

Лабораторная работа № 13

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

Бабенко Артём Сергеевич

Содержание

Теоретическое введение	3
Цель работы	4
Ход работы	5
Выводы	8
Контрольные вопросы	9

Теоретическое 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.

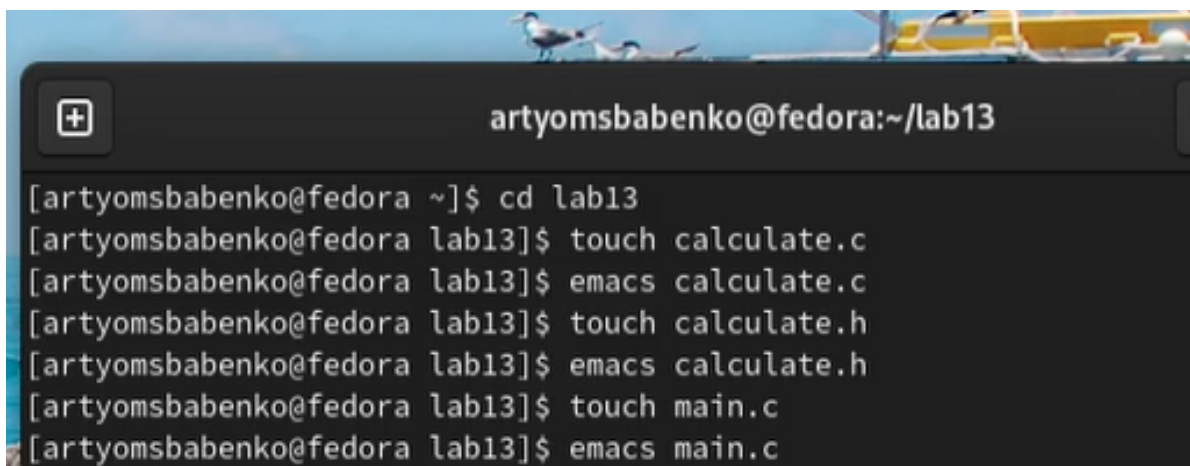
После завершения написания исходного кода программы (возможно состоящей из нескольких файлов), необходимо её скомпилировать и получить исполняемый модуль.

Цель работы

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.

Ход работы

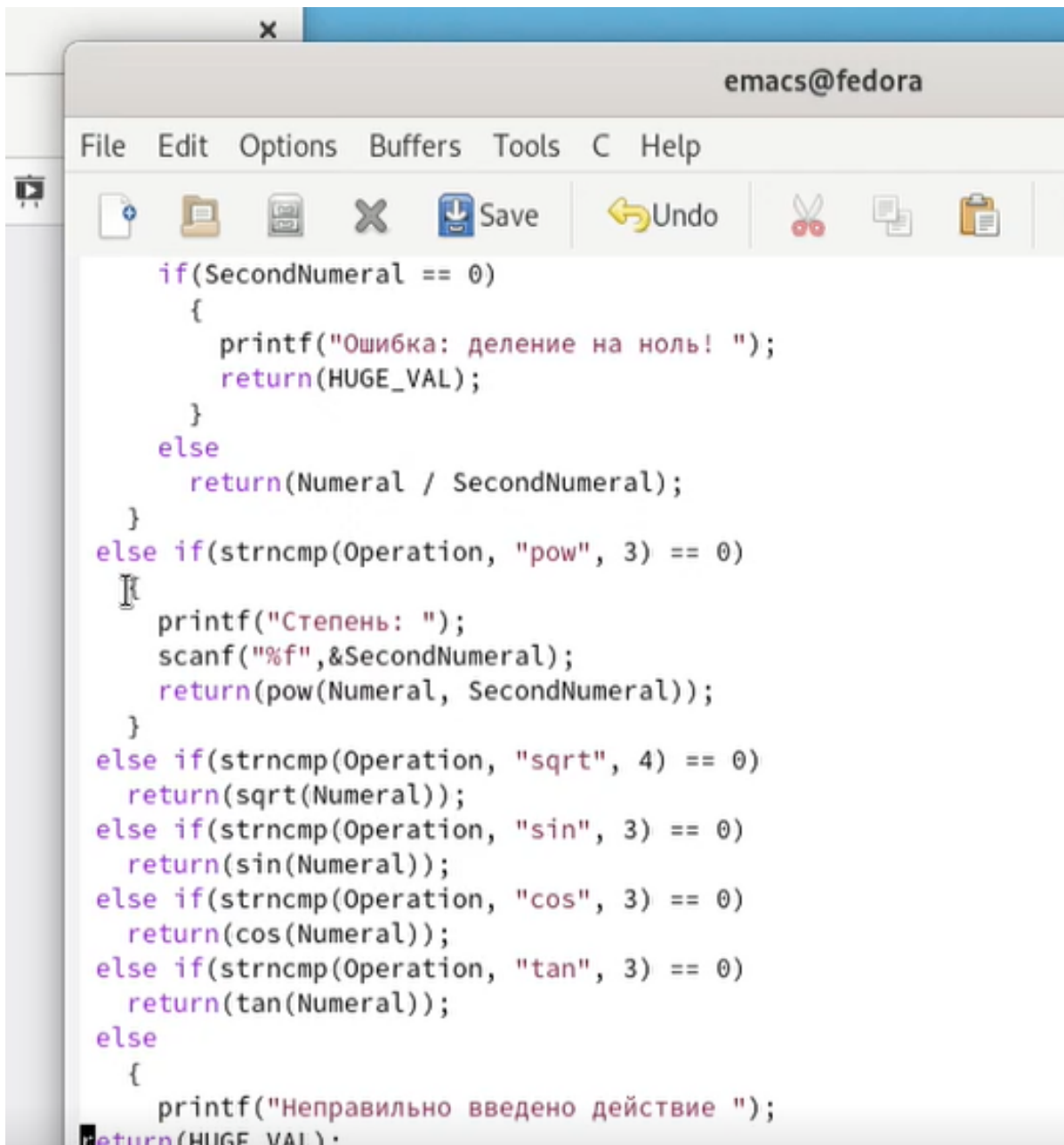
1. Создал необходимые файлы (рис.1).

A screenshot of a terminal window with a dark background and a light blue header bar. The header bar contains a plus icon in a square on the left and the text 'artyomsbabenko@fedora:~/lab13' on the right. The terminal shows a series of commands and their outputs. The commands are: 'cd lab13', 'touch calculate.c', 'emacs calculate.c', 'touch calculate.h', 'emacs calculate.h', 'touch main.c', and 'emacs main.c'. The output for each command is the same as the command itself, indicating successful execution.

```
[artyomsbabenko@fedora ~]$ cd lab13
[artyomsbabenko@fedora lab13]$ touch calculate.c
[artyomsbabenko@fedora lab13]$ emacs calculate.c
[artyomsbabenko@fedora lab13]$ touch calculate.h
[artyomsbabenko@fedora lab13]$ emacs calculate.h
[artyomsbabenko@fedora lab13]$ touch main.c
[artyomsbabenko@fedora lab13]$ emacs main.c
```

Рис.1. Создание файлов

2. Написал в файлы данный текст (рис.2).



```
if(SecondNumeral == 0)
{
    printf("Ошибка: деление на ноль! ");
    return(HUGE_VAL);
}
else
    return(Numeral / SecondNumeral);
}
else if(strncmp(Operation, "pow", 3) == 0)
{
    printf("Степень: ");
    scanf("%f",&SecondNumeral);
    return(pow(Numeral, SecondNumeral));
}
else if(strncmp(Operation, "sqrt", 4) == 0)
    return(sqrt(Numeral));
else if(strncmp(Operation, "sin", 3) == 0)
    return(sin(Numeral));
else if(strncmp(Operation, "cos", 3) == 0)
    return(cos(Numeral));
else if(strncmp(Operation, "tan", 3) == 0)
    return(tan(Numeral));
else
{
    printf("Неправильно введено действие ");
    return(HUGE_VAL);
}
```

Рис.2. Листинг

3. Выполнил компиляцию программы посредством gcc (рис.3).

```
[artyomsbabenko@fedora lab13]$ emacs main.c
[artyomsbabenko@fedora lab13]$ gcc -c calculate.c
[artyomsbabenko@fedora lab13]$ gcc -c main.c
[artyomsbabenko@fedora lab13]$ gcc calculate.o main.o -o calcul -lm
/usr/bin/ld: невозможно найти calculate.o: Нет такого файла или каталога
collect2: ошибка: выполнение ld завершилось с кодом возврата 1
[artyomsbabenko@fedora lab13]$ gcc calculate.o main.o -o calcul -lm
[artyomsbabenko@fedora lab13]$
```

Рис.3. Листинг

4. Протестировал полученный калькулятор (рис.4).

```
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 12
Операция (+,-,*,/,pow,sqrt,sin,cos,tan): +
Второе слагаемое: 2
14.00
```

Рис.4. Тест калькулятора

Выводы

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

Контрольные вопросы

1. С помощью команды `help`.
2. Это расширение.
3. 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.
4. В отличие от компилятора C анализатор `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.