Отчет по Проектно-технологической практике (тестированию, отладке и профилированию ПО)
Задание №3
Отладка

Задание №1

Найдем ошибки, допущенные в программе номер 1 при помощи отладчика **gdb**. Программа должна находить факториал числа. При первоначальном запуске видно, что программа работает неверно. Поставим при помощи команды **break** точку остановы на цикл в котором перемножаются числа, тем самым мы сможем отслеживать процесс получения факториала. Число изначально уменьшается на 1, что уже является ошибкой при нахождении факториала, а так же нет условия, которое вовремя останавливает цикл, не давая перемножать число на 0.

Код программы с ошибкой

```
#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;
}
```

Исправленый код программы

```
#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 != 0)
   {
      result *= n;
      n--;
   }
   return result;
}
```

Найдем ошибки, допущенные в программе номер 2, Программа направлена на поиск среднего и максимального значения массива. Программа получает на вход 5 чисел, которые должна записывать в массив. Поставим точку остановы при помощи команды **break** на цикл, который отвечает за запись чисел в массив. Запустим программу и при помощи команды step будем проверять работу программы построчно. Можно заметить, что все числа помещаются на позицию второго элемента, что является ошибкой. Исправим эту ошибку, заменив индекс “1” на переменную цикла. После этого следует цикл для вывода элементов, из вывода можно понять, что в цикле неверно указан элемент, с которого нужно начать вывод, исправим эту ошибку. Сохраним изменения, проверим исправленые блоки кода. Все работает корректно, можно проверять дальше. Далее, в программе следует вызов функции для поиска среднего. Поставим точку остановы на цикл и при помощи команды **step** будем проверять работу построчно. Запустив прорамму, можно понять, что в цикл, программа не заходит, тк неверно указано условие для переменной. Исправим ошибку и проверим изменения. Все работает верно, можно идти дальше. После функции нахождения среднего вызывается функция для нахождения максимального элемента, поставим точку остановы аналогично прошлой функции. Можно заметить, что в цикле неверно указан индекс, с которого нужно начинать проверку, а так же допущена ошибка в условии нахождения максимального. Исправив и сохранив изменения программа стала работать верно.

Код программы с ошибкой

```
#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;
}
```

Исправленный код программы

```
#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 = 0; i < n; i++)
        if (max < a[i])
            max = a[i];
    return max;
}
```

Вопросы

1. **С какими ключами нужно скомпилировать программу, чтобы можно было пользоваться отладчиком gdb? Что произойдёт, если собрать программу без этого ключа и загрузить её в gdb?**

   Для того чтобы пользоваться отладчиком, необходимо скомпилировать программу с ключом **-g**. Если собрать программу без этого ключа, то отладчик не будет видеть имена переменных, функций, а так же не будут работать некоторые функции.

2. **Как запустить программу под отладчиком? Как досрочно завершить её работу (предположим, что Вы остановили выполнение программы на точке останова и дальше выполнять программу не нужно)?**

   Для того чтобы запустить программу под откладчиком, необходимо сначала собрать ее с определенным ключом, а потом воспользоваться командой **gdb [имя файла]**.

```
gcc -std=c99 -Wall -Werror -g -o app.exe main.c
gdb app.exe
```

   Для завершения работы отладчика досрочно необходимо воспользоваться командой **q** или **quit** и в дальнейшем согласиться на завершение работы программы при помощи команды **y**.

3. **Выполнение Вашей программы было остановлено на какой-то из точек останова. Как посмотреть в каком месте Вашей программы Вы остановились?**

   Информацию о том, где остановился отладчик можно увидеть в командной строке. Там будет указан номер строки и имя функции в которой он находится.

4. **С помощью какой команды можно посмотреть значение переменной? Изменить значение переменной?**

   Посмотреть значение переменной можно при помщи команды **print [имя переменной]**. А для того чтобы задать значение переменной необходимо воспользоваться командой **set [имя переменной] = [значение]**.

5. **С помощью каких команд можно выполнить программу в пошаговом режиме? Чем отличаются эти команды друг от друга?**

   Для пошагового выполнения необходимо запустить программу при помощи комнады **run**, далее воспользовавашись командой **finish**, **step** или **next**. Команда **step** выполняет текущую строку программы и переходит к следующей. Команда **next** похожа на **step**, однако команда **step** заходит в функции, а команда **next** переходит к следущей строке. Команда **finish** выполняет команду **next** без остановок, пока не дойдет до конца текущей функции.

6. **Выполнение Вашей программы было остановлено на какой-то из точек останова. Как понять, какая последовательность вызовов функций привела Вас сюда?**

   Посмотреть последовательность вызова функций можно при помощи команды **bt.**

7. **Как можно установить точку останова в программе?**

   Точку остановы на строку можно установить при помощи команды **break [номер строки].**

8. **Какая точка останова считается временной?**

   Точка остановы является временной, если после ее прохождения она перестает быть точкой остановы. Создать временную точку остановы можно при помощи команды **tbreak**.

9. **Как временно выключить/включить точку останова или пропустить некоторое количество срабатываний?**

   Для временного отключения точки остановы необходимо ввести команду **ignore [номер строки] [количество итераций, которое точка не будет срабатывать]**

10. **Как задать условие остановки на точке останова?**

    Чтобы задать условие точки остановы, необходимо к команде **break** добавить команду **if [условие]**.

11. Чем отличаются точки наблюдения от точек останова?

Точки наблюдения (watchpoints) позволяют отслеживать изменения значения переменной или памяти в определенной точке программы. Когда значение переменной изменяется, программа останавливается и дебаггер уведомляет об этом. Точки останова (breakpoints) используются для приостановки выполнения программы в определенном месте. Когда программа достигает точки останова, она приостанавливается, и дебаггер позволяет проверить состояние программы на этом этапе.Таким образом, основное отличие между точками наблюдения и точками останова в GDB состоит в том, что точки наблюдения используются для отслеживания изменения значений, а точки останова для приостановки выполнения программы.

12. Приведите пример ситуации, в которой удобно использовать точку наблюдения.

Предположим, у нас есть программа, которая должна сортировать массив чисел. Мы установили точки останова для отслеживания процесса сортировки, но заметили, что на каком-то этапе значение переменной "temp", которое используется для сортировки, неправильно меняется, что приводит к некорректным результатам. В этой ситуации нам может потребоваться установить точку наблюдения на переменную "temp", чтобы следить за ее изменениями в процессе выполнения программы.

13. С помощью какой команды можно посмотреть содержимое области памяти?

Для отображения содержимого области памяти используется команда **x/[размер][формат] &[имя переменной]**.

Задание №2

Для решения данной задачи напишем простую программу, которая будет выводить размер типа переменной.

```
#include <stdio.h>
#include <stdint.h>

int main()
{
    printf("sizeof(char)  = %ld\n", sizeof(char));
    printf("sizeof(int)   = %ld\n", sizeof(int));
    printf("sizeof(unsigned)   = %ld\n", sizeof(unsigned));
    printf("sizeof(short int) = %ld\n", sizeof(short int));
    printf("sizeof(long int) = %ld\n", sizeof(long int));
    printf("sizeof(long long) = %ld\n", sizeof(long long));
    printf("sizeof(int32_t) = %ld\n", sizeof(int32_t));
    printf("sizeof(int64_t) = %ld\n", sizeof(int64_t));

}
```

Ubuntu 22.04.3 LTS
gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0

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

Windows 10 Home Версия 22H2 сборка ОС 19045, 4170
gcc (Rev3, Built by MSYS2 project) 13.2.0

Тип	Размеры, 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 c2 = -100;	0x9c
char c3 = 5;	0x05
int i1 = 19;	0x13 0x00 0x00 0x00
int i2 = -11;	0xf5 0xff 0xff 0xff
unsigned u1 = 42;	0x2a 0x00 0x00 0x00
unsigned u2 = -104;	0x98 0xff 0xff 0xff
long long l1 = 1336357643;	0x0b 0x33 0xa7 0x4f 0x00 0x00 0x00 0x00
long long l2 = -5484588;	0xd4 0x4f 0xac 0xff 0xff 0xff 0xff 0xff

1. Каждый блок представляет из себя 1 байт.
2. **0x –** показывает, что это представление в 16сс.
3. Последующие символы отвечают за адрес, каждый символ раен 4 битам.
4. Положительные числа просто переводятся в 16сс.
5. Отридцательные числа переводятся в 2сс, 1 меняется на 0, а 0 на 1. Потом дописывается 1, показывающая, что число отридцательное. Полученное число переводится в 16сс.

Задание 4

Для выполнения этого задания создадим целочисленный массив из 10 элементов. Для того чтобы посчитать размер этого массива, необходимо перемножить количество элементов на размер одного элемента. (4 * 10 = 40 байт)

Воспользуемся командой x/40xb &[имя массива]

```
0x7fffffffde30:    0x00  0x00  0x00  0x00  0x01  0x00  0x00  0x00
0x7fffffffde38:    0x02  0x00  0x00  0x00  0x03  0x00  0x00  0x00
0x7fffffffde40:    0x04  0x00  0x00  0x00  0x05  0x00  0x00  0x00
0x7fffffffde48:    0x06  0x00  0x00  0x00  0x07  0x00  0x00  0x00
0x7fffffffde50:    0x08  0x00  0x00  0x00  0x09  0x00  0x00  0x00
```

В каждой строке по 8 элементов, значит по 8 байт(2 элемента). Значит 4 блока в строке хранят в себе 1 элемент массива.

Для того чтобы продемонстрировать особенности сложения указателя и числа на примере массива, создадим переменную, которая будет хранить адрес начального элемента массива, посмотрим ее адрес и значение до и после сложения с числом 4.

Код программы

```
#include <stdio.h>

#define N 10

int main()
{
    int arr[N];
    for (int i = 0; i < N; i++)
        arr[i] = i;
    int *pa = arr;
    pa += 4;
}
```

Для проверки значений поставим точки остановы на строку создания указателя и на строку сложения числа с указателем.

До сложения

```
(gdb) print pa
$1 = (int *) 0x10101000000
(gdb) p *pa
$2 = 0
```

После сложения

```
(gdb) print pa
$3 = (int *) 0x7fffffffde30
(gdb) print *pa
$5 = 4
```

Проанализировав работу программы, можно понять, что при сложении указателя с числом, указатель увеличивается на размер типа указателя, умноженного на число, которое прибавлялось. (pa = pa + sizeof([type]) * [num])