

Российский университет дружбы народов

Факультет физико-математических и естественных наук

Лабораторная работа № 13. Средства, применяемые при разработке программного обеспечения в ОС типа UNIX/Linux

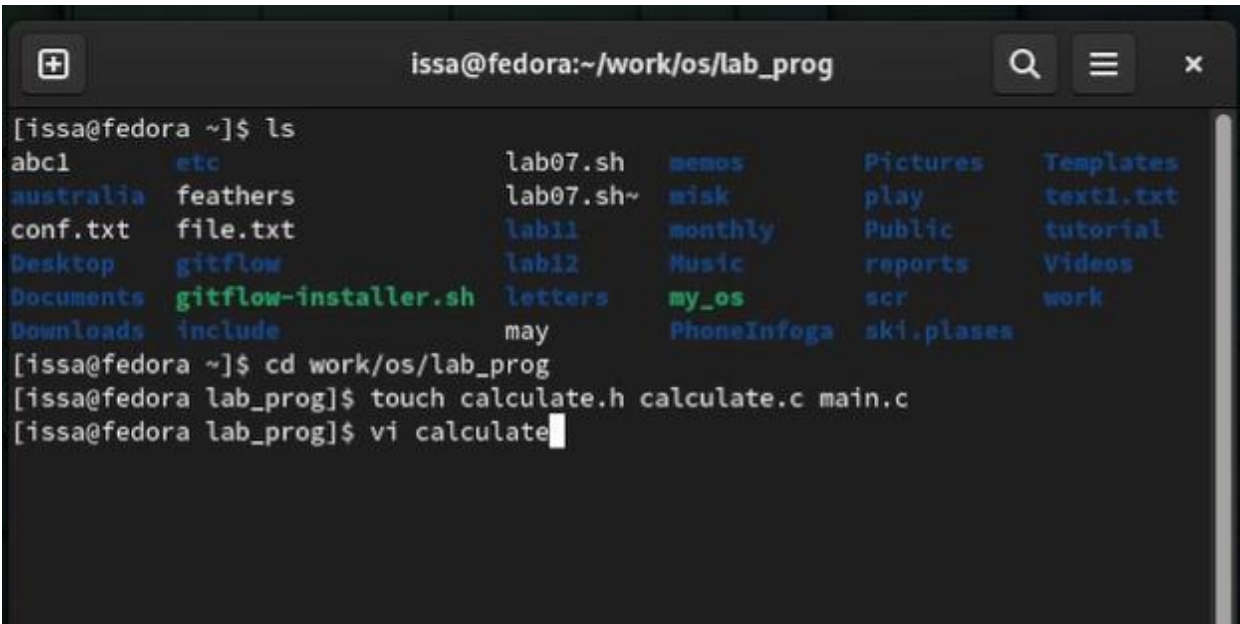
- Имя : исса гадир
- Студенческий билет : 1032218267
- Группа : нфибд-01-21

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

выполнения работы

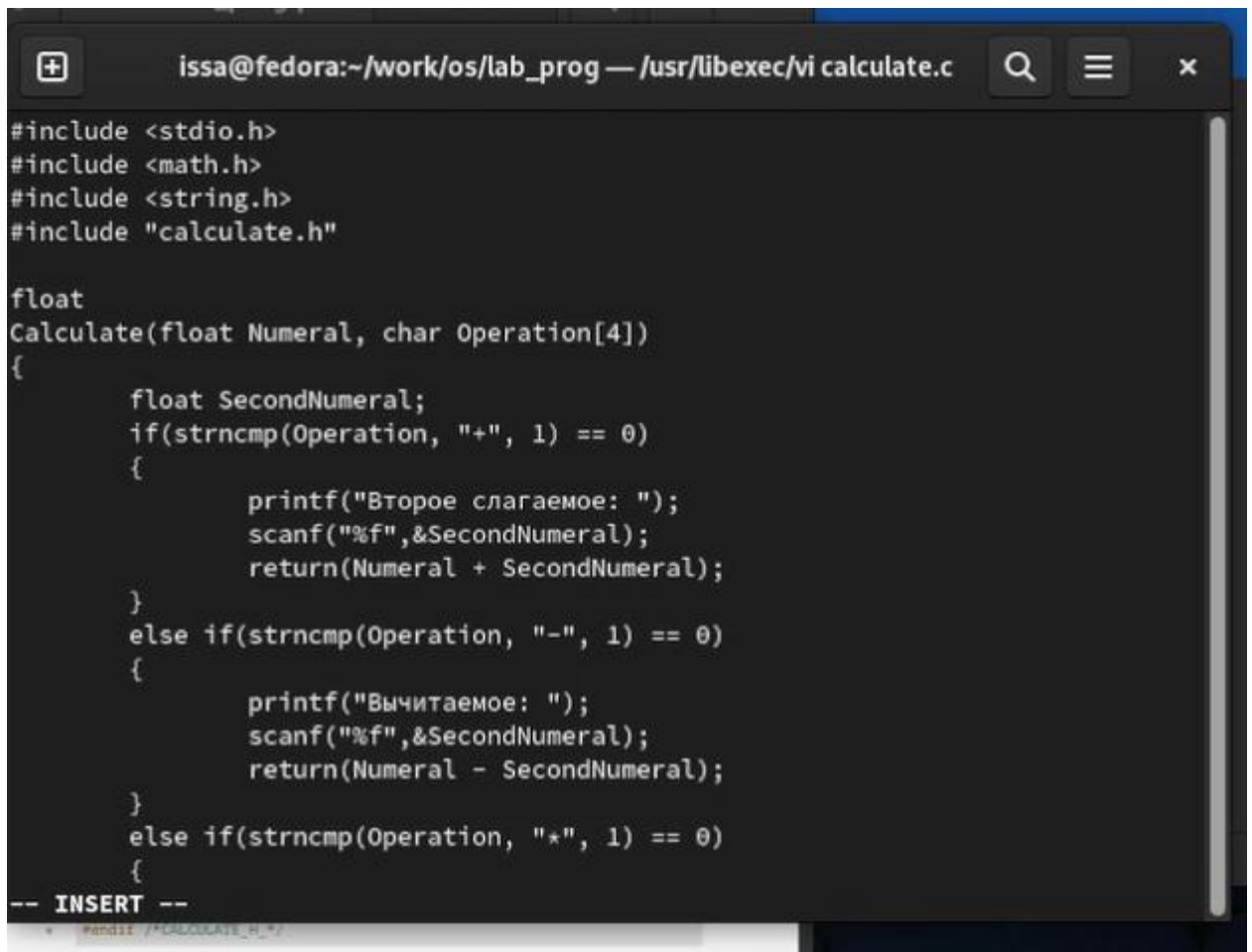
1. В домашнем каталоге создайте подкаталог `~/work/os/lab_prog`.



```
issa@fedora:~/work/os/lab_prog
[issa@fedora ~]$ ls
abcl      etc          lab07.sh  memos      Pictures  Templates
australia feathers     lab07.sh~ msk        play      text1.txt
conf.txt  file.txt    lab11     monthly    Public    tutorial
Desktop   gitflow     lab12     Music      reports   Videos
Documents gitflow-installer.sh letters    my_os     scr        work
Downloads include     may       PhoneInfoga ski.plases

[issa@fedora ~]$ cd work/os/lab_prog
[issa@fedora lab_prog]$ touch calculate.h calculate.c main.c
[issa@fedora lab_prog]$ vi calculate
```

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. Реализация функций калькулятора в файле `calculate.h`:

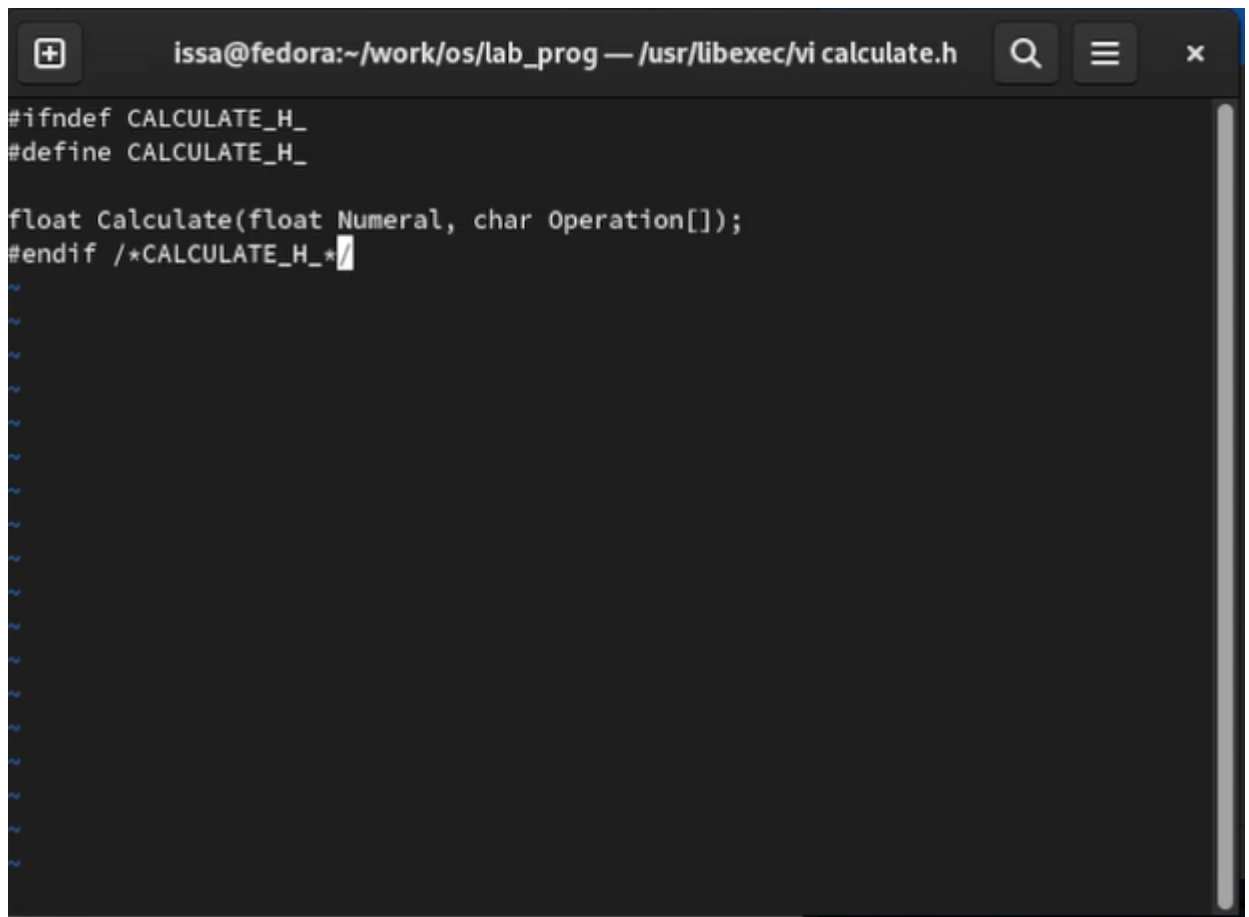


```
issa@fedora:~/work/os/lab_prog — /usr/libexec/vi calculate.c
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "calculate.h"

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("Второе множительное: ");
        scanf("%f",&SecondNumeral);
        return(Numeral * SecondNumeral);
    }
    else if(strncmp(Operation, "/", 1) == 0)
    {
        printf("Второе делительное: ");
        scanf("%f",&SecondNumeral);
        return(Numeral / SecondNumeral);
    }
    else
    {
        printf("Неизвестная операция\n");
        return(0);
    }
}

-- INSERT --
#endif /* CALCULATE_H */
```

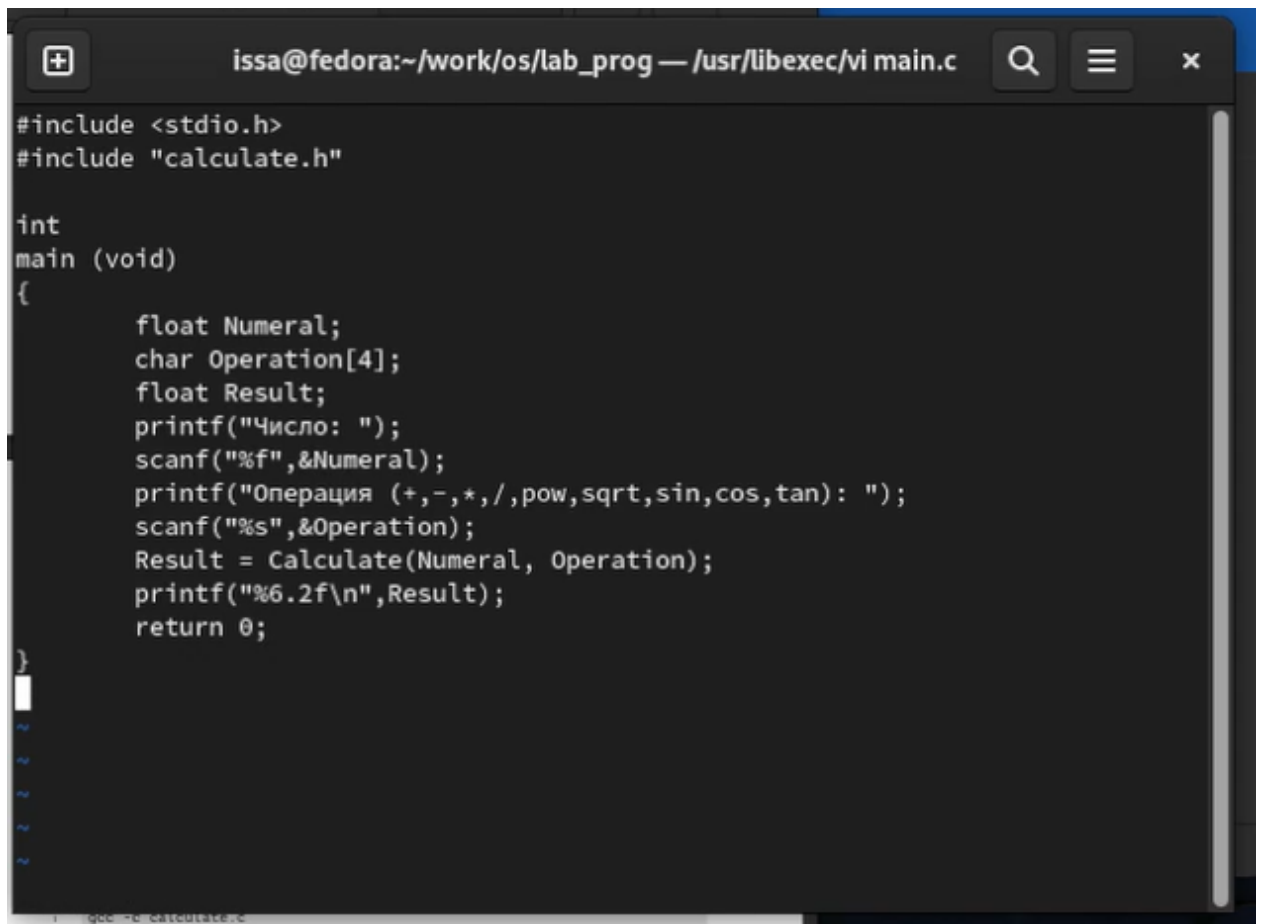
Интерфейсный файл `calculate.h`, описывающий формат вызова функции-калькулятора:

A screenshot of a terminal window with a dark background. The title bar at the top shows the user 'issa@fedora' and the current directory and file path '~ /work/os/lab_prog — /usr/libexec/vi calculate.h'. The terminal displays the content of the 'calculate.h' header file. It starts with a preprocessor guard: '#ifndef CALCULATE_H_' followed by '#define CALCULATE_H_'. Below this, there is a function declaration: 'float Calculate(float Numeral, char Operation[]);'. The file ends with '#endif /*CALCULATE_H_*/'. The rest of the terminal window is empty, showing only blue line numbers on the left margin.

```
#ifndef CALCULATE_H_
#define CALCULATE_H_

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

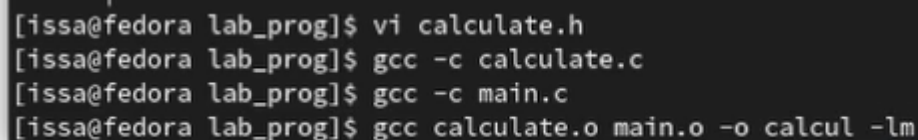
Основной файл main.c, реализующий интерфейс пользователя к калькулятору:



```
issa@fedora:~/work/os/lab_prog — /usr/libexec/vi main.c
#include <stdio.h>
#include "calculate.h"

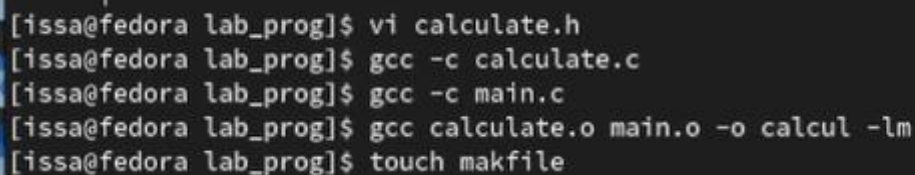
int
main (void)
{
    float Numeral;
    char Operation[4];
    float Result;
    printf("Число: ");
    scanf("%f",&Numeral);
    printf("Операция (+,-,*,/,pow,sqrt,sin,cos,tan): ");
    scanf("%s",&Operation);
    Result = Calculate(Numeral, Operation);
    printf("%.2f\n",Result);
    return 0;
}
```

3. Выполните компиляцию программы посредством gcc:



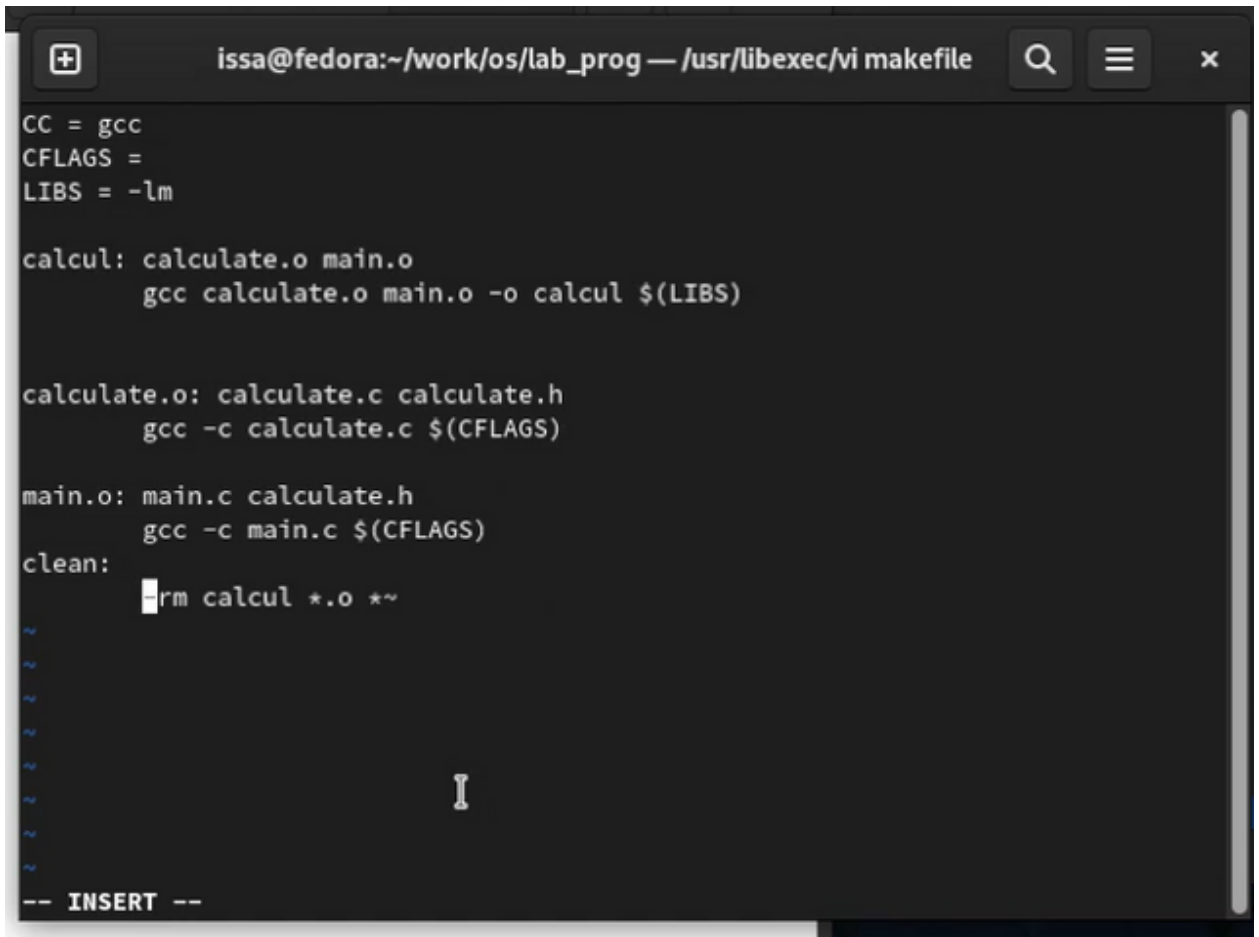
```
[issa@fedora lab_prog]$ vi calculate.h
[issa@fedora lab_prog]$ gcc -c calculate.c
[issa@fedora lab_prog]$ gcc -c main.c
[issa@fedora lab_prog]$ gcc calculate.o main.o -o calcul -lm
```

4. При необходимости исправьте синтаксические ошибки.



```
[issa@fedora lab_prog]$ vi calculate.h
[issa@fedora lab_prog]$ gcc -c calculate.c
[issa@fedora lab_prog]$ gcc -c main.c
[issa@fedora lab_prog]$ gcc calculate.o main.o -o calcul -lm
[issa@fedora lab_prog]$ touch makfile
```

5. Создайте Makefile со следующим содержанием:

A screenshot of a terminal window with a dark background. The title bar at the top shows the user 'issa' on a 'fedora' machine, in the directory '~/work/os/lab_prog', editing the file '/usr/libexec/vi makefile'. The terminal content shows a Makefile with variables CC=gcc, CFLAGS=, and LIBS=-lm. It defines targets 'calcul' (linking calculate.o and main.o), 'calculate.o' (compiling calculate.c), and 'main.o' (compiling main.c). A 'clean' target is also defined to remove the executable and object files. The editor is in 'INSERT' mode, indicated by the prompt at the bottom.

```
issa@fedora:~/work/os/lab_prog — /usr/libexec/vi 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 *~

-- INSERT --
```

6. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile): – Запустите отладчик GDB, загрузив в него программу для отладки

```
issa@fedora:~/work/os/lab_prog — gdb ./calcul
[issa@fedora ~]$ cd work/os/lab_prog
[issa@fedora lab_prog]$ ls
calcul calculate.c calculate.h calculate.o main.c main.o makefile
[issa@fedora lab_prog]$ gdb ./calcul
GNU gdb (GDB) Fedora 12.1-1.fc36
Copyright (C) 2022 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"...
Reading symbols from ./calcul...

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n])
```

```
<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...

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
(No debugging symbols found in ./calcul)
(gdb) run
Starting program: /home/issa/work/os/lab_prog/calcul
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Число: 5
Операция (+,-,*,/,pow,sqrt,sin,cos,tan):
-
Вычитаемое: backtrace
5.00
[Inferior 1 (process 3698) exited normally]
Missing separate debuginfos, use: dnf debuginfo-install glibc-2.35-11.fc36.x86_64
(gdb)
```

7. С помощью утилиты `splint` попробуйте проанализировать коды файлов `calculate.c` и `main.c`.

```
[issa@fedora ~]$ cd work/os/lab_prog
[issa@fedora lab_prog]$ ls
calcul calculate.c calculate.h calculate.o main.c main.o makefile
[issa@fedora lab_prog]$ split calculate.c
[issa@fedora lab_prog]$ split main.c
[issa@fedora lab_prog]$
```

Контрольные вопросы

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

--help

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

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. Что такое суффикс в контексте языка программирования? Приведите примеры использования.

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. Каково основное назначение компилятора языка C в UNIX?

Скомпилирует файл

5. Для чего предназначена утилита make?

Получите файл выполнения

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

`Calcul: calculate.o main.o Gcc calculate.o main.o -o`

элементам этого файла. 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. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?

'Main' это один из значений его атрибута

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; – `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; – 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.

Backtrace: вывод на экран пути к текущей точке останова Break: установить точку останова Clear: удалить все точки останова

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

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

Программисты могут использовать текстовый редактор, инструмент визуального программирования или интегрированную среду разработки (IDE), такую как комплект разработки программного обеспечения (SDK), для создания исходного кода. В больших средах разработки программ часто используются системы управления, которые помогают программистам отделять и отслеживать различные состояния и уровни файлов исходного кода.

12. Каковы основные задачи, решаемые программой splint?

Splint особенно хорош для проверки типов назначений переменных и функций, эффективности, неиспользуемых переменных и идентификаторов функций, недостижимого кода и возможных утечек памяти. Существует множество полезных опций, помогающих контролировать шину (см. Шина человека).

