

# Презентация по лабораторной работе 13

---

Елизавета Александровна Гайдамака

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

1. В домашнем каталоге создайте подкаталог `~/work/os/lab_prog`.
2. Создайте в нём файлы: `calculate.h`, `calculate.c`, `main.c`. 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.

3. Выполните компиляцию программы посредством gcc:
4. При необходимости исправьте синтаксические ошибки.
5. Создайте Makefile со следующим содержанием.
6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile).
7. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c.

1. Как получить информацию о возможностях программ gcc, make, gdb и др.?

man

2. Назовите и дайте краткую характеристику основным этапам разработки приложений в UNIX.

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

3. Что такое суффикс в контексте языка программирования?

Приведите примеры использования.

Суффикс это составная часть имени файла. Система сборки каких-либо программ (например язык java) требует, чтобы имена файлов исходного кода заканчивались на .java.

4. Каково основное назначение компилятора языка C в UNIX?

Компилировать файлы c.

5. Для чего предназначена утилита make?

make — утилита предназначенная для автоматизации преобразования файлов из одной формы в другую.

6. Приведите пример структуры Makefile. Дайте характеристику основным элементам этого файла.

Пример Makefile:

```
#  
# Makefile  
#
```

```
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)
```

7. Назовите основное свойство, присущее всем программам отладки.

Что необходимо сделать, чтобы его можно было использовать?

Возможность останавливать выполнение программы и выводить информацию о переменных. В gdb можно ставить точки останова с помощью `break`.

8. Назовите и дайте основную характеристику основным командам отладчика gdb.

- `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;



- `list` – 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 передается 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;

9. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.

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

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

- Выведите информацию об имеющихся в проекте точка останова:

```
info breakpoints
```

- Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова:

`run`

`5`

`-`

`backtrace`

– Отладчик выдаст следующую информацию:

```
#0 Calculate (Numeral=5, Operation=0x7fffffffdd280 "-")  
at calculate.c:21
```

```
#1 0x0000000000400b2b in main () at main.c:17
```

а команда `backtrace` покажет весь стек вызываемых функций от начала программы до текущего места.

- Посмотрите, чему равно на этом этапе значение переменной Numeral, введя:

```
print Numeral
```

На экран должно быть выведено число 5. - Сравните с результатом вывода на экран после использования команды:

```
display Numeral
```

- Уберите точки останова:

```
info breakpoints
```

```
delete 1
```

10. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе при его первом запуске.

У меня не было ошибок, так как я изначально исправила их в файлах.

11. Назовите основные средства, повышающие понимание исходного кода программы.

Для статической проверки программ на языке C на наличие уязвимостей в системе безопасности и типичных ошибок программирования можно использовать splint.

12. Каковы основные задачи, решаемые программой splint?

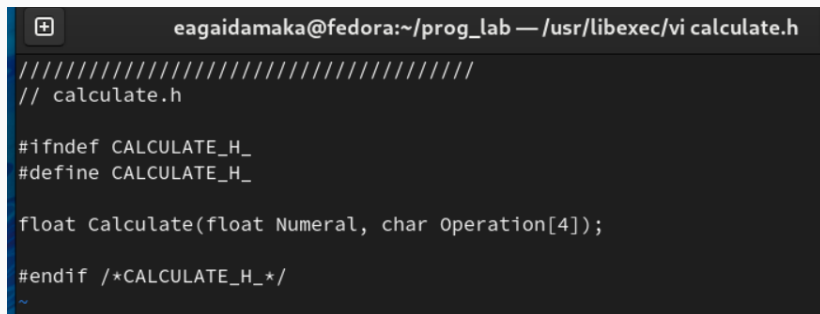
Splint — это инструмент для статической проверки программ на языке C на наличие уязвимостей в системе безопасности и типичных ошибок программирования. С минимальными усилиями Splint можно использовать в качестве лучшего lint(1). Если приложить дополнительные усилия для добавления аннотаций к программам, Splint может выполнять более строгие проверки, чем любой

# Выполнение лабораторной работы

Создаем файл `calculate.c`.

```
#####  
// 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);  
        return(Numeral * SecondNumeral);  
    }  
}
```

Создаем файл calculate.h.

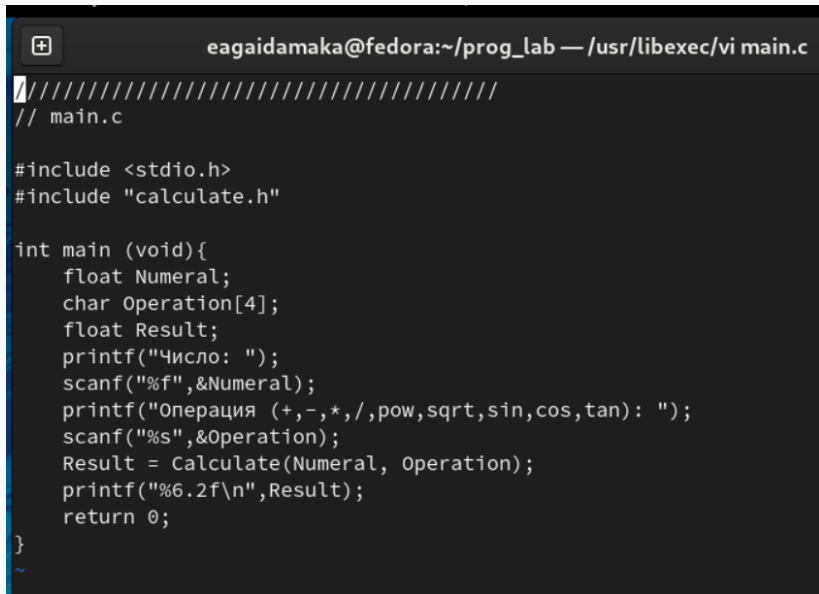


The image shows a terminal window with a dark background. The title bar at the top reads "eagaidamaka@fedora:~/prog\_lab — /usr/libexec/vi calculate.h". The terminal content shows the creation of a header file named calculate.h. It starts with a line of slashes, followed by a comment "// calculate.h". Then, it uses preprocessor directives to define the function signature: "#ifndef CALCULATE\_H\_" followed by "#define CALCULATE\_H\_". The function signature is "float Calculate(float Numeral, char Operation[4]);". It ends with "#endif /\*CALCULATE\_H\_\*/" and a tilde "~" on the next line.

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

Рис. 2: Рис.2

Создаем файл main.c.

A terminal window with a dark background. The title bar shows the user 'eagaidamaka' on a 'fedora' machine, in the directory '~/prog\_lab', editing the file 'main.c' using the 'vi' editor. The terminal content shows the creation of a new file 'main.c' with a series of slashes, followed by the C code for a calculator program. The code includes stdio.h and a custom header 'calculate.h'. The main function declares variables for a number, an operation, and a result, then prompts the user for input and performs a calculation using a 'Calculate' function.

```
eagaidamaka@fedora:~/prog_lab — /usr/libexec/vi main.c
// main.c

#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;
}
```



Выполняем компиляцию.

```
[eagaidamaka@fedora prog_lab]$ gcc -c calculate.c
[eagaidamaka@fedora prog_lab]$ gcc -c main.c
[eagaidamaka@fedora prog_lab]$ gcc calculate.o main.o -o calcul -lm
[eagaidamaka@fedora prog_lab]$ ls
calcul calculate.c calculate.h calculate.o main.c main.o Makefile
[eagaidamaka@fedora prog_lab]$
```

Рис. 4: Рис.4

Создаем Makefile.

```
[eagaidamaka@fedora prog_lab]$ cat Makefile
#
# Makefile
#

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 *~

# End Makefile[eagaidamaka@fedora prog_lab]$
```

Рис. 5: Рис.5

Проверяем работу программы с помощью gdb.



```
(gdb) run
Starting program: /home/eagaidamaka/prog_lab/calcul
Downloading separate debug info for /lib64/libm.so.6...
Downloading separate debug info for /lib64/libc.so.6...
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 3
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 5
      8.00
[Inferior 1 (process 3760) exited normally]
(gdb)
```

Рис. 6: Рис.6

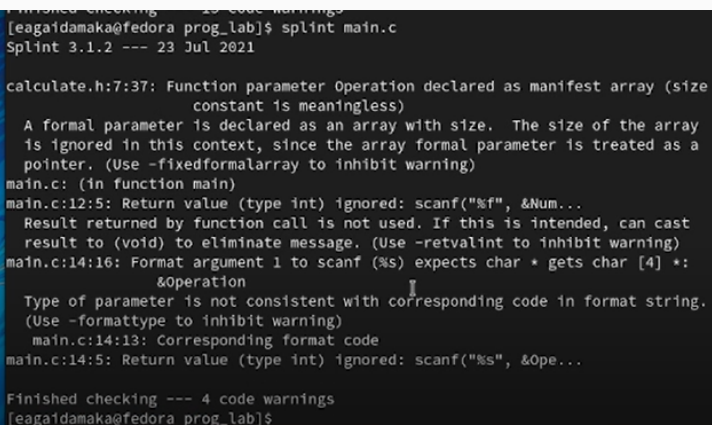
Смотрим информацию о файле `calculate.c` с помощью `splint`.

```
[eagaidamaka@fedora prog_lab]$ splint calculate.c
Splint 3.1.2 --- 23 Jul 2021

calculate.h:7:37: 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:9:37: Function parameter Operation declared as manifest array (size
        constant is meaningless)
calculate.c: (in function Calculate)
calculate.c:13:9: 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:18:9: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:23:9: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:28:9: Return value (type int) ignored: scanf("%f", &Sec...
calculate.c:29:12: Dangerous equality comparison involving float types:
```

Рис. 7: Рис.8

Смотрим информацию о файле `main.c` с помощью `splint`.



```
Finished checking --- 10 code warnings
[eagaidamaka@fedora prog_lab]$ splint main.c
Splint 3.1.2 --- 23 Jul 2021

calculate.h:7:37: 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:12:5: 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:14:16: 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:14:13: Corresponding format code
main.c:14:5: Return value (type int) ignored: scanf("%s", &Ope...

Finished checking --- 4 code warnings
[eagaidamaka@fedora prog_lab]$
```

Рис. 8: Рис.9

Благодаря данной работе 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.