

Лабораторная работа №13

Дисциплина: Операционные системы

Аксенова Анастасия

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Библиография	24
5	Выводы	25

List of Tables

List of Figures

3.1	Создание подкаталога	7
3.2	Создание файлов	7
3.3	Программа в calculate.c	8
3.4	Программа в calculate.c	8
3.5	Программа в calculate.h	9
3.6	Программа в main.c	9
3.7	Компиляция программы	10
3.8	Программа в Makefile.....	10
3.9	Программа в Makefile.....	11
3.10	Удаление файлов.....	11
3.11	Компиляция файлов.....	11
3.12	Работа с gdb	12
3.13	Работа с gdb - run	12
3.14	Работа с gdb - list.....	13
3.15	Работа с gdb - list 12,15	13
3.16	Работа с gdb - list calculate.c:20,29	14
3.17	Работа с gdb - list calculate.c:20,27	15
3.18	Работа с gdb - info breakpoints.....	15
3.19	Работа с gdb - run	16
3.20	Работа с gdb - print Numeral	17
3.21	Работа с gdb - display Numeral	17
3.22	Работа с gdb - info breakpoints.....	18
3.23	Результат команды splint calculate.c	19
3.24	Результат конмады splint main.c	19

1 Цель работы

Цель данной лабораторной 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.

2 Задание

1. Сделать отчёт по лабораторной работе №14 в формате Markdown.
2. создать на языке программирования С калькулятора с простейшими функциями.

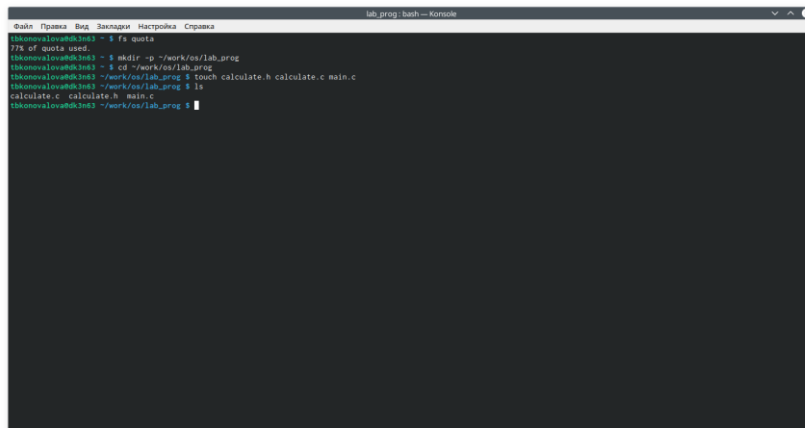
3 Выполнение лабораторной работы

1). В домашнем каталоге создаю подкаталог `~/work/os/lab_prog` помощью команды «`mkdir -p ~/work/os/lab_prog`» (Рисунок 3.1). Вся необходимая информация про создания каталогов указана в следующем источнике: Программное обеспечение GNU/Linux. Лекция 9. Хранилище и дистрибутив (Г. Курячий, МГУ).

```
tbkonovalova@dk3n63 ~ $ mkdir -p ~/work/os/lab_prog
tbkonovalova@dk3n63 ~ $
```

Figure 3.1: Создание подкаталога

2). Создала в каталоге файлы: `calculate.h`, `calculate.c`, `main.c`, используя команды «`cd ~/work/os/lab_prog`» и «`touch calculate.h calculate.c main.c`» (алгоритм действий представлен на рис. 3.2).

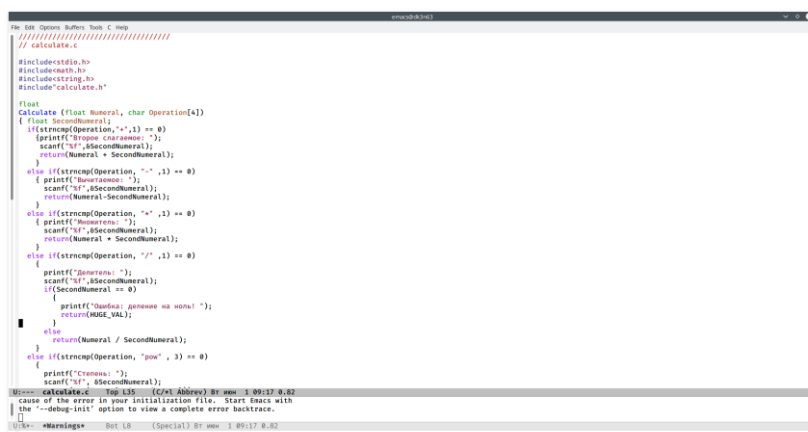


```
tbkonovalova@dk3n63 ~ $ cd ~/work/os/lab_prog
tbkonovalova@dk3n63 ~/work/os/lab_prog $ touch calculate.h calculate.c main.c
tbkonovalova@dk3n63 ~/work/os/lab_prog $ ls
calculate.c calculate.h main.c
tbkonovalova@dk3n63 ~/work/os/lab_prog $
```

Figure 3.2: Создание файлов

Это будет примитивнейший калькулятор, способный складывать, вычитать,

умножать и делить, возводить число в степень, брать квадратный корень, вычислять \sin , \cos , \tan . При запуске он будет запрашивать первое число, операцию, второе число. После этого программа выведет результат и остановится. Открыв редактор Emacs, приступила к редактированию созданных файлов. Реализация функций калькулятора в файле `calculate.c` (Программа представлена на рис. 3.3, 3.4). Вся необходимая информация про написания программ указана в следующем источнике: Программное обеспечение GNU/Linux. Лекция 10. Минимальный набор знаний (Г. Курячий, МГУ).



```

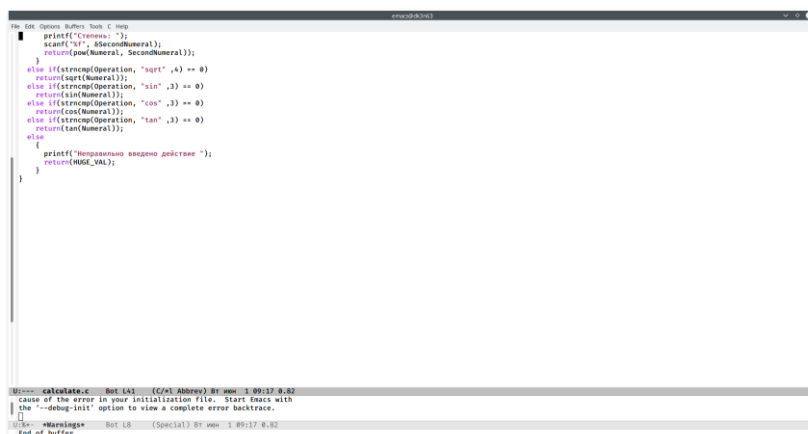
// calculate.c

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

float
calculate (float Numeral, char Operation[4])
{ float SecondNumeral;
  if (strcmp (Operation, "+") == 0)
    { printf("Введите второе число: ");
      scanf("%f", &SecondNumeral);
      return (Numeral + SecondNumeral);
    }
  else if (strcmp (Operation, "-") == 0)
    { printf("Введите второе число: ");
      scanf("%f", &SecondNumeral);
      return (Numeral - SecondNumeral);
    }
  else if (strcmp (Operation, "*") == 0)
    { printf("Введите второе число: ");
      scanf("%f", &SecondNumeral);
      return (Numeral * SecondNumeral);
    }
  else if (strcmp (Operation, "/") == 0)
    { printf("Введите второе число: ");
      scanf("%f", &SecondNumeral);
      if (SecondNumeral == 0)
        { printf("Ошибка! деление на ноль!");
          return (NOE_VAL);
        }
      return (Numeral / SecondNumeral);
    }
  else if (strcmp (Operation, "pow", 3) == 0)
    { printf("Введите второе число: ");
      scanf("%f", &SecondNumeral);
      return (pow (Numeral, SecondNumeral));
    }
  }
}

```

Figure 3.3: Программа в `calculate.c`



```

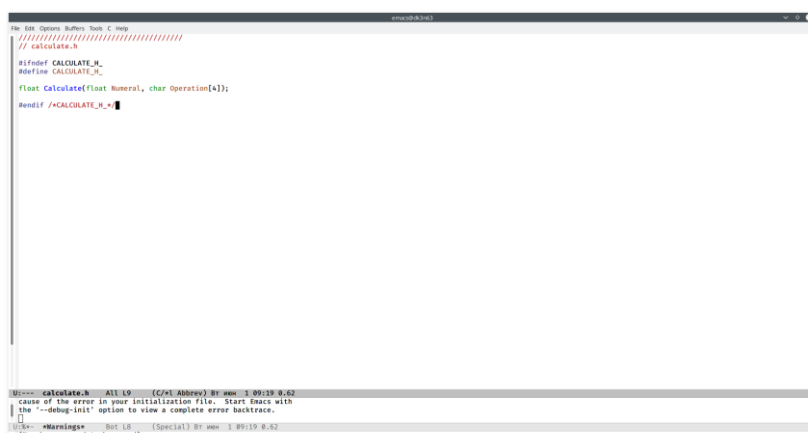
    { printf("Введите второе число: ");
      scanf("%f", &SecondNumeral);
      return (sqrt (Numeral));
    }
  else if (strcmp (Operation, "sin", 3) == 0)
    { return (sin (Numeral));
    }
  else if (strcmp (Operation, "cos", 3) == 0)
    { return (cos (Numeral));
    }
  else if (strcmp (Operation, "tan", 3) == 0)
    { return (tan (Numeral));
    }
  else
    { printf("Неправильно введено действие ");
      return (NOE_VAL);
    }
  }
}

```

Figure 3.4: Программа в `calculate.c`

Интерфейсный файл `calculate.h`, описывающий формат вызова функции каль-

кулятора (Скриншот 3.5). Подробная информация о написании программ в Linux указана в следующем источнике: Электронный ресурс: Электронный ресурс: <https://it.wikireading.ru/34160>



```
File Edit Options Buffers Tools C Help
calculate.h
// calculate.h
#ifndef CALCULATE_H_
#define CALCULATE_H_

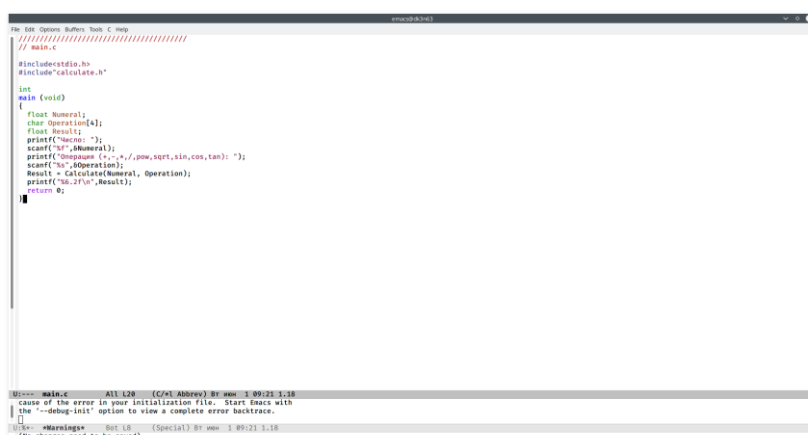
float Calculate(float Numeral, char Operation[4]);

#endif // CALCULATE_H_

U:--- calculate.h All L9 (C/*1 Abbrev) Br new 1 09:19 8.62
cause of the error in your initialization file. Start Emacs with
the --debug-init option to view a complete error backtrace.
U:--- Warnings Bot L8 (Special) Br new 1 09:19 8.62
(No changes need to be saved)
```

Figure 3.5: Программа в calculate.h

Основной файл main.c, реализующий интерфейс пользователя к калькулятору (Скриншот 3.6).



```
File Edit Options Buffers Tools C Help
main.c
// main.c
#include <stdio.h>
#include "calculate.h"

int
main (void)
{
    float Numeral;
    char Operation[4];
    float Result;
    printf("Numeral: ");
    scanf("%f", &Numeral);
    printf("Operation (+, -, *, /, pow, sqrt, sin, cos, tan): ");
    scanf("%s", &Operation);
    Result = Calculate(Numeral, Operation);
    printf("%s: %f\n", Operation, Result);
    return 0;
}
```

Figure 3.6: Программа в main.c

3). Выполнила компиляцию программы посредством gcc (версия компилятора :8.3.0-19), используя команды «gcc -c calculate.c», «gcc -c main.c» и «gcc calculate.o main.o -o calcul -lm» (алгоритм действий представлен на рис. 3.7).

```
tbkonovalova@dk3n63 ~/work/os/lab_prog $ gcc -c calculate.c
tbkonovalova@dk3n63 ~/work/os/lab_prog $ gcc -c main.c
tbkonovalova@dk3n63 ~/work/os/lab_prog $ gcc calculate.o main.o -o calcul -lm
```

Figure 3.7: Компиляция программы

4). В ходе компиляции программы никаких ошибок выявлено не было.

5). Создала Makefile с необходимым содержанием (Рисунок 3.8).



Figure 3.8: Программа в Makefile

Данный файл необходим для автоматической компиляции файлов calculate.c (цель calculate.o), main.c (цель main.o), а также их объединения в один исполняемый файл calcul (цель calcul). Цель clean нужна для автоматического удаления файлов. Переменная CC отвечает за утилиту для компиляции. Переменная CFLAGS отвечает за опции в данной утилите. Переменная LIBS отвечает за опции для объединения объектных файлов в один исполняемый файл.

6). Далее исправила Makefile (Скриншот 3.9). Подробная информация о написании программ в Linux указана в следующем источнике: Электронный ресурс: <https://vunivere.ru/work23597>

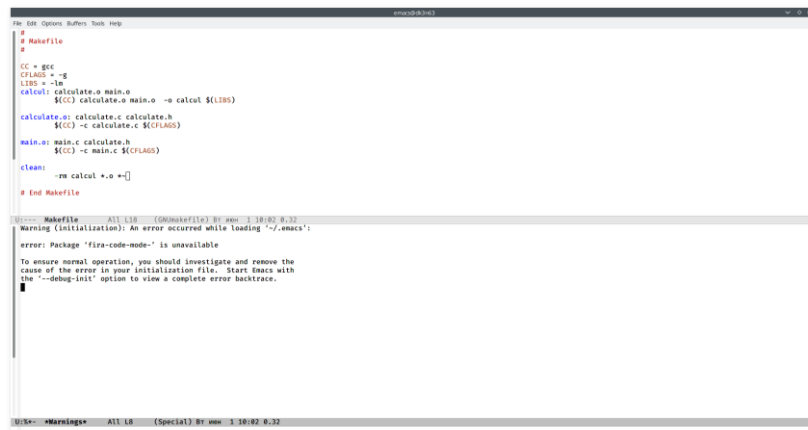


Figure 3.9: Программа в Makefile

В переменную CFLAGS добавила опцию -g, необходимую для компиляции объектных файлов и их использования в программе отладчика GDB. Сделала так, что утилита компиляции выбирается с помощью переменной CC. После этого я удалила исполняемые и объектные файлы из каталога с помощью команды «make clear» (Рисунок 3.10). Выполнила компиляцию файлов, используя команды «make calculate.o», «make main.o», «make calcul» (Рисунок 3.11).

```

tbkonovalova@dk3n63 ~/work/os/lab_prog $ make clean
rm calcul *.o *~

```

Figure 3.10: Удаление файлов

```

tbkonovalova@dk3n63 ~/work/os/lab_prog $ make calculate.o
gcc -c calculate.c -g
tbkonovalova@dk3n63 ~/work/os/lab_prog $ make main.o
gcc -c main.c -g
tbkonovalova@dk3n63 ~/work/os/lab_prog $ make calcul
gcc calculate.o main.o -o calcul -lm

```

Figure 3.11: Компиляция файлов

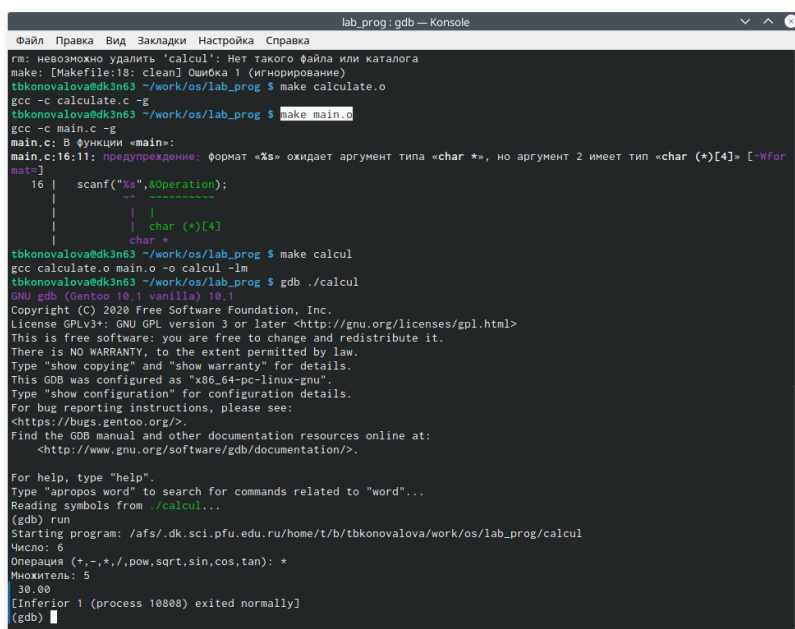
Далее с помощью gdb выполнила отладку программы calcul. Запустила отладчик GDB, загрузив в него программу для отладки, используя команду: «gdb./calcul» (алгоритм действий представлен на рис. 3.12).

```
tbkonovalova@dk3n63 ~/work/os/lab_prog $ gdb ./calcul
GNU gdb (Gentoo 10.1 vanilla) 10.1
Copyright (C) 2020 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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"...
Reading symbols from ./calcul...
(gdb) █
```

Figure 3.12: Работа с gdb

Для запуска программы внутри отладчика ввела команду «run» (Рисунок 3.13).



```
lab_prog: gdb — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
gdb: невозможно удалить 'calcul': Нет такого файла или каталога
make: [Makefile:18: clean] Ошибка 1 (игнорирование)
tbkonovalova@dk3n63 ~/work/os/lab_prog $ make calculate.o
gcc -c calculate.c -g
tbkonovalova@dk3n63 ~/work/os/lab_prog $ make main.o
gcc -c main.c -g
main.c: В функции «main»:
main.c:16:11: предупреждение: формат «%s» ожидает аргумент типа «char *», но аргумент 2 имеет тип «char (*)[4]» [-Wformat=]
   16 |     scanf("%s", &operation);
      |           ^~
      |           |
      |           | char (*)[4]
      |           |
      |           char *
tbkonovalova@dk3n63 ~/work/os/lab_prog $ make calcul
gcc calculate.o main.o -o calcul -lm
tbkonovalova@dk3n63 ~/work/os/lab_prog $ gdb ./calcul
GNU gdb (Gentoo 10.1 vanilla) 10.1
Copyright (C) 2020 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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"...
Reading symbols from ./calcul...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/t/b/tbkonovalova/work/os/lab_prog/calcul
Число: 6
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Множитель: 5
30.00
[Inferior 1 (process 10808) exited normally]
(gdb) █
```

Figure 3.13: Работа с gdb - run

Для постраничного (по10строк) просмотра исходного кода использовала команду «list» (Рисунок 3.14).

```
lab_prog: gdb — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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"...
Reading symbols from ./calcul...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/t/b/tbkonovalova/work/os/lab_prog/calcul
Число: 6
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Множитель: 5
30.00
[Inferior 1 (process 10808) exited normally]
(gdb) list
1  //////////////////////////////////////////////////
2  // main.c
3
4  #include <stdio.h>
5  #include "calculate.h"
6
7  int
8  main (void)
9  {
10     float Numeral;
(gdb) list
11     char Operation[4];
12     float Result;
13     printf("Число: ");
14     scanf("%f",&Numeral);
15     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
16     scanf("%s",&Operation);
17     Result = Calculate(Numeral, Operation);
18     printf("%g.2f\n",Result);
19     return 0;
20 }
```

Figure 3.14: Работа с gdb - list

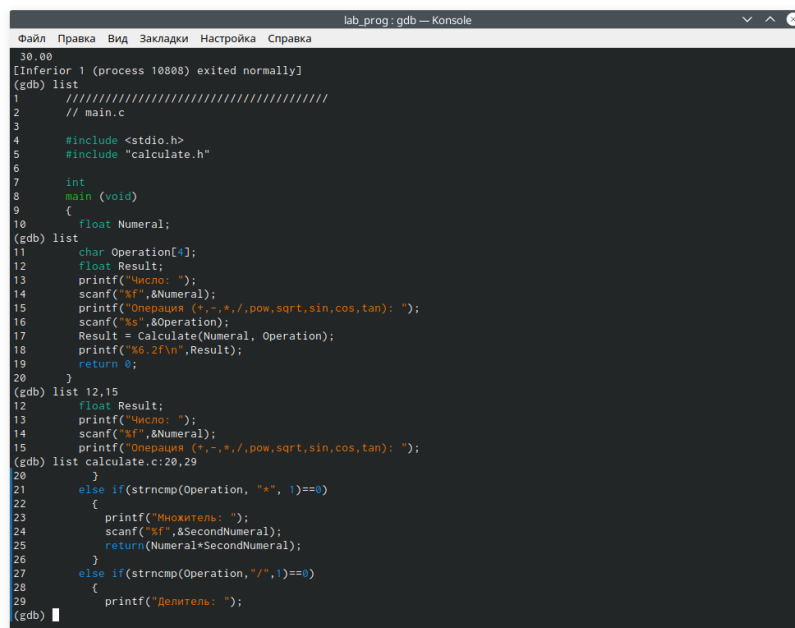
Для просмотра строк с 12 по 15 основного файла использовала команду «list 12,15» (Рисунок 3.15).

```
lab_prog: gdb — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
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"...
Reading symbols from ./calcul...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/t/b/tbkonovalova/work/os/lab_prog/calcul
Число: 6
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Множитель: 5
30.00
[Inferior 1 (process 10808) exited normally]
(gdb) list
1  //////////////////////////////////////////////////
2  // main.c
3
4  #include <stdio.h>
5  #include "calculate.h"
6
7  int
8  main (void)
9  {
10     float Numeral;
(gdb) list
11     char Operation[4];
12     float Result;
13     printf("Число: ");
14     scanf("%f",&Numeral);
15     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
16     scanf("%s",&Operation);
17     Result = Calculate(Numeral, Operation);
18     printf("%g.2f\n",Result);
19     return 0;
20 }
(gdb) list 12,15
12     float Result;
13     printf("Число: ");
14     scanf("%f",&Numeral);
15     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
(gdb)
```

Figure 3.15: Работа с gdb - list 12,15

Для просмотра определённых строк не основного файла использовала команду «list calculate.c:20,29» (Рисунок 3.16).



```
lab_prog: gdb — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
30.00
[Inferior 1 (process 10808) exited normally]
(gdb) list
1  ///////////////////////////////////////////////////////////////////
2  // main.c
3
4  #include <stdio.h>
5  #include "calculate.h"
6
7  int
8  main (void)
9  {
10     float Numeral;
(gdb) list
11     char Operation[4];
12     float Result;
13     printf("Число: ");
14     scanf("%f",&Numeral);
15     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
16     scanf("%s",&Operation);
17     Result = Calculate(Numeral, Operation);
18     printf("%g.2f\n",Result);
19     return 0;
20 }
(gdb) list 12,15
12     float Result;
13     printf("Число: ");
14     scanf("%f",&Numeral);
15     printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
(gdb) list calculate.c:20,29
20 }
21 else if(strncmp(Operation, "*", 1)==0)
22 {
23     printf("Умножитель: ");
24     scanf("%f",&SecondNumeral);
25     return (Numeral*SecondNumeral);
26 }
27 else if(strncmp(Operation, "/", 1)==0)
28 {
29     printf("Делитель: ");
```

Figure 3.16: Работа с gdb - list calculate.c:20,29

Установила точку останова в файле calculate.c на строке номер 21, используя команды «list calculate.c:20,27» и «break 21» (Рисунок 3.17).

```

lab_prog: gdb — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
19      return 0;
20    }
(gdb) list 12,15
12      float Result;
13      printf("Число: ");
14      scanf("%f",&Numeral);
15      printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
(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) 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) info breakpoints
No breakpoints or watchpoints.
(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 0x555555400991: file calculate.c, line 21.
(gdb)

```

Figure 3.17: Работа с gdb - list calculate.c:20,27

Вывела информацию об имеющихся в проекте точках останова с помощью команды «info breakpoints» (Скриншот 3.18).

```

lab_prog: gdb — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
12      float Result;
13      printf("Число: ");
14      scanf("%f",&Numeral);
15      printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
(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) 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) info breakpoints
No breakpoints or watchpoints.
(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 0x555555400991: file calculate.c, line 21.
(gdb) info breakpoints
Num  Type      Disp Enb Address             What
1    breakpoint keep y  0x0000555555400991 in Calculate at calculate.c:21
(gdb)

```

Figure 3.18: Работа с gdb - info breakpoints

Запустила программу внутри отладчика и убедилась, что программа остановилась в момент прохождения точки останова. Использовала команды «run», «5», «*» и «backtrace» (Скриншот 3.19).

```

lab_prog: gdb — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
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) info breakpoints
No breakpoints or watchpoints.
(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 0x55555400991: file calculate.c, line 21.
(gdb) info breakpoints
Num  Type             Disp Enb Address            What
1    breakpoint      keep y   0x000055555400991 in calculate at calculate.c:21
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/t/b/tbkonovalova/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Выводимое: 4
1.00
[Inferior 1 (process 11372) exited normally]
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/t/b/tbkonovalova/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffce44 "*") at calculate.c:21
21  else if(strncmp(Operation, "*", 1)==0)
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffce44 "*") at calculate.c:21
#1 0x000055555400c31 in main () at main.c:17
(gdb)

```

Figure 3.19: Работа с gdb - run

Посмотрела, чему равно на этом этапе значение переменной Numeral, введя команду «print Numeral» (Скриншот 3.20).


```
lab_prog: gdb — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
25     return(Numeral*SecondNumeral);
26     }
27     else if(strncmp(Operation,"/",1)==0)
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb) list calculate.c:20,27
20     }
21     else if(strncmp(Operation,"*",1)==0)
22     {
23         printf("Умножение: ");
24         scanf("%f",&SecondNumeral);
25         return(Numeral*SecondNumeral);
26     }
27     else if(strncmp(Operation,"/",1)==0)
(gdb) break 21
Breakpoint 1 at 0x555555400991: file calculate.c, line 21.
(gdb) info breakpoints
Num  Type      Disp Enb Address             What
1     breakpoint keep y  0x0000555555400991 in Calculate at calculate.c:21
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/t/b/tbkonovalova/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Выводимое: 4
1.00
[Inferior 1 (process 11372) exited normally]
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/t/b/tbkonovalova/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffce44 "*") at calculate.c:21
21     else if(strncmp(Operation,"*",1)==0)
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffce44 "*") at calculate.c:21
#1 0x0000555555400c31 in main () at main.c:17
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb)
```

Figure 3.20: Работа с gdb - print Numeral

Сравнила с результатом вывода на экран после использования команды «display Numeral». Значения совпадают (Скриншот 3.21).

```
lab_prog: gdb — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
25     return(Numeral*SecondNumeral);
26     }
27     else if(strncmp(Operation,"/",1)==0)
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb) list calculate.c:20,27
20     }
21     else if(strncmp(Operation,"*",1)==0)
22     {
23         printf("Умножение: ");
24         scanf("%f",&SecondNumeral);
25         return(Numeral*SecondNumeral);
26     }
27     else if(strncmp(Operation,"/",1)==0)
(gdb) break 21
Breakpoint 1 at 0x555555400991: file calculate.c, line 21.
(gdb) info breakpoints
Num  Type      Disp Enb Address             What
1     breakpoint keep y  0x0000555555400991 in Calculate at calculate.c:21
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/t/b/tbkonovalova/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Выводимое: 4
1.00
[Inferior 1 (process 11372) exited normally]
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/t/b/tbkonovalova/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Breakpoint 1, Calculate (Numeral=5, Operation=0x7fffffffce44 "*") at calculate.c:21
21     else if(strncmp(Operation,"*",1)==0)
(gdb) backtrace
#0 Calculate (Numeral=5, Operation=0x7fffffffce44 "*") at calculate.c:21
#1 0x0000555555400c31 in main () at main.c:17
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb)
```

Figure 3.21: Работа с gdb - display Numeral

Убрала точки останова с помощью команд «info breakpoints» и «delete 1» (алгоритм действий представлен на рис. 3.22).

```

lab_prog: gdb — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
(gdb) list calculate.c:20,27
20      }
21      else if(strcmp(Operation, "+") != 0)
22      {
23          printf("Числа некорректны: ");
24          scanf("%f", &SecondNumeral);
25          return (Numeral + SecondNumeral);
26      }
27      else if(strcmp(Operation, "/", 1) != 0)
(gdb) break 21
Breakpoint 1 at 0x55555400991: file calculate.c, line 21.
(gdb) info breakpoints
Num   Type             Disp Enb Address             What
1     breakpoint       keep y   0x000055555400991 in calculate at calculate.c:21
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/t/b/tbkonovalova/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): -
Выводимое: 4
1.00
[Inferior 1 (process 11372) exited normally]
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/t/b/tbkonovalova/work/os/lab_prog/calcul
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): *
Breakpoint 1, calculate (Numeral=5, Operation=0x7fffffffce44 "+") at calculate.c:21
21      else if(strcmp(Operation, "+") != 0)
(gdb) backtrace
#0 calculate (Numeral=5, Operation=0x7fffffffce44 "+") at calculate.c:21
#1 0x000055555400c31 in main () at main.c:17
(gdb) print Numeral
$1 = 5
(gdb) display Numeral
1: Numeral = 5
(gdb) info breakpoints
Num   Type             Disp Enb Address             What
1     breakpoint       keep y   0x000055555400991 in calculate at calculate.c:21
1     breakpoint already hit 1 time
(gdb) delete 1
(gdb)
  
```

Figure 3.22: Работа с gdb - info breakpoints

7). Далее воспользовалась командами «splint calculate.c» и «splint main.c» (алгоритм действий представлен на рис. 3.23 , 3.24). С помощью утилиты splint выяснилось, что в файлах calculate.c и main.c присутствует функция чтения scanf, возвращающая целое число (тип int), но эти числа не используются и нигде не сохраняются. Утилита вывела предупреждение о том, что в файле calculate.c происходит сравнение вещественного числа с нулем. Также возвращаемые значения (тип double) в функциях pow, sqrt, sin, cos и tan записываются в переменную типа float, что свидетельствует о потере данных.

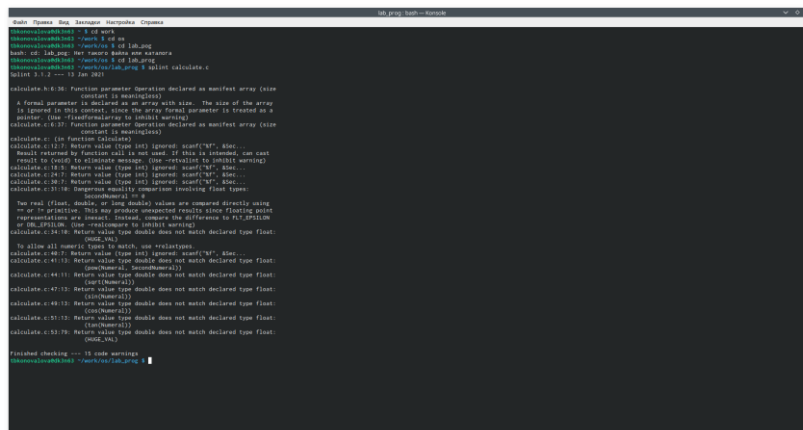


Figure 3.23: Результат команды `splint calculate.c`

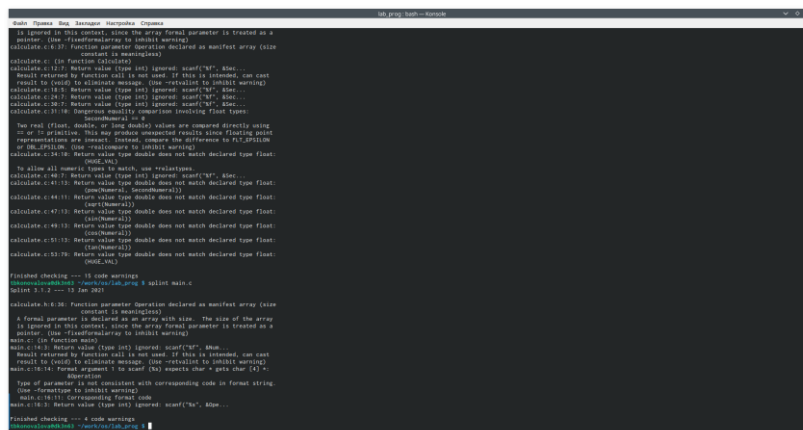


Figure 3.24: Результат команды `splint main.c`

Контрольные вопросы:

1). Чтобы получить информацию о возможностях программ gcc, make, gdbи др. нужно воспользоваться командой тапили опцией -help(-h) для каждой команды.

2). Процесс разработки программного обеспечения обычно разделяется на следующие этапы:

1. планирование, включающее сбор и анализ требований к функционалу и другим характеристикам разрабатываемого приложения;

2. проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, определение языка программирования;
3. 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;
4. 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.

3). Для имени входного файла суффикс определяет какая компиляция требуется. Суффиксы указывают на тип объекта. Файлы с расширением (суффиксом) .c воспринимаются как программы на языке C, файлы с расширением .cpp как файлы на языке C++, а файлы с расширением .o считаются объектными. Например, в команде «gcc -o main.c»: gcc по расширению (суффиксу) .c распознает тип файла для компиляции и формирует объектный модуль – файл с расширением .o. Если требуется получить исполняемый файл с определённым именем (например, hello), то требуется воспользоваться опцией -o в качестве параметра задать имя создаваемого файла: «gcc -o hello main.c». В ходе выполнения данной лабораторной работы я приобрела простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования C калькулятора с простейшими функциями.

4). Основное назначение компилятора языка Си в UNIX заключается в компиляции всей программы и получении исполняемого файла/модуля.

5). Для сборки разрабатываемого приложения и собственно компиляции полезно воспользоваться утилитой make. Она позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает

взаимосвязи между файлами.

б). Для работы с утилитой `make` необходимо в корне рабочего каталога с Вашим проектом создать файл с названием `makefile` или `Makefile`, в котором будут описаны правила обработки файлов Вашего программного комплекса. В самом простом случае `Makefile` имеет следующий синтаксис: ... : ...<команда 1>... Сначала задаётся список целей, разделённых пробелами, за которым идёт двоеточие и список зависимостей. Затем в следующих строках указываются команды. Строки с командами обязательно должны начинаться с табуляции. В качестве цели в `Makefile` может выступать имя файла или название какого-то действия. Зависимость задаёт исходные параметры (условия) для достижения указанной цели. Зависимость также может быть названием какого-то действия. Команды – собственно действия, которые необходимо выполнить для достижения цели. Общий синтаксис `Makefile` имеет вид: `target1 [target2...]:[:] [dependment1...][(tab)commands] [#commentary][(tab)commands] [#commentary]`. Здесь знак `#` определяет начало комментария (содержимое от знака `#` и до конца строки не будет обрабатываться). Одинарное двоеточие указывает на то, что последовательность команд должна содержаться в одной строке. Для переноса можно в длинной строке команд можно использовать обратный слэш `()`. Двойное двоеточие указывает на то, что последовательность команд может содержаться в нескольких последовательных строках. Пример более сложного синтаксиса `Makefile`:
`## Makefile for abcd.c`
`CC = gcc`
`CFLAGS =`
`# Compile abcd.c normally`
`abcd: abcd.c$(CC) -o abcd $(CFLAGS) abcd.o`
`clean: -rm abcd.o ~`
`# End Makefile for abcd.c`
В этом примере в начале файла заданы три переменные: `CC` и `CFLAGS`. Затем указаны цели, их зависимости и соответствующие команды. В командах происходит обращение к значениям переменных. Цель с именем `clean` производит очистку каталога от файлов, полученных в результате компиляции. Для её описания использованы регулярные выражения.

7). Во время работы над кодом программы программист неизбежно сталкивается с появлением ошибок в ней. Использование отладчика для поиска и устранения

ошибок в программе существенно облегчает жизнь программиста. В комплект программ GNU для ОС типа UNIX входит отладчик GDB (GNU Debugger). Для использования GDB необходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле. Для этого следует воспользоваться опцией -g компилятора gcc: gcc -c file.c -g. После этого для начала работы с gdb необходимо в командной строке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл: gdb file.o

8). Основные команды отладчика gdb: 1. backtrace – вывод на экран пути к текущей точке останова (по сути вывод – названий всех функций); 2. break – установить точку останова (в качестве параметра может быть указан номер строки или название функции); 3. clear – удалить все точки останова в функции; 4. continue – продолжить выполнение программы; 5. delete – удалить точку останова; 6. display – добавить выражение в список выражений, значения которых отображаются при достижении точки останова программы; 7. finish – выполнить программу до момента выхода из функции; 8. info breakpoints – вывести на экран список используемых точек останова; 9. info watchpoints – вывести на экран список используемых контрольных выражений; 10. list – вывести на экран исходный код (в ходе выполнения данной лабораторной работы я приобрела простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования C калькулятора с простейшими функциями. в качестве параметра может быть указано название файла и через двоеточие номера начальной и конечной строк); 11. next – выполнить программу пошагово, но без выполнения вызываемых в программе функций; 12. print – вывести значение указываемого в качестве параметра выражения; 13. run – запуск программы на выполнение; 14. set – установить новое значение переменной; 15. step – пошаговое выполнение программы; 16. watch – установить контрольное выражение, при изменении значения которого программа будет остановлена. Для выхода из gdb можно воспользоваться командой

quit (или её сокращённым вариантом q) или комбинацией клавиш Ctrl-d. Более подробную информацию по работе с gdb можно получить с помощью команд `gdb-hi` и `mangdb`.

9). Схема отладки программы показана в 6 пункте лабораторной работы.

10). При первом запуске компилятор не выдал никаких ошибок, но в коде программы `main.c` допущена ошибка, которую компилятор мог пропустить (возможно, из-за версии 8.3.0-19): в строке `scanf("%s", &Operation);` нужно убрать знак `&`, потому что имя массива символов уже является указателем на первый элемент этого массива.

11). Система разработки приложений UNIX предоставляет различные средства, повышающие понимание исходного кода. К ним относятся: `gscope` – исследование функций, содержащихся в программе, `gdb` – критическая проверка программ, написанных на языке Си.

12). Утилита `splint` анализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвращаемых значений, обнаруживает синтаксические и семантические ошибки. В отличие от компилятора Санализатор `splint` генерирует комментарии с описанием разбора кода программы и осуществляет общий контроль, обнаруживая такие ошибки, как одинаковые объекты, определённые в разных файлах, или объекты, чьи значения не используются в работе программы, переменные с некорректно заданными значениями и типами и многое другое.

4 Библиография

1. Программное обеспечение GNU/Linux. Лекция 9. Хранилище и дистрибутив (Г. Курячий, МГУ)
2. Программное обеспечение GNU/Linux. Лекция 10. Минимальный набор знаний (Г. Курячий, МГУ)
3. Программное обеспечение GNU/Linux. Лекция 11. udev, DBus, PolicyKit (Г. Курячий, МГУ)
4. Электронный ресурс: <https://vunivere.ru/work23597>
5. Электронный ресурс: <https://it.wikireading.ru/34160>

5 Выводы

В ходе выполнения данной лабораторной работы я приобрела простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.