ЗАДАНИЕ №3.4

ЧАСТЬ 1. Как представлены в памяти локальные переменные

1.

Создадим несколько локальных переменных

```
#include <stdio.h>


int main (void)
{
   short digit = 4;
   char letter = 1;
   double code = 2.2;
   int number = 3;


   return 0;
}
```

2.

Тогда на дампе они представлены следующим образом:

```
(gdb) x/1f &code
0x7fffffffdfb8:      2.2000000000000002
(gdb) x/1c &letter
0x7fffffffdfb1:      1 '\001'
(gdb) x/1h &digit
0x7fffffffdfb2:      4 '\004'
(gdb) x/1w &number
0x7fffffffdfb4:      3 '\003'
```

3.

Имя переменной	Размер переменной	Адрес переменной
letter	1	0x7fffffffdfb1

digit	2	0x7fffffffdfb2
code	8	0x7fffffffdfb8
number	4	0x7fffffffdfb4

4.

Переменные располагаются в памяти последовательно, в порядке возрастания размера типа. Переменные располагаются в ячейках, кратных размеру

ЧАСТЬ 2. Как представлены в памяти структуры

1.

Создадим структуру с несколькими полями и опишем в main переменную структурного типа. Для наглядности проинициализируем поля структуры значениями:

```
#include <stdio.h>

struct test
{
   char letter;
   double code;
   int number;
   short digit;
};

int main (void)
{
   struct test a;
   a.letter = 1;
   a.code = 2.2;
   a.number = 3;
   a.digit = 4;
```

```
    return 0;
}
```

2.

Тогда дамп памяти, содержащий структуру, выглядит так:

```
(gdb) print sizeof(a)
$1 = 24
(gdb) x/24b &a
0x7fffffffdfa0:	1	-29	-1	-1	-1	127	0	0
0x7fffffffdfa8:	-102	-103	-103	-103	-103	-103	1	64
0x7fffffffdfb0:	3	0	0	0	4	0	0	0
```

0x7fffffffdfa0 — a.letter

0x7fffffffdfa8 — a.code

0x7fffffffdfb0 — a.number

0x7fffffffdfb4 — a.digit

3.

Имя поля	Размер поля	Адрес поля
letter	1	0x7fffffffdfa0
code	8	0x7fffffffdfa8
number	4	0x7fffffffdfb0
digit	2	0x7fffffffdfb4

Поля располагаются в памяти в порядке их объявления в структуре. Поля располагаются в ячейках памяти, кратных размеру поля

4.

0x7fffffffdfa0 — адрес памяти, по которому располагается переменная структурного типа

Адрес совпадает с адресом поля, объявленного первым в структуре

5.

Реализуем упаковку структуры:

```
#include <stdio.h>

#pragma pack(push, 1)
struct test
{
   char letter;
   double code;
   int number;
   short digit;
} a;
#pragma pack(pop)

int main (void)
{
   a.letter = 1;
   a.code = 2.2;
   a.number = 3;
   a.digit = 4;

   return 0;
}
```

Дамп памяти:

```
(gdb) print sizeof(a)
```

$1 = 15

(gdb) x/15b &a

0x555555558018 <a>: 1 -102 -103 -103 -103 -103 -103 1

0x555555558020 <a+8>: 64 3 0 0 0 4 0

0x555555558018 — a.letter

0x555555558019 — a.code

0x555555558021 — a.number

0x555555558025 — a.digit

Имя поля	Размер поля	Адрес поля
a.letter	1	0x555555558018
a.code	8	0x555555558019
a.number	4	0x555555558021
a.digit	2	0x555555558025

Поля располагаются в памяти в порядке их объявления в структуре. Между полями в памяти нет дыр

0x555555558018 - адрес памяти, по которому располагается переменная структурного типа

Адрес совпадает с адресом поля, объявленного первым в структуре

6.

Расположим поля структуры так, чтобы занимаемое ею место было минимальным:

```
#include <stdio.h>

struct test
{
```

```
    char letter;
    short digit;
    int number;
    double code;
} a;

int main (void)
{
    a.letter = 1;
    a.code = 2.2;
    a.number = 3;
    a.digit = 4;

    return 0;
}
```

В таком случае размер структуры равен:

```
(gdb) print sizeof(a)
$1 = 16
(gdb) x/16b &a
0x555555558020 <a>:    1       0       4       0       3       0       0       0
0x555555558028 <a+8>:          -102    -103    -103    -103    -103    -103    1       64
```

Соответственно, занимаемое структурой место минимально, когда ее поля расположены в порядке возрастания размера, занимаемого полем (как и в случае с локальными переменными)

7.

Но несмотря на это есть завершающее выравнивание

Оно равно 1, потому что при таком расположении полей есть единственная пустая ячейка памяти между полем digit и полем letter