Федеральное агентство связи
Федеральное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(ФГОУ ВПО «СибГУТИ»)

Факультет ИВТ

Кафедра вычислительных систем

Лабораторная работа №2

"Отладка программ"

Выполнил:
студент гр. ИВ - 521
Прокопенко Р. П.

Проверила:
Старший преподаватель Кафедры ВС
Перышкова Е.Н.

Новосибирск
2016

Цель

Целью лабораторной работы является получение навыков отладки программ на примере использования отладчика GDB.

Задание

В приведённых программах содержатся ошибки. С помощью отладчика необходимо локализовать и исправить их.

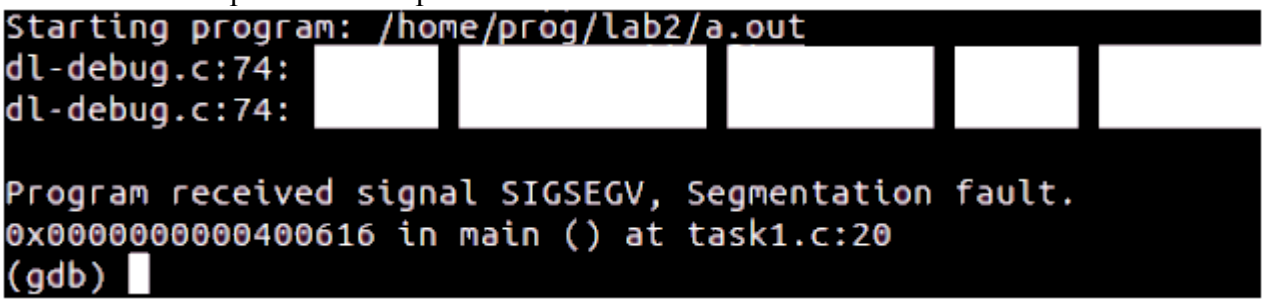
Тестовые данные

Задание 1

Исходная программа	Исправленная программа

#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;
}
	#include <stdio.h>
	#include <stdlib.h>
	int * init(int* arr, int n) {
	  arr = malloc(n * sizeof(int));
	  if (arr == NULL) {
	  printf("ERROR\n");
	  exit(1);
	  }
	  int i;
	  for (i = 0; i < n; ++i)
	  arr[i] = i;
	  return arr;
	}
	
	int main() {
	  int* arr = NULL;
	  int n = 10;
	  arr = init(arr, n);
	  int i;
	  for (i = 0; i < n; ++i)
	  printf("%d\n", arr[i]);
	  return 0;
	}

При компиляции и запуске исходной программы ошибки не происходит, но при запуске мы ожидаем вывод массива, но этого не происходит. При запуске через отладчик происходит ошибка сегментирования на строке 20.



Делаем breakpoint на 20 строку и проверяем адрес arr.

```
(gdb) n
(gdb) p arr
$1 = (int *) 0x0
(gdb) n
n
(gdb) n
n
(gdb) p arr
p arr
$2 = (int *) 0x0
(gdb)
```

Как видим, после выполнения *init()* адрес *arr* не изменился.
Результат после исправления программы.

```
MS@7817:~/prog/lab2$ ./a.out
0
1
2
3
4
5
6
7
8
9
```

1) Сделали процедуру *init()* функцией, возвращающей значение типа *int и стали возвращать из нее указатель на массив *arr*.
2) Предусмотрели ситуацию, если память под массив не выделилась: в функции *init()* выводим сообщение об ошибке и завершаем работу программы.
3) В *main* инициализируем *arr* с помощью функции *init()*.

Задание 2

Исходная программа	**Исправленная программа**

#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;
}
	#include <stdio.h>
	typedef struct {
	  char str[3];
	  int num;
	} NumberRepr;
	void format(NumberRepr* number) {
	  sprintf(number->str, **"%d"**, number->num);
	  number->str[3] = '\0';
	}
	int main() {
	  NumberRepr number = { .num = 1025 };
	  format(&number);
	  printf("str: %s\n", number.str);
	  printf("num: %d\n", number.num);
	  return 0;
	}

Компилируем и запускаем программу. Не соответствует ожидаемым выходным данным: “str: 102”, “num:1025”.

```
MS@7817:~/prog/lab2$ gcc -g -O0 task2.c

MS@7817:~/prog/lab2$ ./a.out
str: 1025
num: 1024
```

Запускаем отладчик, и проверяем программу пошагово. Breakpoint установим на *main*. В функции *format* некорректно обрабатывается *number->num*.

```
(gdb) s
(gdb) n
(gdb) p number->num
$2 = 1024
(gdb) n
str: 1025
(gdb)
```

Из поля структуры *number->str* считывается весь *number->num*. Это связанно с тем что в конце *str* нет символа конца строки нуля-терминатора «\0».
Исправленная версия прогаммы после компиляции и запуска.

```
$ gcc -g -O0 task2.c
$ ./a.out
str: 102
num: 1025
```

1) Изменили формат (второй аргумент *sprintf*) с трезнакового int на простой int.
2) Четвертый символ в строке *str* делаем нуль-терминатор, чтобы в строке остались три цифры из *num*.

Задание 3

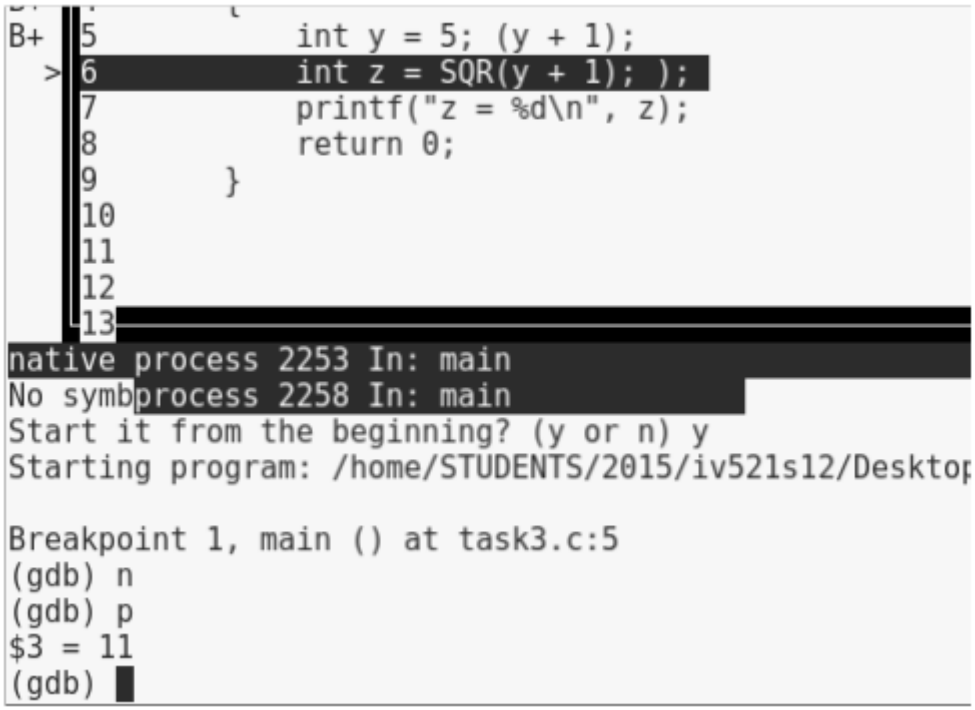
Исходная программа	**Исправленная программа**

#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;
}
	#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;
	}

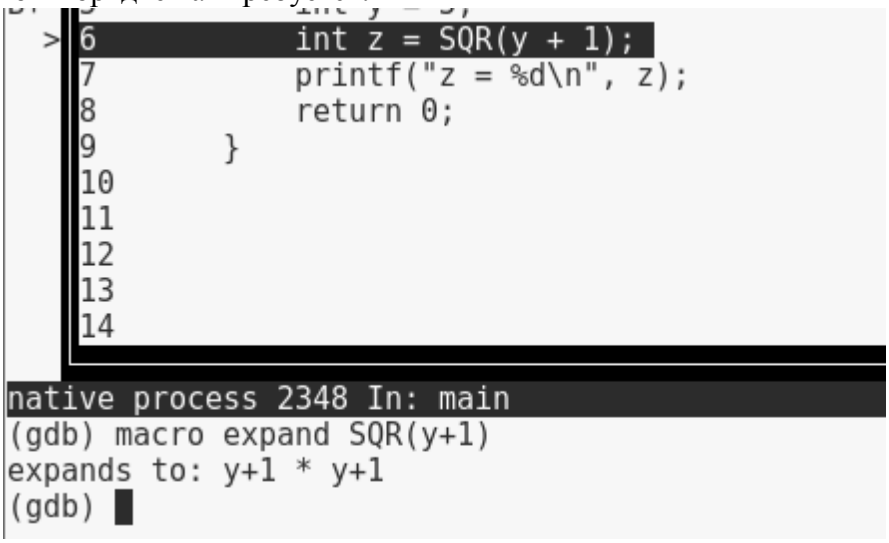
Компилируем и запускаем исходную программу. Ожидается результат, в котором *Z* должно было быть равным 36, а получилось 11.

```
$ gcc -g3 -O0 task3.c
$ ./a.out
z = 11
```

Запускаем через отладчик. Видим что макрос *SQR* сработал некорректно.



Посмотрим как представляется макрос в программе. Видим, что операции выполняются не в том порядке как требуется.



1) Изменили описание макроса. Аргументы которые принимает макрос, записали в скобках, чтобы очередность арифметических действий была соблюдена.

Задание 4

Исходная программа	**Исправленная программа**

#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]);
	#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 - 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;
}
```

```
        }
      }
   }
}
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;
}
```

После компиляции и запуска программы видим что массив не отсортирован.

```
[iv521s12@mwp05 ~]$ mc
$ gcc -g -O0 task4.c
$ ./a.out
4 0 5 7 10 15
```

Запускаем отладчик.

```
8       void bubble_sort(int* array, int size) {
9           int i, j;
10          for (i = 0; i < size - 1; ++i) {
>11             for (j = 0; j < size - i; ++j) {
12                  if (array[j] > array[j + 1]) {
13                      swap(&array[j], &array[j + 1]
14                  }
15              }
16          }
17      }
native process 2670 In: bubble_sort


Breakpoint 1, main () at task4.c:19
(gdb) s
(gdb) s
bubble_sort (array=0x7fffffffdc10, size=6) at task4.c:10
(gdb) s
```

Видим что в функции *bubble_sort* допущена ошибка, *size - i*. Эта запись означает что когда *i* не равно нулю, внутренний цикл не будет доходить до конца массива. Заменяем *i* на 1, компилируем и запускаем.

```
$ gcc -g -O0 task4.c
$ ./a.out
4 5 7 10 15 21
```

1) Изменили условие выхода из внутреннего цикла *for* на *j < size - 1*.

Вывод

В ходе выполнения данной лабораторной работы были изучены базовые принципы работы отладчика GDB, основные команды для работы с ним и произведена отладка четырех программ.