

Отчет по лабораторной работе №14

Тема:

Средства, применяемые при разработке программного обеспечения в ОС типа UNIX/Linux.

Российский Университет Дружбы Народов

Факультет Физико-Математических и Естественных Наук

Дисциплина: *Операционные системы*

Студент: Мухамедияр Адиль

Группа: НКНбд-01-20

Москва, 2021г.

Цель работы

Приобрести простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования C калькулятора с простейшими функциями.

Введение

Процесс разработки программного обеспечения обычно разделяется на следующие этапы:

- планирование, включающее сбор и анализ требований к функционалу и другим характеристикам разрабатываемого приложения;
- проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, определение языка программирования;
- непосредственная разработка приложения:
- кодирование — по сути создание исходного текста программы (возможно в нескольких вариантах);
- анализ разработанного кода;
- сборка, компиляция и разработка исполняемого модуля;
- тестирование и отладка, сохранение произведённых изменений;
- документирование. Для создания исходного текста программы разработчик может воспользоваться любым удобным для него редактором текста: vi, vim, mceditor, emacs, geany и др. После завершения написания исходного кода программы (возможно состоящей из нескольких файлов), необходимо её скомпилировать и получить исполняемый модуль.

Ход работы:

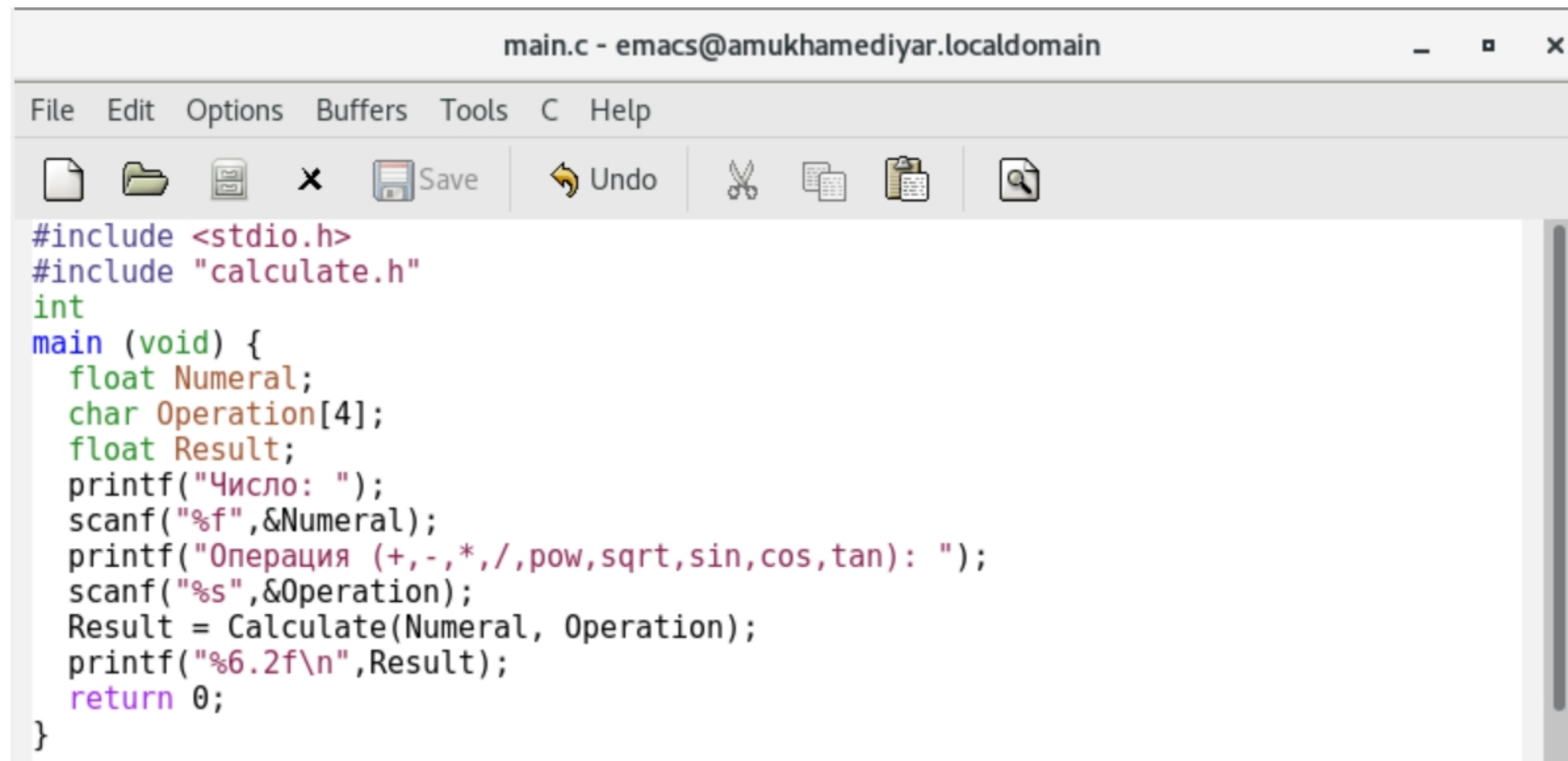
1. В домашнем каталоге создал подкаталог *~/work/os/lab_prog*.

```
[amukhamediyar@amukhamediyar ~]$ mkdir work
mkdir: невозможно создать каталог «work»: Файл существует
[amukhamediyar@amukhamediyar ~]$ cd work
[amukhamediyar@amukhamediyar work]$ mkdir os
mkdir: невозможно создать каталог «os»: Файл существует
[amukhamediyar@amukhamediyar work]$ cd os
[amukhamediyar@amukhamediyar os]$ mkdir lab_prog
[amukhamediyar@amukhamediyar os]$ cd lab_prog
[amukhamediyar@amukhamediyar lab_prog]$ █
```

2. Создал в нём файлы: calculate.h, calculate.c, main.c. Это примитивнейший калькулятор, способный складывать, вычитать, умножать, делить, возводить число в степень, вычислять квадратный корень, вычислять sin, cos, tan.

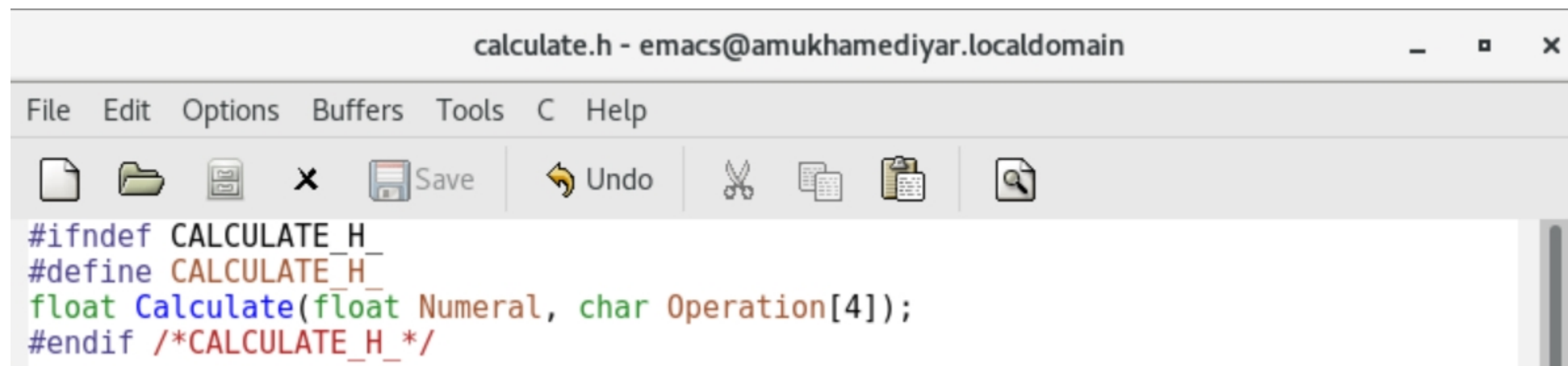
```
[amukhamediyar@amukhamediyar lab_prog]$ touch calculate.h
[amukhamediyar@amukhamediyar lab_prog]$ touch calculate.c
[amukhamediyar@amukhamediyar lab_prog]$ touch main.c
[amukhamediyar@amukhamediyar lab_prog]$ █
```

- Основной файл main.c, реализующий интерфейс пользователя к калькулятору:



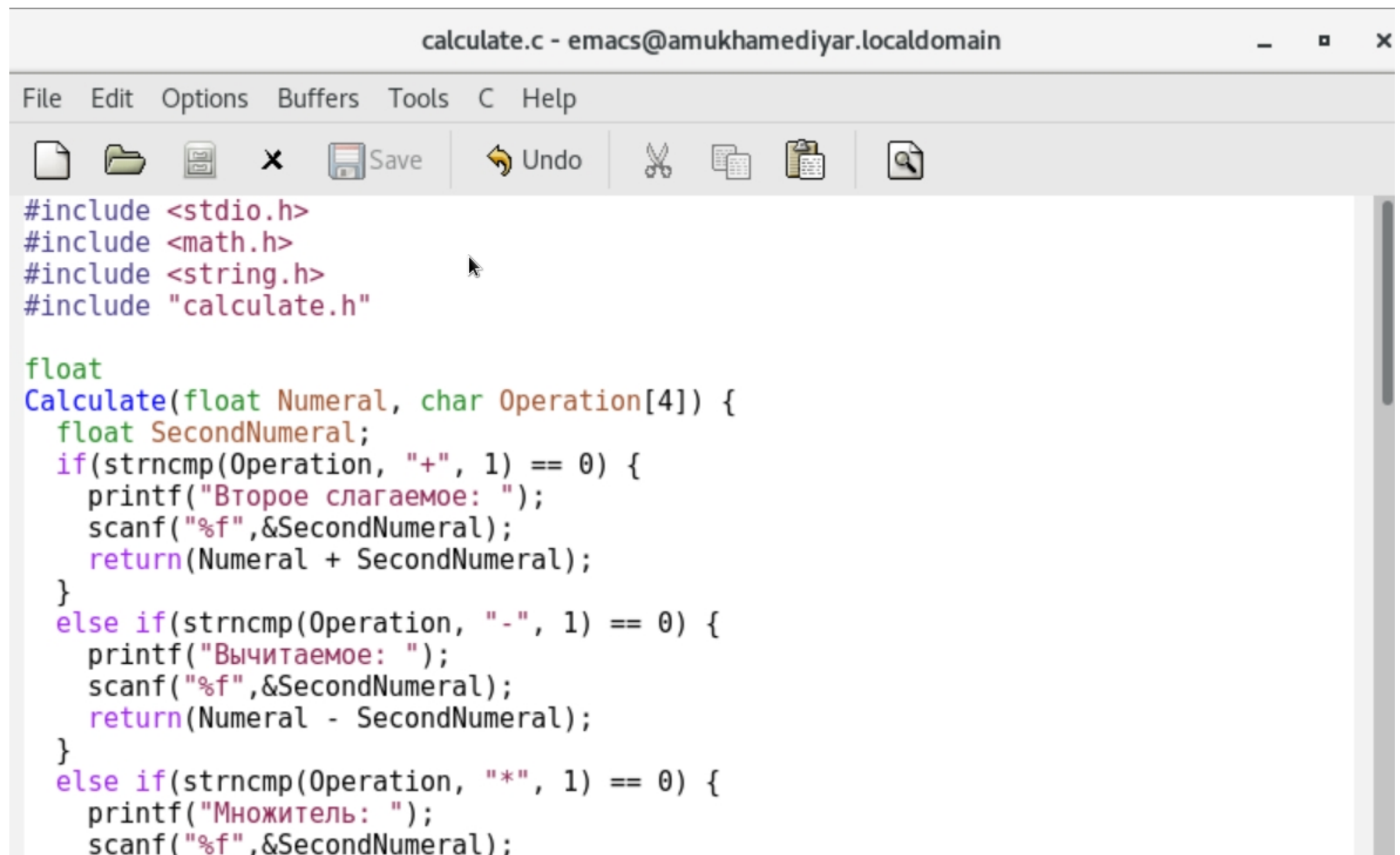
```
main.c - emacs@amukhamediyar.localdomain
File Edit Options Buffers Tools C Help
[Icons: New, Open, Save, Close, Undo, Redo, Cut, Copy, Paste, Find]
#include <stdio.h>
#include "calculate.h"
int
main (void) {
    float Numeral;
    char Operation[4];
    float Result;
    printf("Число: ");
    scanf("%f",&Numeral);
    printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
    scanf("%s",&Operation);
    Result = Calculate(Numeral, Operation);
    printf("%6.2f\n",Result);
    return 0;
}
```

- Интерфейсный файл calculate.h, описывающий формат вызова функции калькулятора:



```
#ifndef CALCULATE_H_
#define CALCULATE_H_
float Calculate(float Numeral, char Operation[4]);
#endif /*CALCULATE_H_*/
```

- Реализация функций калькулятора в файле calculate.c:



```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "calculate.h"

float
Calculate(float Numeral, char Operation[4]) {
    float SecondNumeral;
    if(strncmp(Operation, "+", 1) == 0) {
        printf("Второе слагаемое: ");
        scanf("%f",&SecondNumeral);
        return(Numeral + SecondNumeral);
    }
    else if(strncmp(Operation, "-", 1) == 0) {
        printf("Вычитаемое: ");
        scanf("%f",&SecondNumeral);
        return(Numeral - SecondNumeral);
    }
    else if(strncmp(Operation, "*", 1) == 0) {
        printf("Множитель: ");
        scanf("%f",&SecondNumeral);
```

- При запуске он запрашивает первое число, операцию, второе число. После этого программа выводит результат и останавливается.

calculate.c - emacs@amukhamediyar.localdomain



File Edit Options Buffers Tools C Help



```
scanf("%f",&SecondNumeral);
return(Numeral * SecondNumeral);
}
else if(strncmp(0peration, "/", 1) == 0) {
printf("Делитель: ");
scanf("%f",&SecondNumeral);
if(SecondNumeral == 0) {
printf("Ошибка: деление на ноль! ");
return(HUGE_VAL);
}
else
return(Numeral / SecondNumeral);
}
else if(strncmp(0peration, "pow", 3) == 0) {
printf("Степень: ");
scanf("%f",&SecondNumeral);
return(pow(Numeral, SecondNumeral));
}
else if(strncmp(0peration, "sqrt", 4) == 0)
return(sqrt(Numeral));
else if(strncmp(0peration, "sin", 3) == 0)
```

- Калькулятор способен складывать, вычитать, умножать, делить, возводить число в степень, вычислять квадратный корень, вычислять sin, cos, tan.

File Edit Options Buffers Tools C Help



Save



Undo



```
else if(strncmp(0peration, "sin", 3) == 0)
    return(sin(Numeral));
else if(strncmp(0peration, "cos", 3) == 0)
    return(cos(Numeral));
else if(strncmp(0peration, "tan", 3) == 0)
    return(tan(Numeral));
else {
    printf("Неправильно введено действие ");
    return(HUGE_VAL);
}
}
```

3. Выполнил компиляцию программы посредством gcc.

```
[amukhamediyar@amukhamediyar lab_prog]$ gcc -c calculate.c
[amukhamediyar@amukhamediyar lab_prog]$ gcc -c main.c
[amukhamediyar@amukhamediyar lab_prog]$ gcc calculate.o main.o -o calcul -lm
[amukhamediyar@amukhamediyar lab_prog]$
```

4. Исправил синтаксические ошибки.

5. Создал *makefile*. В содержании файла указаны флаги компиляции, тип компилятора и файлы, которые должен собрать сборщик.



```
# Makefile
```

```
CC = gcc
```

```
CFLAGS =
```

```
LIBS = -lm
```

```
calcul: calculate.o main.o
```

```
gcc calculate.o main.o
```

```
-o calcul $(LIBS)
```

```
calculate.o: calculate.c calculate.h
```

```
gcc -c calculate.c $(CFLAGS)
```

```
main.o: main.c calculate.h
```

```
gcc -c main.c $(CFLAGS)
```

```
clean:
```

```
-rm calcul *.o *~
```

```
# End Makefile
```

6. С помощью *gdb* выполнил отладку программы. Но с начало исправил *makefile*.



Save



Undo



```
CC = gcc
CFLAGS = -g
LIBS = -lm

calcul: calculate.o main.o
gcc calculate.o main.o -o calcul $(LIBS)

calculate.o: calculate.c calculate.h
gcc -c calculate.c $(CFLAGS)

main.o: main.c calculate.h
gcc -c main.c $(CFLAGS)

clean: -rm calcul *.o *~
```

- После ввел следующую команду.

```
[amukhamediyar@amukhamediyar lab_prog]$ make clean
rm calcul *.o *~
[amukhamediyar@amukhamediyar lab_prog]$ make
gcc -c calculate.c -g
gcc -c main.c -g
gcc calculate.o main.o -o calcul -lm
```

- Запустил отладчик GDB, загрузив в него программу для отладки: `gdb ./calcul`.

```
[amukhamediyar@amukhamediyar lab_prog]$ gdb ./calcul
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/amukhamediyar/work/os/lab_prog/calcul...(no debugging symbol
s found)...done.
(gdb) █
```

- Для запуска программы внутри отладчика ввел команду *run*.

```
(gdb) run
Starting program: /home/amukhamediyar/work/os/lab_prog/./calcul
Число: 7
Операция (+, -, *, /, pow, sqrt, sin, cos, tan): +
Второе слагаемое: 112
119.00
[Inferior 1 (process 3579) exited normally]
Missing separate debuginfos, use: debuginfo-install glibc-2.17-324.el7_9.x86_64
(gdb)
```

- Для постраничного (по 9 строк) просмотра исходного кода использовал команду *list*.

```
(gdb) list
1      #include <stdio.h>
2      #include "calculate.h"
3
4      int main (void) {
5          float Numeral;
6          char Operation[4];
7          float Result;
8          printf("Число: ");
9          scanf("%f",&Numeral);
10         printf("Операция (+, -, *, /, pow, sqrt, sin, cos, tan): ");
(gdb)
```

- Для просмотра строк с 12 по 15 основного файла использовал *list* с параметрами: *list 12,15*.


```
(gdb) list 12, 15
12      Result = Calculate(Numeral, Operation);
13      printf("%6.2f\n",Result);
14      return 0;
15  }
```

- Для просмотра определённых строк не основного файла использовал *list* с параметрами: *list calculate.c:20,29*.

```
(gdb) list calculate.c:20,29
20      }
21      else if(strncmp(Operation, "*", 1) == 0)
22      {
23          printf("Множитель: ");
24          scanf("%f",&SecondNumeral);
25          return(Numeral * SecondNumeral);
26      }
27      else if(strncmp(Operation, "/", 1) == 0)
28      {
29          printf("Делитель: ");
(gdb)
```

- Установил точку останова в файле *calculate.c* на строке номер 21:

```
list calculate.c:20,27 break 21
```

```
(gdb) list calculate.c:20,27
20      }
21      else if(strncmp(Operation, "*", 1) == 0)
22      {
23          printf("Множитель: ");
24          scanf("%f",&SecondNumeral);
25          return(Numeral * SecondNumeral);
26      }
27      else if(strncmp(Operation, "/", 1) == 0)
(gdb) break 21
Breakpoint 1 at 0x400810: file calculate.c, line 21.
(gdb)
```

- Вывел информацию об имеющихся в проекте точка останова: *info breakpoints*

```
(gdb) info breakpoints
Num      Type           Disp Enb Address                  What
1        breakpoint     keep y   0x000000000000400810 in Calculate at calculate.c:21
(gdb)
```

- Запустил программу внутри отладчика. Отладчик выдал следующую информацию, а команда *backtrace* показала весь стек вызываемых функций от начала программы.

Посмотрел, чему равно на этом этапе значение переменной *Numeral*, введя:

```
print Numeral
```

Сравнил с результатом вывода на экран после использования команды:

```
display Numeral
```

```
(gdb) run
Starting program: /home/amukhamediyar/work/os/lab_prog/./calcul
Число: 5
Операция (+, -, *, /, pow, sqrt, sin, cos, tan): -
Вычитаемое: backtrace
5.00
[Inferior 1 (process 6292) exited normally]
Missing separate debuginfos, use: debuginfo-install glibc-2.17-324.el7_9.x86_64
(gdb) █
```

- Убрал точки останова:

```
info breakpoints delete 1
```

```
(gdb) info breakpoints
Num      Type           Disp Enb Address                  What
1        breakpoint     keep y   0x000000000000400810 in Calculate at calculate.c:21
(gdb) delete 1
(gdb) █
```

7. С помощью утилиты *splint* попробовал проанализировать коды файлов **calculate.c**.

```
[amukhamediyar@amukhamediyar lab_prog]$ splint calculate.c
Splint 3.1.2 --- 11 Oct 2015
```

```
calculate.h:3:37: Function parameter Operation declared as manifest array (size
                    constant is meaningless)
  A formal parameter is declared as an array with size.  The size of the array
  is ignored in this context, since the array formal parameter is treated as a
  pointer. (Use -fixedformalarray to inhibit warning)
calculate.c:6:37: Function parameter Operation declared as manifest array (size
                    constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:12:5: Return value (type int) ignored: scanf("%f", &Sec...
  Result returned by function call is not used. If this is intended, can cast
  result to (void) to eliminate message. (Use -retvalint to inhibit warning)
calculate.c:18:5: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:24:5: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:30:5: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:31:8: Dangerous equality comparison involving float types:
                    SecondNumeral == 0
  Two real (float, double, or long double) values are compared directly using
  == or != primitive. This may produce unexpected results since floating point
  representations are inexact. Instead, compare the difference to FLT_EPSILON
  or DBL_EPSILON. (Use -realcompare to inhibit warning)
calculate.c:33:13: Return value type double does not match declared type float:
                    (HUGE_VAL)
  To allow all numeric types to match, use +relaxtypes.
```

- С помощью утилиты *splint* попробовал проанализировать коды файлов **main.c**.

```
[amukhamediyar@amukhamediyar lab_prog]$ splint main.c
Splint 3.1.2 --- 11 Oct 2015

calculate.h:3:37: Function parameter Operation declared as manifest array (size
                    constant is meaningless)
    A formal parameter is declared as an array with size.  The size of the array
    is ignored in this context, since the array formal parameter is treated as a
    pointer. (Use -fixedformalarray to inhibit warning)
main.c: (in function main)
main.c:9:3: Return value (type int) ignored: scanf("%f", &Num...
    Result returned by function call is not used. If this is intended, can cast
    result to (void) to eliminate message. (Use -retvalint to inhibit warning)
main.c:11:14: Format argument 1 to scanf (%s) expects char * gets char [4] *:
                    &Operation
    Type of parameter is not consistent with corresponding code in format string.
    (Use -formattype to inhibit warning)
    main.c:11:11: Corresponding format code
main.c:11:3: Return value (type int) ignored: scanf("%s", &Ope...

Finished checking --- 4 code warnings
[amukhamediyar@amukhamediyar lab_prog]$ █
```

Вывод

Приобрел простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования C калькулятора с простейшими функциями.

Библиография

[Основы операционной системы UNIX](#)

[Программы UNIX-подобных операционных систем](#)

Ответы на контрольные вопросы:

1. Информацию об этих программах можно получить с помощью функций info и man.

2. Unix поддерживает следующие основные этапы разработки приложений:

• создание исходного кода программы; - представляется в виде файла

• сохранение различных вариантов исходного текста;

• анализ исходного текста; необходимо отслеживать изменения исходного кода, а также при работе более двух программистов над проектом программы нужно, чтобы они не делали изменений кода в одно время.

• компиляция исходного текста и построение исполняемого модуля;

• тестирование и отладка; - проверка кода на наличие ошибок

• сохранение всех изменений, выполняемых при тестировании и отладке.

3. Использование суффикса ".c" для имени файла с программой на языке Си отражает удобное и полезное соглашение, принятое в ОС UNIX. Для любого имени входного файла суффикс определяет какая компиляция требуется. Суффиксы и префиксы указывают тип объекта. Одно из полезных свойств компилятора Си — его способность по суффиксам определять типы файлов. По суффиксу .c компилятор распознает, что файл abcd.c должен компилироваться, а по суффиксу .o, что файл abcd.o является объектным модулем и для получения исполняемой программы необходимо выполнить редактирование связей. Простейший пример командной строки для компиляции программы abcd.c и построения исполняемого модуля abcd имеет вид: gcc -o abcd abcd.c. Некоторые проекты предпочитают показывать префиксы в начале текста изменений для старых (old) и новых (new) файлов. Опция — prefix может быть использована для установки такого префикса. Плюс к этому команда bzr diff -p1 выводит префиксы в форме которая подходит для команды patch -p1.
4. Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.
5. При разработке большой программы, состоящей из нескольких исходных файлов заголовков, приходится постоянно следить за файлами, которые требуют перекомпиляции после внесения изменений. Программа make освобождает пользователя от такой рутинной работы и служит для документирования взаимосвязей между файлами. Описание взаимосвязей и соответствующих действий хранится в так называемом make-файле, который по умолчанию имеет имя makefile или Makefile.
6. В общем случае make-файл содержит последовательность записей (строк), определяющих зависимости между файлами. Первая строка записи представляет собой список целевых (зависимых) файлов, разделенных пробелами, за которыми следует двоеточие и список файлов, от которых зависят целевые. Текст, следующий за точкой с запятой, и все последующие строки, начинающиеся с литеры табуляции, являются командами ОС UNIX, которые необходимо выполнить для обновления целевого файла. Таким образом, спецификация взаимосвязей имеет формат: target1 [target2...]: [:] [dependment1...] [(tab)commands] [#commentary] [(tab)commands] [#commentary], где # — специфицирует начало комментария, так как содержимое строки, начиная с # и до конца строки, не будет обрабатываться командой make; ; — последовательность команд ОС UNIX должна содержаться в одной строке make-файла (файла описаний), есть возможность переноса команд (), но она считается как одна строка; :: — последовательность команд ОС UNIX может содержаться в нескольких последовательных строках файла описаний.
7. Пошаговая отладка программ заключается в том, что выполняет один оператор программы и, затем контролируются те переменные, на которые должен был воздействовать данный оператор. Если в программе имеются уже отлаженные подпрограммы, то подпрограмму можно рассматривать, как один оператор программы и воспользоваться вторым способом отладки программ. Если в программе существует достаточно большой участок программы, уже отлаженный ранее, то его можно выполнить, не контролируя переменные, на которые он воздействует. Использование точек останова позволяет пропускать уже отлаженную часть программы. Точка останова устанавливается в местах, где необходимо проверить содержимое переменных или просто проконтролировать, передаётся ли управление данному оператору.
8. backtrace - вывод на экран пути к текущей точке останова (по сути вывод названий всех функций) break - установить точку останова (в качестве параметра может быть указан номер строки или название функции) clear - удалить все точки останова в функции continue - продолжить выполнение программы delete - удалить точку останова display - добавить выражение в список выражений, значения которых отображаются при достижении точки останова программы finish - выполнить программу до момента выхода из функции info breakpoints - вывести на экран список используемых точек останова info watchpoints - вывести на экран список используемых контрольных выражений list - вывести на экран исходный код (в качестве параметра может быть указано название файла и через двоеточие номера начальной и конечной строк) next - выполнить программу пошагово, но без выполнения вызываемых в программе функций print - вывести значение указываемого в качестве параметра выражения run - запуск программы на выполнение set - установить новое значение переменной step - пошаговое выполнение программы watch - установить контрольное выражение, при изменении значения которого программа будет остановлена
9.
 - Выполнил компиляцию программы.
 - Увидел ошибки в программе.
 - Открыл редактор и исправил программу.
 - Загрузил программу в отладчик *gdb*.
 - run — отладчик выполнил программу, ввела требуемые значения.
 - Использовал другие команды отладчика и проверил работу программы.
10. Отладчику не понравился формат %s для &Operation, т.к %s — символьный формат, а значит необходим только Operation.
11. Если вы работаете с исходным кодом, который не вами разрабатывался, то назначение различных конструкций может быть не совсем понятным. Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся:

- cscope - исследование функций, содержащихся в программе;
- splint — критическая проверка программ, написанных на языке Си.

12.
 - Проверка корректности задания аргументов всех использованных в программе функций, а также типов возвращаемых ими значений;
 - Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка Си, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки;
 - Общая оценка мобильности пользовательской программы.