

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 13

дисциплина: Операционные системы

Студент группы НПИбд-01-21

Студенческий билет № 1032205621

Фамилия Имя Отчество Дессие Абди Бедаса

МОСКВА

20 ____ г.

Цель работы:

Приобретение простейших навыков разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

Ход работы:

- В домашнем каталоге создали подкаталог ~/work/os/lab_prog

The screenshot shows a Linux desktop environment with a dark theme. At the top, there is a header bar with 'Activities' and 'Terminal' tabs, the date 'Jun 1 06:02', and system icons for battery, volume, and network. Below the header is a dock with icons for Home, Search, and other applications. A file manager window is open, showing the user's home directory (~). The terminal window, titled 'dabedasa@fedora:~', displays the command 'mkdir ~/work_os_lab_prog.' followed by the output of 'ls' which includes 'Desktop', 'Downloads', 'laba903.sh~', 'Pictures', 'Templates', 'work_os_lab_prog.', 'Documents', 'laba4.sh~', 'Music', 'Public', and 'Videos'. A tooltip at the bottom right of the screen says "'work\os\prog.' selected (containing 0 items)'.

```
[dabedasa@fedora ~]$ mkdir ~/work_os_lab_prog.  
[dabedasa@fedora ~]$ ls  
Desktop Downloads laba903.sh~ Pictures Templates work_os_lab_prog.  
Documents laba4.sh~ Music Public Videos 'work\os\prog.'  
[dabedasa@fedora ~]$
```

"work\os\prog." selected (containing 0 items)

- Создали в нём файлы: calculate.h, calculate.c, main.c.

The screenshot shows a Fedora desktop environment. In the top panel, there is an 'Activities' button, a 'Terminal' icon, the date 'Jun 1 06:05', and system icons for battery, signal, and volume. Below the panel, a file manager window is open, showing a sidebar with icons for Home, Documents, Downloads, Pictures, and Videos. The main area displays a folder named 'work_os_lab_prog.' containing three files: 'calculate.h', 'calculate.c', and 'main.c'. A terminal window is also open, titled 'dabedasa@fedora:~/work_os_lab_prog.', showing the command history:

```
[dabedasa@fedora ~]$ cd ~/work_os_lab_prog.  
[dabedasa@fedora work_os_lab_prog.]$ touch calculate.h  
[dabedasa@fedora work_os_lab_prog.]$ touch calculate.c  
[dabedasa@fedora work_os_lab_prog.]$ touch main.c  
[dabedasa@fedora work_os_lab_prog.]$
```

A tooltip at the bottom right of the screen says: "work\os\prog." selected (containing 0 items).

Реализация функций калькулятора в файле calculate.c:

Activities Text Editor Jun 4 15:40

Open calculate.c ~/work_os_lab_prog. Save

ggo calculate.c

```
1 #include <stdio.h>
2 #include <math.h>
3 #include <string.h>
4 #include "calculate.h"
5 float
6 calculate(float Numeral, char operation[4])
7 {
8     float SecondNumeral;
9     if (strcmp(Operation, "+", 1) == 0)
10    {
11        printf("second term: ");
12        scanf("%f", $SecondNumeral);
13        return (Numeral + SecondNumeral);
14    }
15    else if (strcmp(Operation, "-", 1) == 0)
16    {
17        printf("subtrahend: ");
18        scanf("%f", $SecondNumeral );
19        return (Numeral - SecondNumeral);
20    }
21    else if (strcmp(Operation, "*", 1) == 0)
22    {
23        printf(" factor: ")
24        scanf("%f", $SecondNumeral);
25        return (Numeral*SecondNumeral);
26    }
27    else if (strcmp(Operation, "/", 1) == 0)
28    {
```

C ▾ Tab Width: 8 ▾ Ln1, Col1 ▾ INS

Activities Text Editor Jun 4 15:41

Open + calculate.c ~ /work_os_lab_prog. Save ☰ ×

ggo × calculate.c ×

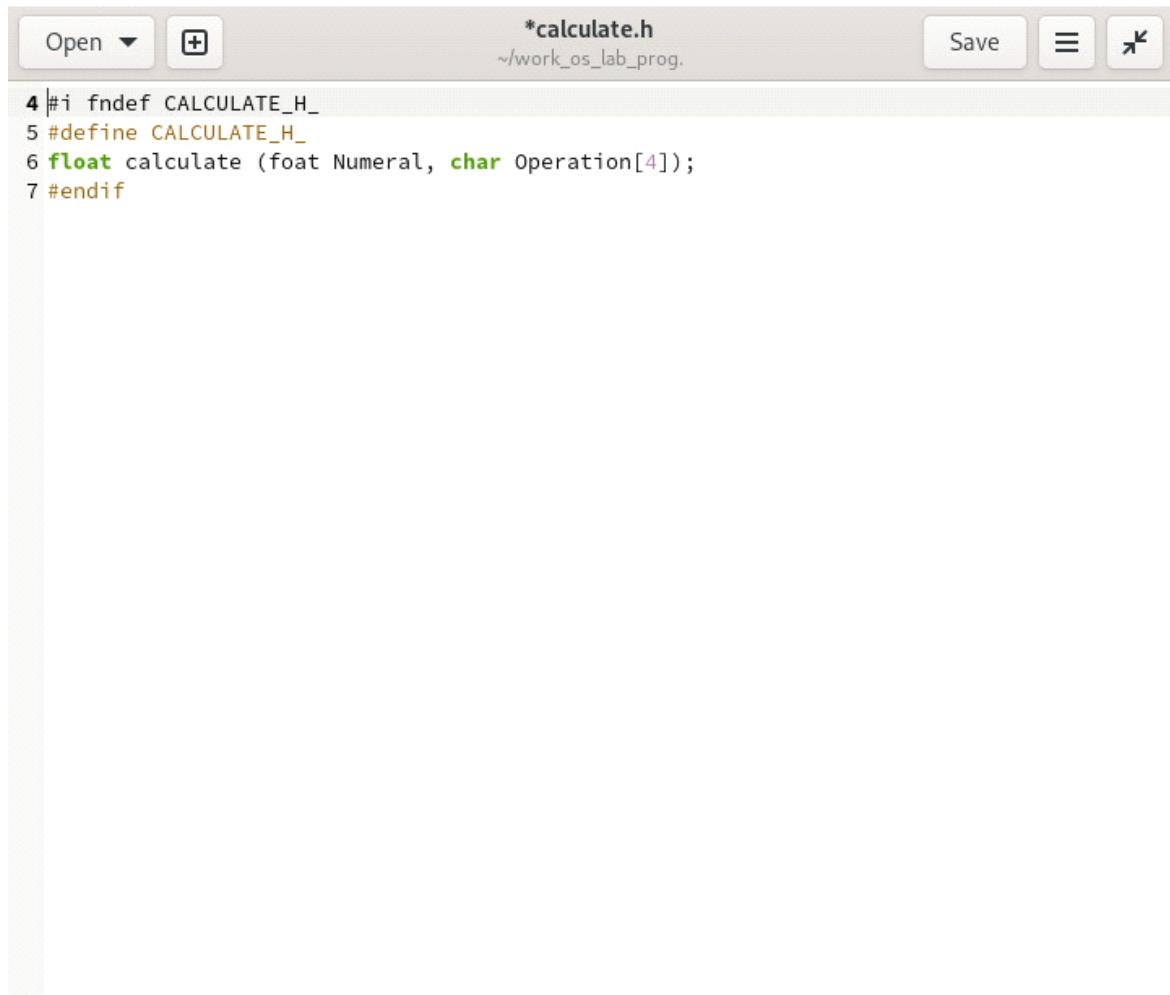
```
29     print ("divider : ");
30     scanf("%f", $SecondNumeral);
31     if (SecondNumeral == 0)
32     {
33         print("error: division by zero!");
34         return (HUGE_VAL);
35     }
36     else
37         return (Numeral/SecondNumeral);
38     }
39     else if (strncmp(Operation, "*", 1) == 0)
40     {
41         printf("factor: ");
42         scanf("%f", $SecondNumeral);
43         return (Numeral*SecondNumeral);
44     else if (strncmp(Operation, "*", 1) == 0)
45     {
46         printf( "divisor: ");
47         scanf("%f", $SecondNumeral);
48         if (secondNumeral == 0)
49         {
50             print("error: division by zero! ");
51             return (HUGE_VAL);
52         }
53         else
54             return (Numeral/SecondNumeral);
55     }
56     else if (strncmp(Operation, "pow", 3) == 0)
57     {
```

C ▾ Tab Width: 8 ▾ Ln1, Col1 ▾ INS

The screenshot shows a Linux desktop interface. In the top panel, there is an 'Activities' button, a 'Text Editor' icon, the date and time 'Jun 4 15:41', and several system icons. Below the top panel, the main window is a text editor titled 'calculate.c' located at '~/.work_os_lab_prog.'. The file contains C code for a calculator. The code includes various conditional statements (if-else blocks) for different operations like addition, subtraction, multiplication, division, power, square root, sine, cosine, and tangent. It also handles an 'else' block for incorrect input. The code uses standard C syntax with printf and scanf functions. The text editor has tabs for 'calculate.c' and 'ggo'. At the bottom of the editor window, there are status indicators for 'C', 'Tab Width: 8', 'Ln1, Col1', and 'INS'.

```
52     }
53     else
54         return (Numeral/SecondNumeral);
55     }
56 else if (strncmp(Operation, "pow", 3) == 0)
57 {
58     printf("power: ");
59     scanf("%f", $SecondNumeral);
60     return(pow(Numeral, SecondNumberal));
61 }
62 else if (strncmp(Operation, "sqrt", 4)== 0)
63 return (sqrt(Numeral));
64 else if (strncmp(Operation, "sin", 3) == 0)
65 return (sin(Numeral));
66 else if (strncmp(Operation, "cos", 3) == 0)
67 return (cos(Numeral));
68 else if (strncmp(Operation, "tan", 3) == 0)
69 return (tan(Numeral));
70 else
71 {
72     printf("action entered incorrectly");
73     return (HUGE_VAL);
74 }
75 }
76
77
78
79
```

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



```
*calculate.h
~/work_os_lab_prog.

4 #i fndef CALCULATE_H_
5 #define CALCULATE_H_
6 float calculate (float Numeral, char Operation[4]);
7 #endif
```

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

The screenshot shows a desktop environment with a terminal window and a text editor window. The terminal window at the bottom has a command prompt and some output. The text editor window is titled 'main.c' and contains C code for a calculator program. The code includes file includes, a main function that reads a number and an operation from the user, performs a calculation, and prints the result.

```
1 #include <stdio.h>
2 #include "calculate.h"
3 int main (void)
4 {
5     float Numeral;
6     char Operation[4];
7     float Result;
8     printf("number:");
9     scanf("%f", &Numeral);
10    printf("operation (+, -, *, /, pow, sqrt, sin, cos, tan): ");
11    scanf("%s", Operation);
12    Result = Calculate(Numeral, Operation);
13    printf("%6.2f \n ", Result);
14    return 0;
15 }
```

- Выполнили компиляцию программы посредством gcc:
- В файле main.c допущена ошибка в строке «scanf("%s",&Operation);», не нужен &.
- Создали Makefile со следующим содержанием:

Activities Terminal Jun 3 01:56

Home work_os_lab_prog.

dabedasa@fedora:~

```
[dabedasa@fedora ~]$ touch makefile
[dabedasa@fedora ~]$ ls
Desktop      laba4.sh~    Music       Templates      'work\os\prog.'
Documents    laba903.sh~  Pictures     Videos
Downloads    makefile    Public      work_os_lab_prog.
[dabedasa@fedora ~]$
```

#if
#de
flo
end

- :*
Wel
To
To

Imp
Ema
Rea
(No
Cop
U:9
Aut

The screenshot shows a terminal window titled "Text Editor" with the status bar indicating "Jun 3 02:10". The window contains a makefile with the following content:

```
1 CC = gcc
2 CFLAGS =
3 LIBS = -lm
4 calcul: calculate.o main.o
5     gcc calculate.o main.o -o calcul $(LIBS)
6 calculate.o: calculate.c calculate.h
7     gcc -c calculate.c $(CFLAGS)
8 main.o: main.c calculate.h
9     gcc -c main.c $(CFLAGS)
10 clean: -rm calcul *.o *~
11
```

The terminal window has a standard Linux-style interface with "Activities", "Text Editor", and "Save" buttons at the top. At the bottom, there are dropdown menus for "Makefile", "Tab Width: 8", and "Ln 10, Col 25", along with a status indicator "INS".

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

- Выполнили отладку программы calcul. Для отладки используем gdb.

Activities Terminal Jun 3 02:13

makefile

dabedasa@fedora:~/work_os_lab_prog.— gdb ./calcul

```
[dabedasa@fedora work_os_lab_prog.]$ gdb ./calcul
GNU gdb (GDB) Fedora 10.2-9.fc35
Copyright (C) 2021 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".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
./calcul: No such file or directory.
(gdb)
```

Makefile Tab Width: 8 Ln 10, Col 25 INS

- С помощью утилиты `splint` проанализировали коды файлов `calculate.c` и `main.c`.

Activities Terminal Jun 4 15:04

dabedasa@fedora:~/work_os_lab_prog.

```
[dabedasa@fedora work_os_lab_prog.]$ splint main.c
Splint 3.1.2 --- 23 Jul 2021

calculate.h:3:38: 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:6:20: Parse Error. (For help on parse errors, see splint -help
    parseerrors.)
*** Cannot continue.
[dabedasa@fedora work_os_lab_prog.]$
```

The screenshot shows a terminal window titled "dabedasa@fedora:~/work_os_lab_prog." with the command "[dabedasa@fedora work_os_lab_prog.]\$ splint calculate.c" running. The output of the command is displayed, showing various warnings and errors from the splint tool. The errors include:

- calculate.h:3:38: 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:6:31: Function parameter operation declared as manifest array (size constant is meaningless)
- calculate.c: (in function calculate)
- calculate.c:9:16: Unrecognized identifier: Operation
- Identifier used in code has not been declared. (Use -unrecog to inhibit warning)
- calculate.c:12:18: Unrecognized identifier: \$SecondNumeral
- calculate.c:12:6: 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:13:24: Variable SecondNumeral used before definition
- An rvalue is used that may not be initialized to a value on some execution path. (Use -usedef to inhibit warning)
- calculate.c:18:5: Return value (type int) ignored: scanf("%f", \$Sec...)
- calculate.c:19:23: Variable SecondNumeral used before definition
- calculate.c:24:10: Parse Error. (For help on parse errors, see splint -help)

Вывод:

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

Ответы на контрольные вопросы:

- Дополнительную информацию о этих программах можно получить с помощью функций `info` и `man`.
- 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.
- 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.
- Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.
- 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.
- makefile для программы abcd.c мог бы иметь вид:

```

#
#
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
```

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. Приведённый выше make-файл для программы abcd.c включает два способа компиляции и построения исполняемого модуля. Первый способ предусматривает обычную компиляцию с построением исполняемого модуля с именем abcd. Второй способ позволяет включать в исполняемый модуль testabcd возможность выполнить процесс отладки на уровне исходного текста.

- Пошаговая отладка программ заключается в том, что выполняется 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 отложенную часть программы. Точка останова устанавливается в местах, где необходимо проверить содержимое переменных или просто проконтролировать, передаётся ли управление данному оператору. Практически во всех отладчиках поддерживается это свойство (а также выполнение программы до курсора и выход из подпрограммы). Затем отладка программы продолжается в пошаговом режиме с контролем локальных и глобальных переменных, а

также внутренних регистров микроконтроллера и напряжений на выводах этой микросхемы.

- - backtrace – выводит весь путь к текущей точке останова, то есть названия всех функций, начиная от main(); иными словами, выводит весь стек функций;
 - break – устанавливает точку останова; параметром может быть номер строки или название функции;
 - clear – удаляет все точки останова на текущем уровне стека (то есть в текущей функции);
 - continue – продолжает выполнение программы от текущей точки до конца;
 - delete – удаляет точку останова или контрольное выражение;
 - display – добавляет выражение в список выражений, значения которых отображаются каждый раз при остановке программы;
 - finish – выполняет программу до выхода из текущей функции; отображает возвращаемое значение, если такое имеется;
 - info breakpoints – выводит список всех имеющихся точек останова;
 - info watchpoints – выводит список всех имеющихся контрольных выражений;
 - splist – выводит исходный код; в качестве параметра передаются название файла исходного кода, затем, через двоеточие, номер начальной и конечной строки;
 - next – пошаговое выполнение программы, но, в отличие от команды step, не выполняет пошагово вызываемые функции;
 - print – выводит значение какого-либо выражения (выражение передаётся в качестве параметра);
 - run – запускает программу на выполнение;
 - set – устанавливает новое значение переменной
 - step – пошаговое выполнение программы;
 - watch – устанавливает контрольное выражение, программа остановится, как только значение контрольного выражения изменится;
- 1) Выполнили компиляцию программы 2) Увидели ошибки в программе 3) Открыли редактор и исправили программу 4) Загрузили программу в отладчик gdb 5) run — отладчик выполнил программу, мы ввели требуемые значения. 6) программа завершена, gdb не видит ошибок.
- 1 и 2.) Мы действительно забыли закрыть комментарии; 3.) отладчику не понравился формат %s для &Operation, т.к %s — символьный формат, а значит необходим только Operation.
- 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.
- 1. Проверка корректности задания аргументов всех использованных в программе функций, а также типов возвращаемых ими значений;
- 2. Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка Си, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки;
- 3. Общая оценка мобильности пользовательской программы.