения этого задания не используется.

Код программы

```c
#include <stdio.h>


struct data
{
    int i;
    unsigned u;
    float f;
    char ch;
    long long ll;
    double d;
    char arr[3];
};

int main(void)
{
    struct data dat;
    dat.ch = 'a';
    dat.i = -7;
    dat.u = 12;
    dat.f = 2.43;
    dat.d = 3.1432;
    dat.ll = 2423423;
    dat.arr[0] = 'y';
    dat.arr[1] = 'e';
    dat.arr[2] = 's';
    return 0;
}
```

Размер структуры

```
Breakpoint 1 at 0x7b0: file main.c, line 27.
(gdb) run
Starting program: /home/x13eav1sx/Рабочий стол/practice6/a.out

Breakpoint 1, main () at main.c:27
27          return 0;
(gdb) p sizeof(dat)
$1 = 40
(gdb)
```

Если поставить поле arr сразу после поля ch, то тогда выравнивания не будет и структура займет наименьшее количество места.

Новый порядок полей структуры

```c
struct data
{
    int i;
    unsigned u;
```

```
    float f;
    char ch;
    char arr[3];
    long long ll;
    double d;
};
```

Новый размер структуры

```
(gdb) b 27
Breakpoint 1 at 0x7b0: file main.c, line 27.
(gdb) run
Starting program: /home/x13eav1sx/Рабочий стол/practice6/a.out

Breakpoint 1, main () at main.c:27
27          return 0;
(gdb) p sizeof(dat)
$1 = 32
(gdb)
```

7. Есть ли у получившейся структуры «завершающее» выравнивание? Чему оно равно? Если нет, то почему его нет?

Для того чтобы понять есть ли у структуры выравнивание, надо сравнить размер структуры с суммой размеров ее полей. В нашем случае:
Размер структуры – 32 байта
размер int – 4 байта
размер unsigned – 4 байта
размер float – 4 байта
размер char – 1 байт
размер char[3] – 3 байта
размер double – 8 байт
размер long long – 8 байт
суммарный размер полей – 32 байта
завершающего выравнивания нет, так как выравнивание идет по наиболее тяжелому(по памяти) полю, а именно по 8-ми байтам. В силу расположения полей – int + unsigned = 8 байт, float + char + char[3] = 8 байт double и long long то же по 8 байт, таким образом выравнивать нечего.