Санкт-Петербургский политехнический университет Петра Великого
Кафедра компьютерных систем и программных технологий

Отчёт по лабораторной работе №4

Дисциплина: Основы вычислительной техники

Тема: Раздельная компиляция

Выполнил студент гр. 3530901/10005 _____ Бикир И. И.
(подпись)

Преподаватель _____ Коренев Д.А.
(подпись)

“___”_____ 2022 г.

Санкт-Петербург
2022

1. Формулировка задачи

1) На языке С разработать функцию, реализующую определенную вариантом задания функциональность. Поместить определение функции в отдельный исходный файл, оформить заголовочный файл. Разработать тестовую программу на языке С.

2) Собрать программу «по шагам». Проанализировать выход препроцессора и компилятора. Проанализировать состав и содержимое секций, таблицы символов, таблицы перемещений и отладочную информацию, содержащуюся в объектных файлах исполняемом файле.

3) Выделить разработанную функцию в статическую библиотеку. Разработать make-файлы ля сборки библиотеки и использующей ее тестовой программы. Проанализировать ход сборки библиотеки и программы, созданные файлы зависимостей.

2. Вариант задания

В массиве чисел сделать перестановку – вначале все числа с нечетными индексами, потом – все с четными.

3. Ход решения

3.1 Текст программ, реализующих определенную вариантом задания функциональность

```c
#include <stdio.h>

int *changeArray(int array[10]) {
    static int myArray[10];
    for (int i = 0; i < 10; i++) {
        myArray[i] = array[i];
    }
    size_t size = sizeof(myArray) / sizeof(myArray[0]);
    int k = 0;
    int temp;

    for (int i = 1; i < size; i = i + 2) {
        for (int j = i - 1; j >= k; j--) {
            temp = myArray[j];
            myArray[j] = myArray[j + 1];
            myArray[j + 1] = temp;
        }
        k++;
    }


    for (int i = 0; i < size; i++) {
        printf("%d ", myArray[i]);
    }

    return myArray;
}

int test(int expected[10], int real[10]) {
    for (int i = 0; i < 10; i++) {
        if (expected[i] != real[i]) {
            return 0;
        }
    }
    return 1;
}

int main()
{
    int *p;

    int array[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    p = changeArray(array);
    printf("\n0 = false, 1 = true");
    int a[10] = {1, 3, 5, 7, 9, 0, 2, 4, 6, 8};
    printf("\n%d", test(a, p));
    printf("\n%d", test(array, p));
}
```

Алгоритм работает путем сдвига каждого элемента с нечетным индексом перед всеми элементами с четными индексами. Таким образом все нечетные оказываются в начале, а четные индексы – в конце.

В файле «main.c» реализована функция changeArray() в которой и находится сам алгоритм преобразования массива чисел. Также реализована функция test(), которая сравнивает два числовых массива и выводит 1, если все сходится, или 0, если есть какое-то различие.

3.3 Сборка программы

Для сборки программы выполним следующую команду:

```
PS C:\Microsoft VS Code\var14> riscv64-unknown-elf-gcc-8.3.0.exe .\main.c --save-temps -march=rv32i -mabi=ilp32 -O1 -v > log 2>&1
```

Программа *riscv64-unknown-elf-gcc* является драйвером компилятора gcc, в данном случае она запускается со следующими параметрами командной строки:

--save-temps – сохранять промежуточные файлы, создаваемые в процессе сборки;

-march=rv32i -mabi=ilp32 – целевым является процессор с базовой архитектурой системы команд RV32I;

-O1 – выполнять простые оптимизации генерируемого кода (мы используем эту опцию в примерах, потому что обычно генерируемый код получается более простым);

-v – печатать (в стандартный поток ошибок) выполняемые драйвером команды, а также дополнительную информацию.

В конце команды используется т.н. «перенаправление вывода»:

>log - вместо печати в консоли (обычно, это означает «на экране») вывод программы направляется в файл с именем “log” (если файл не существует, он создается; если файл существует, его содержимое будет утеряно);

2>&1 – поток вывода ошибок (2 – стандартный «номер» этого потока) «связывается» с поток вывода («номер» 1), т.е. сообщения об ошибках (и информация, вывод которой вызван использованием флага “-v”, см.выше) также выводятся в файл “log”.

В результате исполнения приведенной команды в текущем каталоге будут созданы следующие файлы:

a.out – исполняемый файл, сгенерированный компоновщиком в результате сборки.

log – текстовый файл, содержащий сообщения компилятора, ассемблера и компоновщика, а также выполняемые команды и дополнительную информацию;

3.4 Сборка программы по шагам

Начнем сборку созданных программ на языке Спо шагам. Первым шагом является препроцессирование файлf исходного текста “main.c” в файл “main.i”.

```
PS C:\Microsoft VS Code\var14> riscv64-unknown-elf-gcc-8.3.0.exe -march=rv32i -mabi=ilp32 -O1 -E main.c -o main.i
```

Драйвер компилятора gcc– riscv64-unknown-elf-gcc– запускается с параметрами командной строки “-march=rv32i -mabi=ilp32”, указывающих что целевым является процессор с базовой архитектурой системы команд RV32I;-O1 – указание выполнять простые оптимизации генерируемого кода; -E – указание остановить процесс сборки после препроцессирования.

В начале файла main.i содержится порядка 1200 строк с инструкциями по линковке stdio.h к проекту, а затем следует код на С, который мало отличаются от исходных версий программ:

```
# 797 "c:\\users\\aribi\\desktop\\учёба\\вуз\\2 курс\\lowprog\\cle\\riscv64-unknown-elf-gcc-8.3.0-2020.04.1-x86_64-w64-mingw32\\riscv64-unknown-elf\\include\\stdio.h" 3

# 2 "main.c" 2


# 3 "main.c"
int *changeArray(int array[10]) {
    static int myArray[10];
    for (int i = 0; i < 10; i++) {
        myArray[i] = array[i];
    }
    size_t size = sizeof(myArray) / sizeof(myArray[0]);
    int k = 0;
    int temp;

    for (int i = 1; i < size; i = i + 2) {
        for (int j = i - 1; j >= k; j--) {
            temp = myArray[j];
            myArray[j] = myArray[j + 1];
            myArray[j + 1] = temp;
        }
        k++;
    }


    for (int i = 0; i < size; i++) {
        printf("%d ", myArray[i]);
    }

    return myArray;
}

int test(int expected[10], int real[10]) {
    for (int i = 0; i < 10; i++) {
        if (expected[i] != real[i]) {
            return 0;
        }
    }
    return 1;
}

int main()
{
    int *p;

    int array[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    p = changeArray(array);
    printf("\n0 = false, 1 = true");
    int a[10] = {1, 3, 5, 7, 9, 0, 2, 4, 6, 8};
    printf("\n%d", test(a, p));
    printf("\n%d", test(array, p));
}
```

Появившиеся нестандартные директивы, начинающиеся с символа “#”, используются для передачи информации об исходном тексте из препроцессора в компилятор.

Следующим шагом является компиляция файла “main.i” в код на языке ассемблера “main.s”:

```
PS C:\Microsoft VS Code\var14> riscv64-unknown-elf-gcc-8.3.0.exe -march=rv32i -mabi=ilp32 -O1 -S main.i -o main.s
```

Драйвер компиляторariscv64-unknown-elf-gcc запускается с параметрами командной строки “-march=rv32i -mabi=ilp32”, указывающих что целевым является процессор с базовой архитектурой системы команд RV32I;-O1 – указание выполнять простые оптимизации генерируемого кода; -S – указание остановить процесс сборки после компиляции (без запуска ассемблера).

```
.file   "main.c"
.option nopic
.attribute arch, "rv32i2p0"
.attribute unaligned_access, 0
.attribute stack_align, 16
```

```
        .text
        .align  2
        .globl  changeArray
        .type   changeArray, @function
changeArray:
        addi    sp,sp,-16
        sw      ra,12(sp)
        sw      s0,8(sp)
        sw      s1,4(sp)
        sw      s2,0(sp)
        lui     a5,%hi(.LANCHOR0)
        addi    s0,a5,%lo(.LANCHOR0)
        addi    s1,s0,40
        addi    a5,a5,%lo(.LANCHOR0)
.L2:
        lw      a4,0(a0)
        sw      a4,0(a5)
        addi    a0,a0,4
        addi    a5,a5,4
        bne     a5,s1,.L2
        lui     a0,%hi(.LANCHOR0+4)
        addi    a0,a0,%lo(.LANCHOR0+4)
        mv      a2,s0
        li      a1,0
        li      a6,0
        li      a7,10
.L5:
        bgt     a6,a1,.L3
        mv      a5,a0
.L4:
        lw      a4,-4(a5)
        lw      a3,0(a5)
        sw      a3,-4(a5)
        sw      a4,0(a5)
        addi    a5,a5,-4
        bne     a5,a2,.L4
.L3:
        addi    a6,a6,1
        addi    a1,a1,2
        addi    a0,a0,8
        addi    a2,a2,4
        bne     a1,a7,.L5
        lui     s2,%hi(.LC2)
.L6:
        lw      a1,0(s0)
        addi    a0,s2,%lo(.LC2)
        call    printf
        addi    s0,s0,4
        bne     s0,s1,.L6
        lui     a0,%hi(.LANCHOR0)
        addi    a0,a0,%lo(.LANCHOR0)
        lw      ra,12(sp)
        lw      s0,8(sp)
        lw      s1,4(sp)
        lw      s2,0(sp)
        addi    sp,sp,16
        jr      ra
        .size   changeArray, .-changeArray
        .align  2
        .globl  test
        .type   test, @function
test:
        lw      a4,0(a0)
        lw      a5,0(a1)
```

```
        bne     a4,a5,.L15
        addi    a5,a0,4
        addi    a1,a1,4
        addi    a0,a0,40
.L14:
        lw      a3,0(a5)
        lw      a4,0(a1)
        bne     a3,a4,.L16
        addi    a5,a5,4
        addi    a1,a1,4
        bne     a5,a0,.L14
        li      a0,1
        ret
.L15:
        li      a0,0
        ret
.L16:
        li      a0,0
        ret
        .size   test, .-test
        .align  2
        .globl  main
        .type   main, @function
main:
        addi    sp,sp,-96
        sw      ra,92(sp)
        sw      s0,88(sp)
        sw      s1,84(sp)
        lui     s0,%hi(.LANCHOR1)
        addi    s0,s0,%lo(.LANCHOR1)
        lw      t3,0(s0)
        lw      t1,4(s0)
        lw      a7,8(s0)
        lw      a6,12(s0)
        lw      a0,16(s0)
        lw      a1,20(s0)
        lw      a2,24(s0)
        lw      a3,28(s0)
        lw      a4,32(s0)
        lw      a5,36(s0)
        sw      t3,40(sp)
        sw      t1,44(sp)
        sw      a7,48(sp)
        sw      a6,52(sp)
        sw      a0,56(sp)
        sw      a1,60(sp)
        sw      a2,64(sp)
        sw      a3,68(sp)
        sw      a4,72(sp)
        sw      a5,76(sp)
        addi    a0,sp,40
        call    changeArray
        mv      s1,a0
        lui     a0,%hi(.LC3)
        addi    a0,a0,%lo(.LC3)
        call    printf
        lw      t3,40(s0)
        lw      t1,44(s0)
        lw      a7,48(s0)
        lw      a6,52(s0)
        lw      a0,56(s0)
        lw      a1,60(s0)
        lw      a2,64(s0)
        lw      a3,68(s0)
```

```
        lw      a4,72(s0)
        lw      a5,76(s0)
        sw      t3,0(sp)
        sw      t1,4(sp)
        sw      a7,8(sp)
        sw      a6,12(sp)
        sw      a0,16(sp)
        sw      a1,20(sp)
        sw      a2,24(sp)
        sw      a3,28(sp)
        sw      a4,32(sp)
        sw      a5,36(sp)
        mv      a1,s1
        mv      a0,sp
        call    test
        mv      a1,a0
        lui     s0,%hi(.LC4)
        addi    a0,s0,%lo(.LC4)
        call    printf
        mv      a1,s1
        addi    a0,sp,40
        call    test
        mv      a1,a0
        addi    a0,s0,%lo(.LC4)
        call    printf
        li      a0,0
        lw      ra,92(sp)
        lw      s0,88(sp)
        lw      s1,84(sp)
        addi    sp,sp,96
        jr      ra
        .size   main, .-main
        .section .rodata
        .align  2
        .set    .LANCHOR1,. + 0
.LC0:
        .word   0
        .word   1
        .word   2
        .word   3
        .word   4
        .word   5
        .word   6
        .word   7
        .word   8
        .word   9
.LC1:
        .word   1
        .word   3
        .word   5
        .word   7
        .word   9
        .word   0
        .word   2
        .word   4
        .word   6
        .word   8
        .bss
        .align  2
        .set    .LANCHOR0,. + 0
        .type   myArray.2550, @object
        .size   myArray.2550, 40
myArray.2550:
        .zero   40
```

```
        .section .rodata.str1.4,"aMS",@progbits,1
        .align  2
.LC2:
        .string "%d "
.LC3:
        .string "\n0 = false, 1 = true"
        .zero   3
.LC4:
        .string "\n%d"
                .ident  "GCC: (SiFive GCC 8.3.0-2020.04.1) 8.3.0"
```

Следующим шагом является ассемблирование файла “main.s”в объектный файл “main.o”:

```
PS C:\Microsoft VS Code\var14> riscv64-unknown-elf-gcc-8.3.0.exe -march=rv32i -mabi=ilp32 -c main.s -o main.o
```

Драйвер компиляторariscv64-unknown-elf-gcc запускается с параметрами командной строки “-march=rv32i -mabi=ilp32”, указывающих что целевым является процессор с базовой архитектурой системы команд RV32I; -c – указание остановить процесс сборки после ассемблирования.

Объектный файл не является текстовым и не может быть напрямую выведен на экран в читаемом формате, для изучения его содержимого используем утилиту objdump:

```
PS C:\Microsoft VS Code\var14> riscv64-unknown-elf-objdump.exe -march=rv32i -mabi=ilp32 -f main.o

main.o:     file format elf32-littleriscv
architecture: riscv:rv32, flags 0x00000011:
HAS_RELOC, HAS_SYMS
start address 0x00000000
```

Файл имеет формат ELF, является объектным файлом 32-разрядной архитектуры RISC-V, содержит символы (флаг HAS_SYMS), содержит таблицу перемещений (флаг HAS_RELOC).

Компоновка

Компоновка программы выполняется по следующей команде:

```
PS C:\Microsoft VS Code\var14> riscv64-unknown-elf-gcc-8.3.0.exe -march=rv32i -mabi=ilp32 -O1 -v main.o -o a.out >log 2>&1
```

Результатом является исполняемый файл *“a.out”*

Объектный файл

Как известно, содержательная часть объектного файла разбита на «разделы», называемые обычно секциями. Следующая команда обеспечивает отображение заголовков секций файла ”main.o”:

```
PS C:\Microsoft VS Code\var14> riscv64-unknown-elf-objdump.exe -h main.o

main.o:     file format elf32-littleriscv

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000023c  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  00000270  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000028  00000000  00000000  00000270  2**2
                  ALLOC
  3 .rodata       00000050  00000000  00000000  00000270  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .rodata.str1.4 00000020  00000000  00000000  000002c0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 .comment      00000029  00000000  00000000  000002e0  2**0
                  CONTENTS, READONLY
  6 .riscv.attributes 0000001c  00000000  00000000  00000309  2**0
                  CONTENTS, READONLY
```

В файле “main.o” имеются следующие секции:

.*text* – секция кода, в которой содержатся коды инструкций (название секции обусловлено историческими причинами);

.*data* – секция инициализированных данных;

.*bss* – секция неинициализированных статических переменных (название секции также обусловлено историческими причинами);

.*rodata* – аналог .data для неизменяемых данных

.*comment* – секция данных о версиях размером 12 байт

.riscv.attributes – информация про RISC-V

Изучим таблицу символов файла:

```
PS C:\Microsoft VS Code\var14> riscv64-unknown-elf-objdump.exe -t main.o

main.o:     file format elf32-littleriscv

SYMBOL TABLE:
00000000 l    df *ABS*  00000000 main.c
00000000 l    d  .text  00000000 .text
00000000 l    d  .data  00000000 .data
00000000 l    d  .bss   00000000 .bss
00000000 l    d  .rodata        00000000 .rodata
00000000 l       .rodata        00000000 .LANCHOR1
00000000 l       .bss   00000000 .LANCHOR0
00000000 l     O .bss   00000028 myArray.2550
00000000 l    d  .rodata.str1.4 00000000 .rodata.str1.4
00000000 l       .rodata.str1.4 00000000 .LC2
00000004 l       .rodata.str1.4 00000000 .LC3
0000001c l       .rodata.str1.4 00000000 .LC4
00000024 l       .text  00000000 .L2
00000070 l       .text  00000000 .L3
00000058 l       .text  00000000 .L4
00000050 l       .text  00000000 .L5
00000088 l       .text  00000000 .L6
000000f8 l       .text  00000000 .L15
00000100 l       .text  00000000 .L16
000000d8 l       .text  00000000 .L14
00000000 l    d  .comment       00000000 .comment
00000000 l    d  .riscv.attributes      00000000 .riscv.attributes
00000000 g     F .text  000000c0 changeArray
00000000         *UND*  00000000 printf
000000c0 g     F .text  00000048 test
00000108 g     F .text  00000134 main
```

Таблица содержит 3 глобальные (флаг g) функции (флаг F) – main, changeArray, test, а также один неопределённый символ (UND).

UND означает, что символ printf использовался в ассемблерном коде, из которого был получен данный объектный фал, но не был определён;

Ассемблер сделал вывод о том, что символ должен быть определён где-то ещё, и отразил это в таблице символов.

Секция .text

Изучим содержимое секции “.text”:

```
PS C:\Microsoft VS Code\var14> riscv64-unknown-elf-objdump.exe -s -j .text main.o

main.o:     file format elf32-littleriscv

Contents of section .text:
 0000 130101ff 23261100 23248100 23229100  ....#&..#$..#"..
 0010 23202101 b7070000 13840700 93048402  # !.............
 0020 93870700 03270500 23a0e700 13054500  .....'..#.....E.
 0030 93874700 e39897fe 37050000 13054500  ..G.....7.....E.
 0040 13060400 93050000 13080000 9308a000  ................
 0050 63c00503 93070500 03a7c7ff 83a60700  c...............
 0060 23aed7fe 23a0e700 9387c7ff e396c7fe  #...#...........
 0070 13081800 93852500 13058500 13064600  ......%.......F.
 0080 e39815fd 37090000 83250400 13050900  ....7....%......
 0090 97000000 e7800000 13044400 e31694fe  ..........D.....
 00a0 37050000 13050500 8320c100 03248100  7........ ...$..
 00b0 83244100 03290100 13010101 67800000  .$A..).......g...
 00c0 03270500 83a70500 6318f702 93074500  .'......c.....E.
 00d0 93854500 13058502 83a60700 03a70500  ..E.............
 00e0 6390e602 93874700 93854500 e396a7fe  c.....G...E.....
 00f0 13051000 67800000 13050000 67800000  ....g.......g...
 0100 13050000 67800000 130101fa 232e1104  ....g.......#...
 0110 232c8104 232a9104 37040000 13040400  #,..#*..7.......
 0120 032e0400 03234400 83288400 0328c400  .....#D..(...(..
 0130 03250401 83254401 03268401 8326c401  .%...%D..&...&..
 0140 03270402 83274402 2324c103 23266102  .'...'D.#$..#&a.
 0150 23281103 232a0103 232ca102 232eb102  #(..#*..#,..#...
 0160 2320c104 2322d104 2324e104 2326f104  # ..#"..#$..#&..
 0170 13058102 97000000 e7800000 93040500  ................
 0180 37050000 13050500 97000000 e7800000  7...............
 0190 032e8402 0323c402 83280403 03284403  .....#...(...(D.
 01a0 03258403 8325c403 03260404 83264404  .%...%...&...&D.
 01b0 03278404 8327c404 2320c101 23226100  .'...'..# ..#"a.
 01c0 23241101 23260101 2328a100 232ab100  #$..#&..#(..#*..
 01d0 232cc100 232ed100 2320e102 2322f102  #,..#...# ..#"..
 01e0 93850400 13050100 97000000 e7800000  ................
 01f0 93050500 37040000 13050400 97000000  ....7...........
 0200 e7800000 93850400 13058102 97000000  ................
 0210 e7800000 93050500 13050400 97000000  ................
 0220 e7800000 13050000 8320c105 03248105  ......... ...$..
 0230 83244105 13010106 67800000           .$A.....g...
```

Разумеется, процедура декодирования кодов инструкций является «механической» (иначе как бы ее реализовывало техническое устройство – процессор), следовательно, разумно поручить ее выполнение ЭВМ:

```
PS C:\Microsoft VS Code\var14> riscv64-unknown-elf-objdump.exe -d -M no-alliases -j .text main.o

main.o:     file format elf32-littleriscv


Disassembly of section .text:

00000000 <changeArray>:
   0:   C:\Users\aribi\Desktop\Учёба\ВУЗ\2 курс\lowprog\cle\riscv64-unknown-elf-gcc-8.3.0-2020.04.1-x86_64-w64-mingw32\bin\riscv64-unknown-elf-objdump.exe: unrecog
nized disassembler option: no-alliases
ff010113                addi    sp,sp,-16
   4:   00112623                sw      ra,12(sp)
   8:   00812423                sw      s0,8(sp)
   c:   00912223                sw      s1,4(sp)
  10:   01212023                sw      s2,0(sp)
  14:   000007b7                lui     a5,0x0
  18:   00078413                mv      s0,a5
  1c:   02840493                addi    s1,s0,40
  20:   00078793                mv      a5,a5

00000024 <.L2>:
  24:   00052703                lw      a4,0(a0)
  28:   00e7a023                sw      a4,0(a5) # 0 <changeArray>
  2c:   00450513                addi    a0,a0,4
  30:   00478793                addi    a5,a5,4
  34:   fe9798e3                bne     a5,s1,24 <.L2>
  38:   00000537                lui     a0,0x0
  3c:   00450513                addi    a0,a0,4 # 4 <changeArray+0x4>
  40:   00040613                mv      a2,s0
  44:   00000593                li      a1,0
  48:   00000813                li      a6,0
  4c:   00a00893                li      a7,10

00000050 <.L5>:
  50:   0305c063                blt     a1,a6,70 <.L3>
  54:   00050793                mv      a5,a0

00000058 <.L4>:
  58:   ffc7a703                lw      a4,-4(a5)
  5c:   0007a683                lw      a3,0(a5)
  60:   fed7ae23                sw      a3,-4(a5)
  64:   00e7a023                sw      a4,0(a5)
  68:   ffc78793                addi    a5,a5,-4
  6c:   fec796e3                bne     a5,a2,58 <.L4>
```

```
00000070 <.L3>:
  70:   00180813                addi    a6,a6,1
  74:   00258593                addi    a1,a1,2
  78:   00850513                addi    a0,a0,8
  7c:   00460613                addi    a2,a2,4
  80:   fd1598e3                bne     a1,a7,50 <.L5>
  84:   00000937                lui     s2,0x0

00000088 <.L6>:
  88:   00042583                lw      a1,0(s0)
  8c:   00090513                mv      a0,s2
  90:   00000097                auipc   ra,0x0
  94:   000080e7                jalr    ra # 90 <.L6+0x8>
  98:   00440413                addi    s0,s0,4
  9c:   fe9416e3                bne     s0,s1,88 <.L6>
  a0:   00000537                lui     a0,0x0
  a4:   00050513                mv      a0,a0
  a8:   00c12083                lw      ra,12(sp)
  ac:   00812403                lw      s0,8(sp)
  b0:   00412483                lw      s1,4(sp)
  b4:   00012903                lw      s2,0(sp)
  b8:   01010113                addi    sp,sp,16
  bc:   00008067                ret

000000c0 <test>:
  c0:   00052703                lw      a4,0(a0) # 0 <changeArray>
  c4:   0005a783                lw      a5,0(a1)
  c8:   02f71863                bne     a4,a5,f8 <.L15>
  cc:   00450793                addi    a5,a0,4
  d0:   00458593                addi    a1,a1,4
  d4:   02850513                addi    a0,a0,40

000000d8 <.L14>:
  d8:   0007a683                lw      a3,0(a5)
  dc:   0005a703                lw      a4,0(a1)
  e0:   02e69063                bne     a3,a4,100 <.L16>
  e4:   00478793                addi    a5,a5,4
  e8:   00458593                addi    a1,a1,4
  ec:   fea796e3                bne     a5,a0,d8 <.L14>
  f0:   00100513                li      a0,1
  f4:   00008067                ret

000000f8 <.L15>:
  f8:   00000513                li      a0,0
  fc:   00008067                ret

00000100 <.L16>:
 100:   00000513                li      a0,0
 104:   00008067                ret

00000108 <main>:
 108:   fa010113                addi    sp,sp,-96
 10c:   04112e23                sw      ra,92(sp)
```

```
00000108 <main>:
 108:   fa010113                addi    sp,sp,-96
 10c:   04112e23                sw      ra,92(sp)
 110:   04812c23                sw      s0,88(sp)
 114:   04912a23                sw      s1,84(sp)
 118:   00000437                lui     s0,0x0
 11c:   00040413                mv      s0,s0
 120:   00042e03                lw      t3,0(s0) # 0 <changeAr
 124:   00442303                lw      t1,4(s0)
 128:   00842883                lw      a7,8(s0)
 12c:   00c42803                lw      a6,12(s0)
 130:   01042503                lw      a0,16(s0)
 134:   01442583                lw      a1,20(s0)
 138:   01842603                lw      a2,24(s0)
 13c:   01c42683                lw      a3,28(s0)
 140:   02042703                lw      a4,32(s0)
 144:   02442783                lw      a5,36(s0)
 148:   03c12423                sw      t3,40(sp)
 14c:   02612623                sw      t1,44(sp)
 150:   03112823                sw      a7,48(sp)
 154:   03012a23                sw      a6,52(sp)
 158:   02a12c23                sw      a0,56(sp)
 15c:   02b12e23                sw      a1,60(sp)
 160:   04c12023                sw      a2,64(sp)
 164:   04d12223                sw      a3,68(sp)
 168:   04e12423                sw      a4,72(sp)
 16c:   04f12623                sw      a5,76(sp)
 170:   02810513                addi    a0,sp,40
 174:   00000097                auipc   ra,0x0
 178:   000080e7                jalr    ra # 174 <main+0x6c>
 17c:   00050493                mv      s1,a0
 180:   00000537                lui     a0,0x0
 184:   00050513                mv      a0,a0
 188:   00000097                auipc   ra,0x0
 18c:   000080e7                jalr    ra # 188 <main+0x80>
 190:   02842e03                lw      t3,40(s0)
 194:   02c42303                lw      t1,44(s0)
 198:   03042883                lw      a7,48(s0)
 19c:   03442803                lw      a6,52(s0)
 1a0:   03842503                lw      a0,56(s0)
 1a4:   03c42583                lw      a1,60(s0)
 1a8:   04042603                lw      a2,64(s0)
 1ac:   04442683                lw      a3,68(s0)
 1b0:   04842703                lw      a4,72(s0)
 1b4:   04c42783                lw      a5,76(s0)
 1b8:   01c12023                sw      t3,0(sp)
 1bc:   00612223                sw      t1,4(sp)
 1c0:   01112423                sw      a7,8(sp)
 1c4:   01012623                sw      a6,12(sp)
 1c8:   00a12823                sw      a0,16(sp)
```

Опция *“-d”* инициирует процесс дизассемблирования, опция *“-M no-aliases”* требует использовать в выводе только инструкции системы команд (но не псевдоинструкции ассемблера). Секция кода теперь содержит намного большее количество строк, поэтому рассмотрим только самые важные участки:

Секция .comment

```
PS C:\Microsoft VS Code\var14> riscv64-unknown-elf-objdump.exe -s -j .comment main.o

main.o:     file format elf32-littleriscv

Contents of section .comment:
 0000 00474343 3a202853 69466976 65204743  .GCC: (SiFive GC
 0010 4320382e 332e302d 32303230 2e30342e  C 8.3.0-2020.04.
 0020 31292038 2e332e30 00                 1) 8.3.0.
```

В секции .comment, записаны инициалы компилятора, которым проводилась процедура.

Вывод компоновщика – исполняемый файл

Сформированный компоновщиком файл *“a.out”*, разумеется, также является «бинарным», и для изучения его содержимого будем пользоваться утилитой *objdump*:

```
PS C:\Microsoft VS Code\var14> riscv64-unknown-elf-objdump.exe -f a.out

a.out:     file format elf32-littleriscv
architecture: riscv:rv32, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x00010090
```

Изучим секции файла:

```
PS C:\Microsoft VS Code\var14> riscv64-unknown-elf-objdump.exe -h a.out

a.out:     file format elf32-littleriscv

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00014a00  00010074  00010074  00000074  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .rodata       00000d54  00024a78  00024a78  00014a78  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .eh_frame     000000b4  00026000  00026000  00016000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  3 .init_array   00000008  000260b4  000260b4  000160b4  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  4 .fini_array   00000004  000260bc  000260bc  000160bc  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  5 .data         0000099c  000260c0  000260c0  000160c0  2**3
                  CONTENTS, ALLOC, LOAD, DATA
  6 .sdata        0000002c  00026a60  00026a60  00016a60  2**3
                  CONTENTS, ALLOC, LOAD, DATA
  7 .sbss         00000014  00026a8c  00026a8c  00016a8c  2**2
                  ALLOC
  8 .bss          00000070  00026aa0  00026aa0  00016a8c  2**2
                  ALLOC
  9 .comment      00000028  00000000  00000000  00016a8c  2**0
                  CONTENTS, READONLY
 10 .riscv.attributes 0000001c  00000000  00000000  00016ab4  2**0
                  CONTENTS, READONLY
 11 .debug_aranges 00000218  00000000  00000000  00016ad0  2**3
                  CONTENTS, READONLY, DEBUGGING
 12 .debug_info   0000924d  00000000  00000000  00016ce8  2**0
                  CONTENTS, READONLY, DEBUGGING
 13 .debug_abbrev 00001c52  00000000  00000000  0001ff35  2**0
                  CONTENTS, READONLY, DEBUGGING
 14 .debug_line   0000a0d0  00000000  00000000  00021b87  2**0
                  CONTENTS, READONLY, DEBUGGING
 15 .debug_frame  000002dc  00000000  00000000  0002bc58  2**2
                  CONTENTS, READONLY, DEBUGGING
 16 .debug_str    00001382  00000000  00000000  0002bf34  2**0
                  CONTENTS, READONLY, DEBUGGING
 17 .debug_loc    000089e7  00000000  00000000  0002d2b6  2**0
                  CONTENTS, READONLY, DEBUGGING
 18 .debug_ranges 000012b0  00000000  00000000  00035c9d  2**0
                  CONTENTS, READONLY, DEBUGGING
```

Состав секций *“a.out”* значительно расширен по сравнению с *“main.o”*. Также увеличились размеры секций *“.text”*, *“.data”*, *“.bss”* и *“.comment”*. Дополнительные секции появились из других объектных файлов, переданных на вход компоновщика.

Инструкции программы

Изучим содержимое секции “.text”:

```
PS C:\Microsoft VS Code\var14> riscv64-unknown-elf-objdump.exe -j .text -d -M no-aliases a.out >a.ds
```

В результате выполнения получили файл *“a.ds”*. Изучим его.

```
00010090 <_start>:
   10090:       00017197                auipc   gp,0x17
   10094:       83018193                addi    gp,gp,-2000 # 268c0 <__global_pointer$>
   10098:       1cc18513                addi    a0,gp,460 # 26a8c <_edata>
   1009c:       25018613                addi    a2,gp,592 # 26b10 <__BSS_END__>
   100a0:       40a60633                sub     a2,a2,a0
   100a4:       00000593                addi    a1,zero,0
   100a8:       3ec000ef                jal     ra,10494 <memset>
   100ac:       00000517                auipc   a0,0x0
   100b0:       2f450513                addi    a0,a0,756 # 103a0 <__libc_fini_array>
   100b4:       2a4000ef                jal     ra,10358 <atexit>
   100b8:       348000ef                jal     ra,10400 <__libc_init_array>
   100bc:       00012503                lw      a0,0(sp)
   100c0:       00410593                addi    a1,sp,4
   100c4:       00000613                addi    a2,zero,0
   100c8:       174000ef                jal     ra,1023c <main>
   100cc:       2a00006f                jal     zero,1036c <exit>
```

“_start” – “точка входа” в нашу программу. Код, начинающийся с метки “_start” обеспечивает инициализацию памяти, регистров процессора и среды времени выполнения, после чего передаёт управление определённой нами функции main.

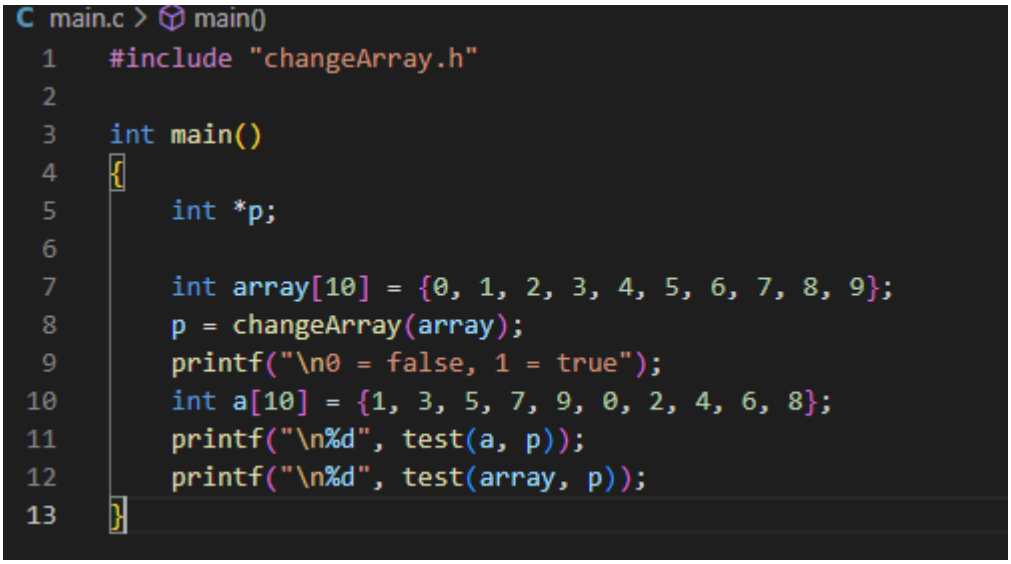
```
0001036c <exit>:
   1036c:       ff010113                addi    sp,sp,-16
   10370:       00000593                addi    a1,zero,0
   10374:       00812423                sw      s0,8(sp)
   10378:       00112623                sw      ra,12(sp)
   1037c:       00050413                addi    s0,a0,0
   10380:       050030ef                jal     ra,133d0 <__call_exitprocs>
   10384:       1b818793                addi    a5,gp,440 # 26a78 <_global_impure_ptr>
   10388:       0007a503                lw      a0,0(a5)
   1038c:       03c52783                lw      a5,60(a0)
   10390:       00078463                beq     a5,zero,10398 <exit+0x2c>
   10394:       000780e7                jalr    ra,0(a5)
   10398:       00040513                addi    a0,s0,0
   1039c:       5080f0ef                jal     ra,1f8a4 <_exit>
```

Можно видеть, что в конце *“exit”* управление передается на символ *“_exit”*

Раздельная компиляция

Разобьём исходную программу на 3 файла.

main.c

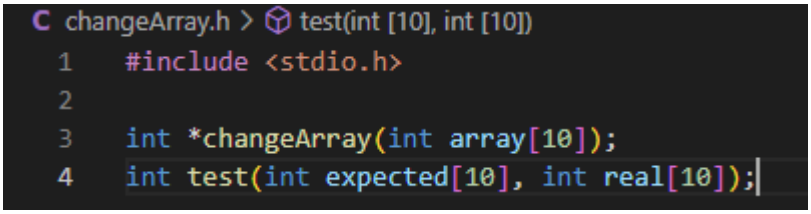


```c
#include "changeArray.h"

int main()
{
    int *p;

    int array[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    p = changeArray(array);
    printf("\n0 = false, 1 = true");
    int a[10] = {1, 3, 5, 7, 9, 0, 2, 4, 6, 8};
    printf("\n%d", test(a, p));
    printf("\n%d", test(array, p));
}
```

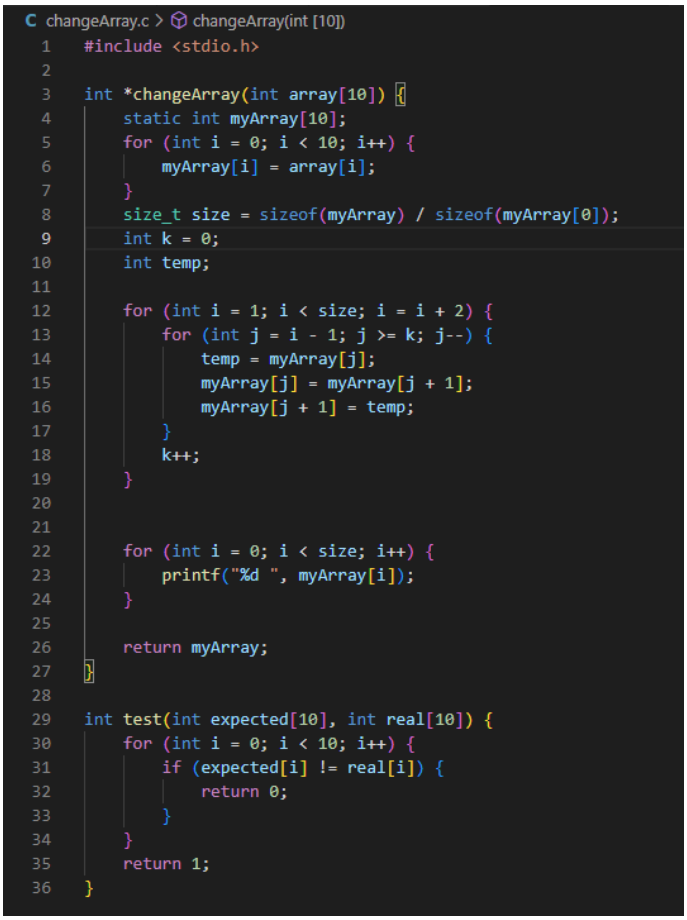
changeArray.h



```c
#include <stdio.h>

int *changeArray(int array[10]);
int test(int expected[10], int real[10]);
```

changeArray.c



```c
#include <stdio.h>

int *changeArray(int array[10]) {
    static int myArray[10];
    for (int i = 0; i < 10; i++) {
        myArray[i] = array[i];
    }
    size_t size = sizeof(myArray) / sizeof(myArray[0]);
    int k = 0;
    int temp;

    for (int i = 1; i < size; i = i + 2) {
        for (int j = i - 1; j >= k; j--) {
            temp = myArray[j];
            myArray[j] = myArray[j + 1];
            myArray[j + 1] = temp;
        }
        k++;
    }


    for (int i = 0; i < size; i++) {
        printf("%d ", myArray[i]);
    }

    return myArray;
}

int test(int expected[10], int real[10]) {
    for (int i = 0; i < 10; i++) {
        if (expected[i] != real[i]) {
            return 0;
        }
    }
    return 1;
}
```

Сборка программы

Сборка программы осуществляется следующей командой:

```
PS C:\Microsoft VS Code\var14\sc> riscv64-unknown-elf-gcc-8.3.0.exe -march=rv32i -mabi=ilp32 -O1 -v .\main.c .\changeArray.c >log 2>&1
PS C:\Microsoft VS Code\var14\sc> dir


    Каталог: C:\Microsoft VS Code\var14\sc


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        23.12.2022     22:03                .vscode
-a----        23.12.2022     22:13         234576 a.out
-a----        23.12.2022     22:07            782 changeArray.c
-a----        23.12.2022     22:07             97 changeArray.h
-a----        23.12.2022     22:13          28068 log
-a----        23.12.2022     22:05            303 main.c


PS C:\Microsoft VS Code\var14\sc>
```

Ранее препроцессирование, компиляция и ассемблирование выполнялось нами по шагам, но на практике это требуется редко, обычно необходимо выполнить все стадии обработки исходного файла, получив в результате объектный файл.

```
PS C:\Microsoft VS Code\var14\sc> riscv64-unknown-elf-gcc-8.3.0.exe -march=rv32i -mabi=ilp32 -O1 -c changeArray.c -o changeArray.o
```

Параметры:

- *“-c”* - приводит к останову процесса сборки после ассемблирования, т.е. после формирования объектного файла

После создания объектного файла его нужно скомпоновать с нашей программой

```
PS C:\Microsoft VS Code\var14\sc> riscv64-unknown-elf-gcc-8.3.0.exe -march=rv32i -mabi=ilp32 -O1 -c .\main.c .\changeArray.o
```

Создание и использование статической библиотеки

Статическая библиотека (static library) является, по сути, архивом (набором, коллекцией) объектных файлов, среди которых компоновщик выбирает «полезные» для данной программы: объектный файл считается «полезным», если в нем определяется еще не разрешенный компоновщиком символ.

Поместим changeArray.o в такой архив:

```
PS C:\Microsoft VS Code\var14\sc> riscv64-unknown-elf-ar.exe -rsc grlib.a changeArray.o
PS C:\Microsoft VS Code\var14\sc> dir


    Каталог: C:\Microsoft VS Code\var14\sc


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        23.12.2022     22:03                .vscode
-a----        23.12.2022     22:13         234576 a.out
-a----        23.12.2022     22:07            782 changeArray.c
-a----        23.12.2022     22:07             97 changeArray.h
-a----        23.12.2022     22:14           1716 changeArray.o
-a----        23.12.2022     22:16           1874 grlib.a
-a----        23.12.2022     22:13          28068 log
-a----        23.12.2022     22:05            303 main.c
-a----        23.12.2022     22:15           1748 main.o
```

Параметры:

- -r – заменить старые файлы с такими названиями (*changeArray.o*), если они уже есть в архиве
- -s – записать «index» в архив. Index – это список всех символов, объявленных во включенных в архив объектных файлах, и его присутствие ускоряет линковку
- -c – создать архив, если его еще не было

Результирующим файлом является “grlib.a” (“.a” – от “archive”).

Используем статическую библиотеку для сборки программ:

```
PS C:\Microsoft VS Code\var14\sc> riscv64-unknown-elf-gcc-8.3.0.exe -march=rv32i -mabi=ilp32 -O1 --save-temps main.c grlib.a -o static.o
PS C:\Microsoft VS Code\var14\sc> dir


    Каталог: C:\Microsoft VS Code\var14\sc


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        23.12.2022     22:03                .vscode
-a----        23.12.2022     22:13         234576 a.out
-a----        23.12.2022     22:07            782 changeArray.c
-a----        23.12.2022     22:07             97 changeArray.h
-a----        23.12.2022     22:14           1716 changeArray.o
-a----        23.12.2022     22:16           1874 grlib.a
-a----        23.12.2022     22:13          28068 log
-a----        23.12.2022     22:05            303 main.c
-a----        23.12.2022     22:17          49726 main.i
-a----        23.12.2022     22:17           1748 main.o
-a----        23.12.2022     22:17           1735 main.s
-a----        23.12.2022     22:17         234576 static.out
```

Изучим таблицы символов полученных исполняемых файлов:

```
PS C:\Microsoft VS Code\var14\sc> riscv64-unknown-elf-objdump.exe -t static.out > _symtable
PS C:\Microsoft VS Code\var14\sc> dir


    Каталог: C:\Microsoft VS Code\var14\sc


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        23.12.2022     22:03                .vscode
-a----        23.12.2022     22:13         234576 a.out
-a----        23.12.2022     22:07            782 changeArray.c
-a----        23.12.2022     22:07             97 changeArray.h
-a----        23.12.2022     22:14           1716 changeArray.o
-a----        23.12.2022     22:16           1874 grlib.a
-a----        23.12.2022     22:13          28068 log
-a----        23.12.2022     22:05            303 main.c
-a----        23.12.2022     22:17          49726 main.i
-a----        23.12.2022     22:17           1748 main.o
-a----        23.12.2022     22:17           1735 main.s
-a----        23.12.2022     22:17         234576 static.out
-a----        23.12.2022     22:18          29308 _symtable
```

Содержимое _symtable

```
00018aac g     F .text  0000005c __sread
00017808 g     F .text  00000004 __malloc_lock
00013748 g     F .text  00000060 _fflush_r
0001cf70 g     F .text  000000c8 _calloc_r
00026a8c g       .sbss  00000000 __bss_start
00010494 g     F .text  000000dc memset
00010144 g     F .text  0000011c main
00026a90 g     O .sbss  00000004 __malloc_max_total_mem
0001f730 g     F .text  00000014 __swbuf
00018c00 g     F .text  00000008 __sclose
0001d19c g     F .text  00000010 fclose
00016eec g     F .text  000007cc _malloc_r
0001cee0 g     F .text  00000030 __ascii_wctomb
00013f68 g     F .text  000000b0 _fwalk
000176b8 g     F .text  0000000c _mbtowc_r

00024974 g     F .text  00000084 .hidden __divsi3
00013b48 g     F .text  00000128 _malloc_trim_r
00018c08 g     F .text  0000017c strcmp
0001cdfc g     F .text  00000018 vfiprintf
00021830 g     F .text  0000136c .hidden __multf3
00018a30 g     F .text  0000007c sprintf
000256cc g     O .rodata        00000100 .hidden __clz_tab
00026a8c g     O .sbss  00000004 _PathLocale
00010358 g     F .text  00000014 atexit
0001cf10 g     F .text  00000060 _write_r
00016ce8 g     F .text  00000014 setlocale
00026a80 g     O .sdata 00000004 _impure_ptr
000134ec g     F .text  0000025c __sflush_r
000215a8 g     F .text  00000144 .hidden __gttf2
0001e4cc g     F .text  000010e0 _svfiprintf_r
000176c4 g     F .text  00000068 __ascii_mbtowc
00022b9c g     F .text  000015a0 .hidden __subtf3
000183e4 g     F .text  00000060 __ulp
00013b34 g     F .text  00000014 __fp_unlock_all
00016c5c g     F .text  00000008 localeconv
```

Видим, что все нужные символы вошли в исполняемый файл.

Несмотря на то, что в нашем случае компоновщик не используется, преимущества использования библиотеки очевидны: при компоновке были использованы необходимые объектные файлы и только они, причем задача выбора необходимых для сборки объектных файлов была возложена на компоновщик (а не нас).

Makefile

Makefile - это набор инструкций для программы make, которая позволяет собирать проекты, состоящие из большого числа “*.c” и “*.h” файлов. Обычно эта программа используется в связке с системами сборки, например cmake, позволяя вести проекты модульно (т.е. проект с включенными подпроектами).

Сборка с помощью Makefile:

```
C:\Microsoft VS Code\var14\sc\ma>mingw32-make.exe
gcc -c main.c
ar -rsc lib.a changeArray.o
gcc main.o lib.a -o output

C:\Microsoft VS Code\var14\sc\ma>dir
 Том в устройстве C не имеет метки.
 Серийный номер тома: 449B-8E08

 Содержимое папки C:\Microsoft VS Code\var14\sc\ma

23.12.2022  22:53    <DIR>          .
23.12.2022  22:53    <DIR>          ..
23.12.2022  22:07               782 changeArray.c
23.12.2022  22:07                97 changeArray.h
23.12.2022  22:51             1 213 changeArray.o
23.12.2022  22:53             1 374 lib.a
23.12.2022  22:05               303 main.c
23.12.2022  22:53             1 193 main.o
23.12.2022  22:52               214 makefile
23.12.2022  22:53            41 629 output.exe
               8 файлов         46 805 байт
               2 папок  13 421 334 528 байт свободно

C:\Microsoft VS Code\var14\sc\ma>
```

Демонстрация работы программы:

```
C:\Microsoft VS Code\var14\sc\ma>output.exe
1 3 5 7 9 0 2 4 6 8
0 = false, 1 = true
1
0
```

Очистка

```
C:\Microsoft VS Code\var14\sc\ma>mingw32-make.exe clean
del *.o *.a

C:\Microsoft VS Code\var14\sc\ma>dir
 Том в устройстве C не имеет метки.
 Серийный номер тома: 449B-8E08

 Содержимое папки C:\Microsoft VS Code\var14\sc\ma

23.12.2022  22:55    <DIR>          .
23.12.2022  22:55    <DIR>          ..
23.12.2022  22:07               782 changeArray.c
23.12.2022  22:07                97 changeArray.h
23.12.2022  22:05               303 main.c
23.12.2022  22:52               214 makefile
23.12.2022  22:53            41 629 output.exe
               5 файлов         43 025 байт
               2 папок  13 418 770 432 байт свободно
```

Что происходит в Makefile:

1. Создаём объектный файл *mainn.o* из исходного *mainn.c*
2. Создаём объектный файл *changeArray.o* из исходного *changeArray.c*
3. Архивируем объектный файл changeArray.o (создаём статическую библиотеку *lib.a*)
4. Компонуем статическую библиотеку *lib.a* с объектным файлом *mainn.o* и получаем исполняемый файл *output*

ВЫВОД:

В данной лабораторной работе мы познакомились с процессом сборки проекта на языке С.

Он состоит из:

1. Препроцессирования: исходного .c файл препроцессируем в .i файл
2. Компиляции: полученный .i файл компилируется в файл ассемблера .s
3. Ассемблирования: файл .s асссемблируется в объектный файл .o
4. Компоновки: объектный файл .o компонуется в исполняемый файл

Также мы ознакомились в makefile’ами, которые упрощают процесс сборки.

Утилита Make позволяет собирать проекты, состоящие из большого количества файлов, вместо использования PS/SH скриптов, и прописывания файлов вручную.