Ильченко Ева ИУ7-24Б

Отчет по Заданию №3.1. Отладка

Цель: проработка на практике умения самостоятельно использовать отладчик для поиска причин ошибок в приложении.

Задание №1

Программа 1

Листинг программы task_01.c

```
#include <stdio.h>

long long unsigned factorial(unsigned n);

int main(void)
{
    unsigned n;
    long long unsigned result;

    printf("Input n: ");
    if (scanf("%u", &n) != 1)
    {
        printf("Input error");
        return 1;
    }

    result = factorial(n);
    printf("factorial(%u) = %llu\n", n, result);
    return 0;
}

long long unsigned factorial(unsigned n)
{
    long long unsigned result = 1;

    while (n--)
        result *= n;

    return result;
}
```

Запустив отладчик gdb, я установила точку останова на функции factorial(unsigned n). Далее я проходила пошагово по программе командой step и после каждого шага смотрела на значения переменных n и result командами print n и print result. Программа выдает неверный ответ, так как она сначала отнимает от n единицу, а уже затем умножает result на новую n. Таким образом, когда n = 1, то из n отнимается единица и result умножается уже на ноль.

Листинг программы task_01.c после исправления ошибки

```c
#include <stdio.h>

long long unsigned factorial(unsigned n);

int main(void)
{
    unsigned n;
    long long unsigned result;

    printf("Input n: ");
    if (scanf("%u", &n) != 1)
    {
        printf("Input error");
        return 1;
    }

    result = factorial(n);
    printf("factorial(%u) = %llu\n", n, result);
    return 0;
}

long long unsigned factorial(unsigned n)
{
    long long unsigned result = 1;

    while (n)
    {
        result *= n;
        n--;
    }

    return result;
}
```

Программа 2

Листинг программы task_02.c

```c
#include <stdio.h>

#define N 5

double get_average(const int a[], size_t n);
int get_max(const int *a, size_t n);

int main()
{
    int arr[N];
    size_t i;
```

```
    printf("Enter %d numbers:\n", N);
    for (i = 0; i < N; i++)
    {
        printf("Enter the next number: ");
        if (scanf("%d", &arr[1]) != 1)
        {
            printf("Input error");
            return 1;
        }
    }

    for (i = 1; i < N; i++)
        printf("Value [%zu] is %d\n", i, arr[i]);

    printf("The average is %g\n", get_average(arr, N));
    printf("The max is %d\n", get_max(arr, N));
    return 0;
}

double get_average(const int a[], size_t n)
{
    double temp = 0.0;

    for (size_t i = 0; i > n; i++)
        temp += a[i];
    temp /= n;

    return temp;
}

int get_max(const int *a, size_t n)
{
    int max = a[0];

    for (size_t i = 1; i < n; i++)
        if (max > a[i])
            max = a[i];

    return max;
}
```

Запустив программу и посмотрев на ее выходные данные, можно сделать вывод, что программа неверно записывает числа в массив, поэтому я установила точку останова на строке 16. Затем я пошагово проходила по циклу командой step, вводила числа и смотрела на элементы массива командой print arr. После двух итераций я заметила, что числа записываются только в ячейку с индексом 1.

Далее я установила точки останова на функции get_average и get_max. Пошагово проходя по телу функции get_average, было замечено, что функция не заходит в цикл for. Выведя значения i и n и посмотрев на условие выполнения цикла, я выяснила, что условие цикла было записано неверно и вместо i>n нужно написать i<n.
Проходя по телу цикла функции get_max и выводя значения a[i] и max после каждого шага, было замечено, что большие значения не записываются в переменную max.
Также при выводе результата я заметила, что программа выводит не все элементы массива. При пошаговом прохождении по коду программы было выяснено, что это происходит, так как отсчет начинается с индекса 1, а не 0

Листинг программы task_02.c после исправления ошибок

```
#include <stdio.h>

#define N 5

double get_average(const int a[], size_t n);
int get_max(const int *a, size_t n);

int main()
{
    int arr[N];
    size_t i;

    printf("Enter %d numbers:\n", N);
    for (i = 0; i < N; i++)
    {
        printf("Enter the next number: ");
        if (scanf("%d", &arr[i]) != 1)
        {
            printf("Input error");
            return 1;
        }
    }

    for (i = 0; i < N; i++)
        printf("Value [%zu] is %d\n", i, arr[i]);

    printf("The average is %g\n", get_average(arr, N));
    printf("The max is %d\n", get_max(arr, N));
    return 0;
}

double get_average(const int a[], size_t n)
{
```

```
    double temp = 0.0;

    for (size_t i = 0; i < n; i++)
        temp += a[i];
    temp /= n;

    return temp;
}

int get_max(const int *a, size_t n)
{
    int max = a[0];

    for (size_t i = 1; i < n; i++)
        if (max < a[i])
            max = a[i];

    return max;
}
```

Программа 3

Листинг программы task_03.c

```
#include <stdio.h>

int div(int a, int b);

int main(void)
{
    int a = 5, b = 2;

    printf("%d div %d = %d\n", a, b, div(a, b));

    a = 10;
    b = 0;

    printf("%d div %d = %d\n", a, b, div(a, b));

    return 0;
}

int div(int a, int b)
{
    return a / b;
}
```

После запуска отладчика командой run была выведена ошибка, которая происходит в функции div. Я установила точку останова на строчку 7 и

пошагово проходила по программе. При втором заходе в функцию div была выдана ошибка. Выведя переменные, было обнаружено, что функция пытается произвести деление на ноль.

Листинг программы task_03.c после исправления ошибок

```
#include <stdio.h>

int div(int a, int b);

int main(void)
{
    int a = 5, b = 2;

    printf("%d div %d = %d\n", a, b, div(a, b));

    a = 10;
    b = 0;

    printf("%d div %d = %d\n", a, b, div(a, b));

    return 0;
}

int div(int a, int b)
{
    return a / b;
}
```

Вопросы:

1. Программу нужно компилировать к ключом -g, чтобы добавить отладочную информацию к исполняемому файлу. Если не добавить этот ключ, то отладчик не сможет начать отладку, так как не будет видеть отладочной информации
2. Чтобы запустить программу под отладчиком, необходимо ее скомпилировать, а затем передать исполняемый файл в gdb

   ```
   gcc-13 -std=c99 -Wall -Werror task_03.c -o app.exe
   gdb app.exe
   ```

   Затем необходимо ввести команду `run`

Чтобы досрочно завершить выполнение программы, необходимо нажать Control+C. Также для завершения программы можно использовать команду `exit`

3. С помощью команды `bt` можно посмотреть, в каком месте было остановлено выполнение программы
4. Узнать значение переменной можно при помощи команды print, например: `print x`
   Изменить значение переменной можно при помощи команды set, например: `set x=10`. Также значение можно изменить при помощи команды `print`, например: `print x=10`
5. Пошагово программу можно выполнить при помощи команд `next` и `step`. Команда `next` выполняет следующую инструкцию в программе, пропуская вызовы функций, а команда `step` выполняет следующую инструкцию в программе, включая вызовы функций.
6. Чтобы понять, какая последовательность вызовов функций привела к остановке программы на точке останова, можно воспользоваться командой `bt`
7. Точку останова можно установить командами:
   ```
   break [имя_файла:]номер_строки
   break имя_функции
   break номер_строки
   tbreak номер_строки
   ```
8. Временная точка останова - это точка останова, которая действует один раз и автоматически удаляется после срабатывания. В отладчике gdb она устанавливается командой `tbreak`
9. Чтобы временно выключить точку останова используется команда `disable` с номером точки останова. Чтобы обратно включить точку останова используется команда `enable` с номером точки останова. Чтобы пропустить некоторое количество срабатываний точки останова используется команда `ignore` с номером точки останова и количеством срабатываний, которое пропускается
10. Условие остановки на точке останова задается командами
    ```
    break номер_строки if условие
    condition номер_точки_останова [условие]
    ```
11. Точки останова используются для приостановки выполнения программы в определенном месте кода, а точки наблюдения используются для отслеживания изменений значений переменных

12. Удобно использовать точку наблюдения в ситуации, когда какая-то переменная изменяет в условии, которое находится в теле цикла. Если она при выполнении программы принимает неверное, значение, то можно установить на ней точку наблюдения, чтобы сразу обнаружить, когда она принимает неверное значение
13. Для просмотра содержимого области памяти используется команда `x /Nxb &имя_переменной`. Эта команда позволяет просматривать память в различных форматах и указывать количество элементов для отображения

Задание №2

ОС macOS Ventura 13.0, gcc 13.2.0

Тип	**Размер, байты**
char	1
int	4
unsigned	4
short int	2
long int	8
long long	8
int32_t	4
int64_t	8

Windows 10, gcc 13.2.0

Тип	**Размер, байты**
char	1
int	4
unsigned	4
short int	2
long int	4
long long	8
int32_t	4
int64_t	8

Задание №3

Описание переменной	**Представление в памяти**
char c1 = ‘a’;	0x61
char c1 = -100;	0x9c
int i1 = 5;	0x00000005
int i2 = -5;	0xfffffffb
unsigned a1 = 5;	0x00000005
unsigned a2 = -5;	0xfffffffb
long long b1 = 10;	0x000000000000000a
long long b2 = -10;	0xfffffffffffffff6

Для переменных типа char их буквенное представление в памяти записывается в виде их кода в ASCII. Если в переменную типа char записывается отрицательное число, то оно переводится в восьмиразрядное двоичное число, где первый бит отвечает за знак
Положительные переменные типа int представляются, как и их значения. Отрицательные переменные типа int переводятся в 32-х разрядное двоичное число, где первый бит отвечает за знак
Положительные переменные типа unsigned представляются, как и их значения. Так как беззнаковые не могут хранить отрицательные числа, то оно будет представлено, как положительное число, полученное путем вычитания модуля переменной из наибольшего беззнакового числа на данной машине. Далее полученное число будет переведено в двоичную 32-х разрядную систему счисления
Переменные типа long long хранятся аналогично переменным типа int. Положительные представляются, как их значения. Отрицательные переводятся в 64-х разрядное двоичное число, где первый бит отвечает за знак.

Задание №4

Пусть задан массив

```
int a = {1, 2, 3};
```

Тогда элементы массива могут иметь следующие адреса

```
a[0] = 0x7ff7bfeff284
a[1] = 0x7ff7bfeff288
a[2] = 0x7ff7bfeff28c
```

То есть адреса их расположения отличаются на 4 бита. Это место нужно, чтобы хранить значение типа int.

Пусть задано

```
int a = {1, 2, 3};
int *p;
p = a;
```

Тогда при выполнении операции сложения указателя с целым числом

```
p = 0x00007ff7bfeff27c
p + 1 = 0x00007ff7bfeff280
p + 2 = 0x00007ff7bfeff284
```

При прибавлении единицы к указателю адрес увеличивается на 4 единицы, так как указатель указывает на массив, содержащий целые числа, которые занимают в памяти 4 байта, то есть при прибавлении числа указатель начинает указывать на следующий элемент массива

```
*p = 1
*(p+1) = 2
*(p+2) = 3
```

Задание №5

Сравнение команд gdb и VS Code для отладки кода

Команда	**gdb**	**VS Code**
Начать отладку	gdb имя_исполняемого_файла	F5 / кнопка начала отладки
Остановить отладку	exit / q	SHIFT + F11 / кнопка остановки отладки
Установить точку остано	b номер_строки	Навести курсор на нужную строку и поставить красную точку
Установить временную точку останова	tbreak номер_строки	Ctrl + F10
Удалить точку останова	d номер_точки_останова	Навести курсор на нужную строку и нажать на красную точку
Выполнить строку, не входя в функцию, если она есть	step	F11
Выполнить строку, войдя в функцию, если она есть	next	F10
Продолжить отладку	continue	F9