Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра вычислительных систем

ОТЧЕТ
по практической работе 1

по дисциплине «**Программирование**»

Выполнил:
студент гр. ИС-241
«9» марта 2023 г. _____ /Бондаренко А.А./

Проверил:
ст. преп. кафедры ВС
«__» марта 2023 г. _____ /Фульман В.О./

Оценка «_____»

Новосибирск 2023

ОГЛАВЛЕНИЕ

ЗАДАНИЕ

В приведенных программах содержатся ошибки. Необходимо с помощью отладчика локализовать и исправить их.

Задание 1

```c
#include <stdio.h>
#include <stdlib.h>

void init(int* arr, int n)
{
    arr = malloc(n * sizeof(int));
    int i;
    for (i = 0; i < n; ++i)
    {
        arr[i] = i;
    }
}

int main()
{
    int* arr = NULL;
    int n = 10;
    init(arr, n);

    int i;
    for (i = 0; i < n; ++i)
    {
        printf("%d\n", arr[i]);
    }
    return 0;
}
```

Задание 2

```c
#include <stdio.h>

typedef struct
{
    char str[3];
    int num;
} NumberRepr;

void format(NumberRepr* number)
{
    sprintf(number->str, "%3d", number->num);
}

int main()
{
    NumberRepr number = { .num = 1025 };

    format(&number);

    printf("str: %s\n", number.str);
```

```
    printf("num: %d\n", number.num);
    return 0;
}
```

Задание 3

```
#include <stdio.h>

#define SQR(x) x * x

int main()
{
    int y = 5;
    int z = SQR(y + 1);
    printf("z = %d\n", z);
    return 0;
}
```

Задание 4

```
#include <stdio.h>

void swap(int* a, int* b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

void bubble_sort(int* array, int size)
{
    int i, j;
    for (i = 0; i < size - 1; ++i) {
        for (j = 0; j < size - i; ++j) {
            if (array[j] > array[j + 1]) {
                swap(&array[j], &array[j + 1]);
            }
        }
    }
}

int main()
{
    int array[100] = {10, 15, 5, 4, 21, 7};

    bubble_sort(array, 6);

    int i;
    for (i = 0; i < 6 ; ++i) {
        printf("%d ", array[i]);
    }
    printf("\n");

    return 0;
}
```

ВЫПОЛНЕНИЕ РАБОТЫ

Задание 1

При запуске исходной программы выходит следующая ошибка:

```
jumkot@Jumkot:~/_C_/1_laba$ gcc -Wall -o ex1 ex1.c
jumkot@Jumkot:~/_C_/1_laba$ ./ex1
Segmentation fault
```

После компиляции с ключами -O0 и -g, запуска через отладчик GDB и расставления точек останова можно видеть, что при выполнении main до захода в init проблем нет:

```
Breakpoint 1, main () at ex1.c:15
15      {
(gdb) n
16          int* arr = NULL;
(gdb) n
17          int n = 10;
(gdb) n
18          init(arr, n);
(gdb) p &arr
$1 = (int **) 0x7fffffffe218
```

Внутри функции init функцией malloc выделяется память под arr, после чего указатель успешно заполняется посредством цикла for:

```
10              arr[i] = i;
1: i = 1
2: arr[i] = 0
3: arr = (int *) 0x5555555592a0
4: &arr = (int **) 0x7fffffffe1e8
(gdb) n
8           for (i = 0; i < n; ++i)
1: i = 1
2: arr[i] = 1
3: arr = (int *) 0x5555555592a0
4: &arr = (int **) 0x7fffffffe1e8
(gdb) n
10              arr[i] = i;
1: i = 2
2: arr[i] = 0
3: arr = (int *) 0x5555555592a0
4: &arr = (int **) 0x7fffffffe1e8
```

Но адрес arr внутри init и за её пределами, в main, не совпадают, что логично. Однако именно это вызывает ошибку – указатель заполняется только внутри функции init, потому что она не возвращает изменённого значения:

```
main () at ex1.c:21
21          for (i = 0; i < n; ++i)
(gdb) p &arr
$2 = (int **) 0x7fffffffe218
(gdb) p arr[0]
Cannot access memory at address 0x0
(gdb) n
23              printf("%d\n", arr[i]);
(gdb) n

Program received signal SIGSEGV, Segmentation fault.
```

После изменения типа возвращаемого значения с void на int* и его дальнейшего присваивания самому arr программа работает:

```
4   int* init(int* arr, int n)
5   {
6       arr = malloc(n * sizeof(int));
7       int i;
8       for (i = 0; i < n; ++i)
9       {
10          arr[i] = i;
11      }
12      return arr;
13  }
```

```
jumkot@Jumkot:~/_C_/1_laba$ gcc -Wall -o ex1 ex1.c
jumkot@Jumkot:~/_C_/1_laba$ ./ex1
0
1
2
3
4
5
6
7
8
9
```

```
Breakpoint 1, main () at ex1.c:19
19          arr = init(arr, n);
(gdb) p &arr
$1 = (int **) 0x7fffffffe218
(gdb) p arr[0]
Cannot access memory at address 0x0
(gdb) n

Breakpoint 2, main () at ex1.c:21
21          for (int i = 0; i < n; ++i)
(gdb) p &arr
$2 = (int **) 0x7fffffffe218
(gdb) p arr[0]
$3 = 0
```

Задание 2

В программе содержится структура из двух полей: строку из трёх элементов и переменную типа int. По ходу выполнения в int-овую переменную записывается целочисленное значение, которое затем записывается в первое поле структуры в виде строки с помощью sprintf.
Сообщение об ошибке при запуске программы отсутствует, но есть очевидная несостыковка: к концу выполнения программы number.num приобрело значение 1024, хотя в начале было задано, как 1025, и явной замены не было:

```
jumkot@Jumkot:~/_C_/1_laba$ gcc -Wall -o ex2 ex2.c
jumkot@Jumkot:~/_C_/1_laba$ ./ex2
str: 1025
num: 1024
```

При использовании отладчика становится видно, что значение изменяется после прохода через функцию format:

```
Breakpoint 2, format (number=0x7fffffffe210) at ex2.c:11
11          sprintf(number->str, "%3d", number->num);
(gdb) p number.str
$3 = "\000\000"
(gdb) p number.num
$4 = 1025
(gdb) n
12      }
(gdb) p number.str
$5 = "102"
(gdb) p number.num
$6 = 1024
```

Используя команду ptype с ключом /o, можно узнать размер структуры и отдельно каждого из её полей. Поле str занимает 3 байта, потом следует один байт для выравнивания и поле num, имеющее тип int и занимающее 4 байта.

```
(gdb) ptype /o number
type = struct {
/*    0      |     3 */    char str[3];
/* XXX  1-byte hole  */
/*    4      |     4 */    int num;

                            /* total size (bytes):    8 */
                          }
```

Использование команды x/ показывает, что хранится в каждом из полей: сначала в str в двоичном виде одни нули, а после заполнения там хранится число 1025 в двоичном виде. Также при просмотре в формате char видно, что само число по символу заняло четыре байта, и следом идёт символ конца строки:

```
(gdb) x/4cb number.str
0x7fffffffe1e0: 0 '\000'        0 '\000'        0 '\000'        0 '\000'
(gdb) x/2tw number.str
0x7fffffffe1e0: 00000000000000000000000000000000        00000000000000000000001000000001
(gdb) n
20          printf("str: %s\n", number.str);
(gdb) x/4cb number.str
0x7fffffffe1e0: 49 '1'  48 '0'  50 '2'  53 '5'
(gdb) x/6cb number.str
0x7fffffffe1e0: 49 '1'  48 '0'  50 '2'  53 '5'  0 '\000'        4 '\004'
(gdb) x/2tw number.str
0x7fffffffe1e0: 00110101001100100011000000110001        00000000000000000000001000000000
```

То есть заполнение строки происходит без ошибок, но при этом значение не помещается в поле str и задействуется область памяти, выделенная под хранение целочисленного значения num, и поэтому после прохода через функцию format его значение меняется.

Чтобы исправить эту ошибку, нужно выделить больше памяти для хранения строки – чтобы туда вместилось всё число вместе с символом конца строки, то есть минимум 5.

```
(gdb) ptype /o number
type = struct {
/*    0      |     5 */    char str[5];
/* XXX  3-byte hole  */
/*    8      |     4 */    int num;

                           /* total size (bytes):   12 */
                         }
```

```
jumkot@Jumkot:~/prog/1_laba$ gcc -Wall -o ex2 ex2.c
jumkot@Jumkot:~/prog/1_laba$ ./ex2
str: 1025
num: 1025
```

Задание 3

Программа успешно запускается, но результат вычислений очевидно неверен: при указанных входных данных z = (y + 1) * (y + 1) = (5 + 1) * (5 + 1) = 6 * 6 = 36

```
jumkot@Jumkot:~/_C_/1_laba$ gcc -Wall -o ex3 ex3.c
jumkot@Jumkot:~/_C_/1_laba$ ./ex3
z = 11
```

Чтобы найти проблему, нужно скомпилировать программу с ключом -g3, включающим дополнительную отладочную информацию (в том числе разворачивание макросов). После запуска в отладчике и использования команды macro expand становится видно, что макрос разворачивается не так, как предполагается:

```
Breakpoint 1 at 0x1149: file ex3.c, line 6.
(gdb) run
Starting program: /home/jumkot/prog/1_laba/ex3

Breakpoint 1, main () at ex3.c:6
6       {
(gdb) macro expand SQR(5 + 1)                 ):
expands to: 5 + 1 * 5 + 1
```

Чтобы исправить это, нужно заключить оба x внутри макроса в скобки – так будут сначала вычисляться значения внутри скобок, а уже потом перемножаться между собой.

```
jumkot@Jumkot:~/_C_/1_laba$ gcc -Wall -o ex3 ex3.c
jumkot@Jumkot:~/_C_/1_laba$ ./ex3
z = 36
```

Задание 4

Здесь представлена некорректно работающая пузырьковая сортировка:

```
int main()
{
    int array[100] = {10, 15, 5, 4, 21, 7};

    bubble_sort(array, 6);
```

```
jumkot@Jumkot:~/_C_/1_laba$ gcc -Wall -o ex4 ex4.c
jumkot@Jumkot:~/_C_/1_laba$ ./ex4
4 0 5 7 10 15
```

При запуске отладчика GDB становится ясно, что функция swap работает корректно, и проблема заключается в алгоритме сортировки на моменте достижения конца массива:

```
16                      swap(&array[j], &array[j + 1]);
1: i = 0
2: j = 5
3: array[j] = 21
4: array[j + 1] = 0
(gdb) n
14              for (j = 0; j < size - i; ++j) {
1: i = 0
2: j = 5
3: array[j] = 0
4: array[j + 1] = 21
```

Из-за неправильного указания ограничения во вложенном цикле for (при первом проходе i равняется нулю) последний элемент сравнивается с элементом конца строки, из-за чего и появляется 0 в отсортированном массиве. Необходимо добавить «-1» в условие цикла:

```
int i, j;
for (i = 0; i < size - 1; ++i) {
    for (j = 0; j < size - i - 1; ++j) {
        if (array[j] > array[j + 1]) {
            swap(&array[j], &array[j + 1]);
```

После внесённых изменений программа выдаёт верно отсортированный массив:

```
jumkot@Jumkot:~/_C_/1_laba$ gcc -Wall -o ex4 ex4.c
jumkot@Jumkot:~/_C_/1_laba$ ./ex4
4 5 7 10 15 21
```

ПРИЛОЖЕНИЕ

ex1.c

```c
#include <stdio.h>
#include <stdlib.h>

int* init(int* arr, int n)
{
    arr = malloc(n * sizeof(int));
    int i;
    for (i = 0; i < n; ++i)
    {
        arr[i] = i;
    }
    return arr;
}

int main()
{
    int n = 10;
    int* arr = NULL;
    arr = init(arr, n);

    for (int i = 0; i < n; ++i)
    {
        printf("%d\n", arr[i]);
    }
    return 0;
}
```

ex2.c

```c
#include <stdio.h>

typedef struct
{
    char str[5];
    int num;
} NumberRepr;

void format(NumberRepr* number)
{
    sprintf(number->str, "%3d", number->num);
}

int main()
{
    NumberRepr number = { .num = 1025 };

    format(&number);

    printf("str: %s\n", number.str);
    printf("num: %d\n", number.num);
    return 0;
}
```

ex3.c

```c
#include <stdio.h>

#define SQR(x) (x) * (x)

int main()
{
    int y = 5;
    int z = SQR(y + 1);
    printf("z = %d\n", z);
    return 0;
}
```

ex4.c

```c
#include <stdio.h>

void swap(int* a, int* b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

void bubble_sort(int* array, int size)
{
    int i, j;
    for (i = 0; i < size - 1; ++i) {
        for (j = 0; j < size - i - 1; ++j) {
            if (array[j] > array[j + 1]) {
                swap(&array[j], &array[j + 1]);
            }
        }
    }
}

int main()
{
    int array[100] = {10, 15, 5, 4, 21, 7};

    bubble_sort(array, 6);

    int i;
    for (i = 0; i < 6 ; ++i) {
        printf("%d ", array[i]);
    }
    printf("\n");

    return 0;
}
```