Лабораторная работа №13: отчет.

Элементы криптографии. Шифрование (кодирование) различных исходных текстов одним ключом.

Евдокимов Максим Михайлович. Группа - НФИбд-01-20.

Содержание

# 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.

# Задание

1. Написать по промеру базовый калькулятор.
2. Изучить примеры создания функций на языке програмирования Си.
3. Изучить среду gdb.

# Указание к работе

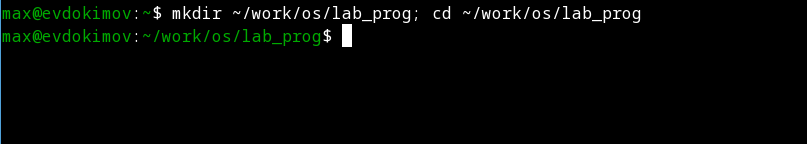
## Тестирование и отладка

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: “1 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: 1 “gdb file.o”. Затем можно использовать по мере необходимости различные команды gdb. Наиболее часто используемые команды gdb приведены в табл. 13.2. 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 -h и man gdb.

# Выполнение лабораторной работы

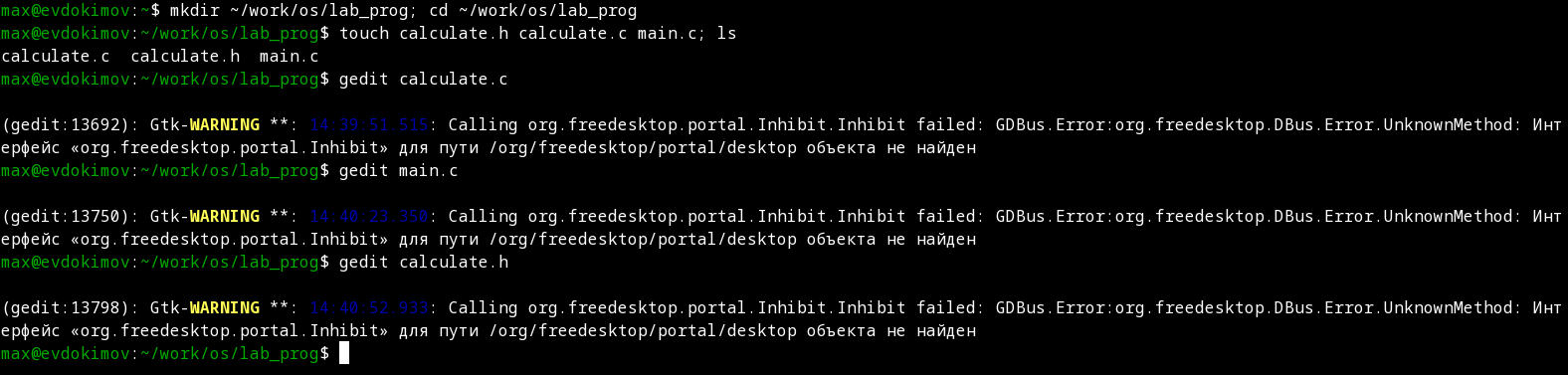
## Калькулятор

1. В домашнем каталоге создайте подкаталог ~/work/os/lab\_prog.



Создание директории

1. 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.



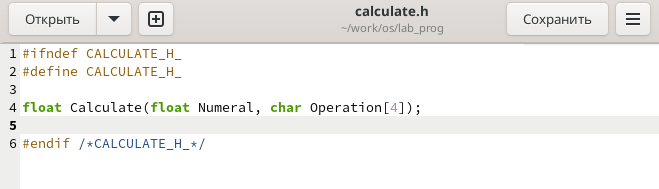
Создание файлов



Файл calculate.c

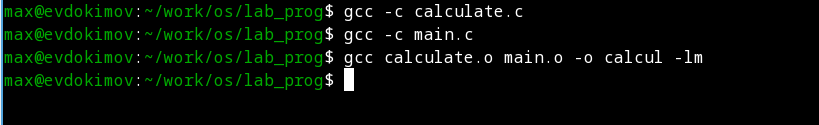


Файл main.c



Файл calculate.h

1. Выполните компиляцию программы посредством gcc:

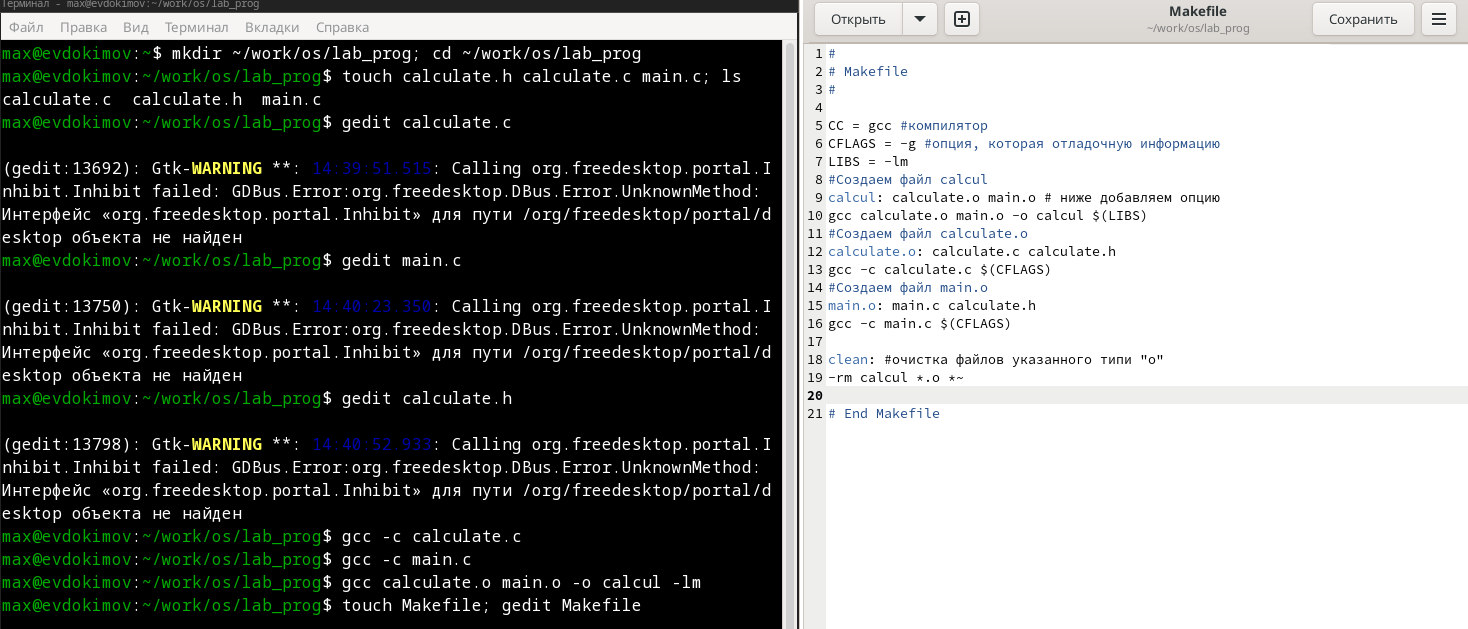


Выполнение кодов

1. При необходимости исправьте синтаксические ошибки:

(нет необходимости)

1. Создайте Makefile со следующим содержанием:



Makefile

## Коды

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);  
}  
else if(strncmp(Operation, "/", 1) == 0)  
{  
printf("Делитель: ");  
scanf("%f",&SecondNumeral);  
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);  
}  
}

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;  
}  
  
calculate.h:  
  
``` C  
#ifndef CALCULATE\_H\_  
#define CALCULATE\_H\_  
  
float Calculate(float Numeral, char Operation[4]);  
  
#endif /\*CALCULATE\_H\_\*/

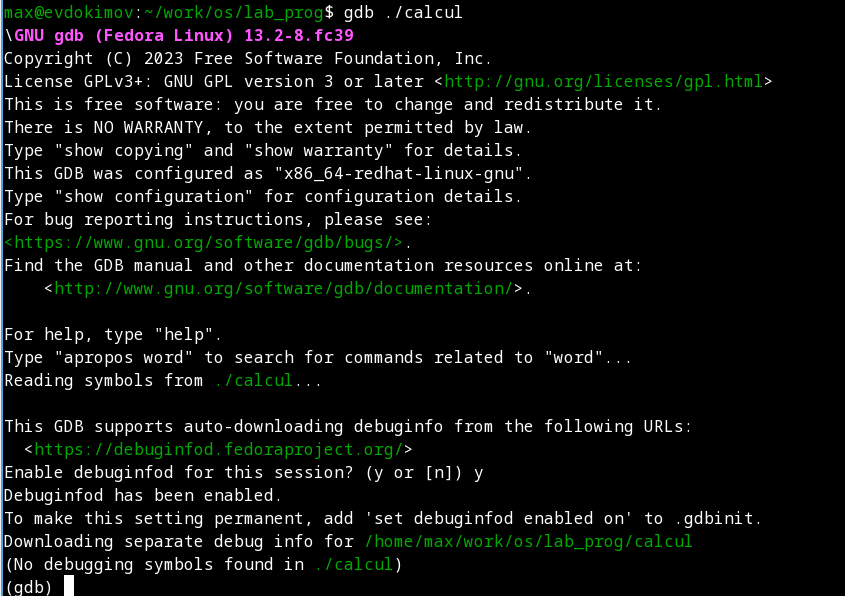
Makefile:

#  
# Makefile  
#  
  
CC = gcc #компилятор  
CFLAGS = -g #опция, которая отладочную информацию  
LIBS = -lm  
#Создаем файл calcul  
calcul: calculate.o main.o # ниже добавляем опцию  
gcc calculate.o main.o -o calcul $(LIBS)  
#Создаем файл calculate.o  
calculate.o: calculate.c calculate.h  
gcc -c calculate.c $(CFLAGS)  
#Создаем файл main.o  
main.o: main.c calculate.h  
gcc -c main.c $(CFLAGS)  
  
clean: #очистка файлов указанного типи "o"  
-rm calcul \*.o \*~  
  
# End Makefile

## Работа с gdb

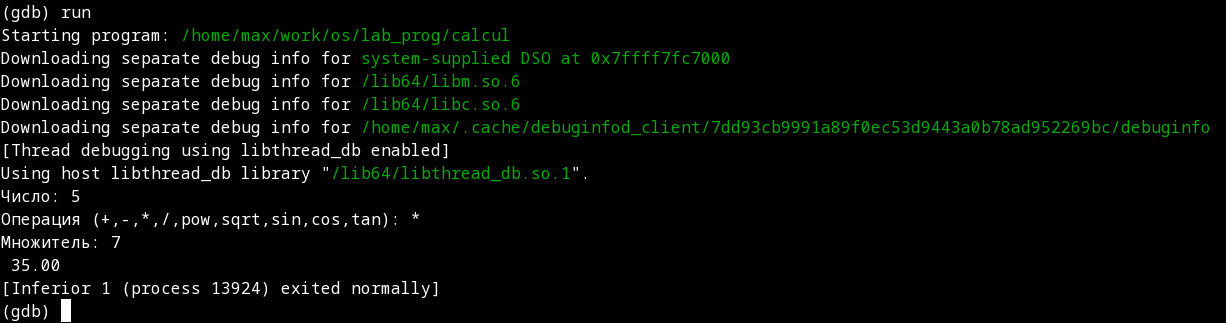
1. С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile):

– Запустите отладчик GDB, загрузив в него программу для отладки “gdb ./calcul”:



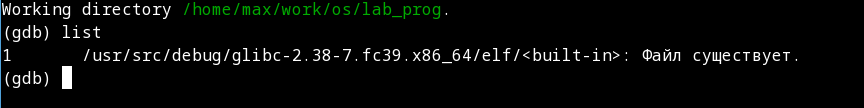
Запуск gdb

– Для запуска программы внутри отладчика введите команду “run”:



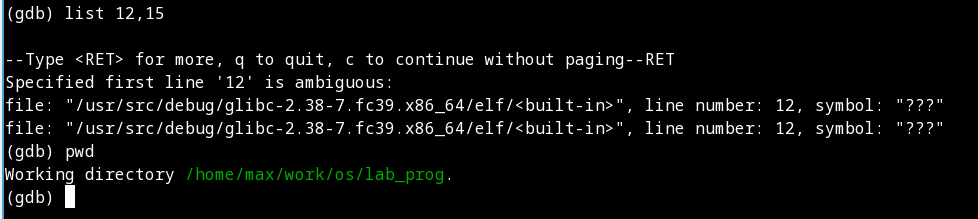
Run кода

– Для постраничного (по 9 строк) просмотра исходного код используйте команду “list”:



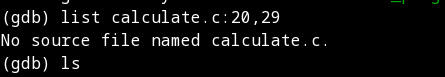
Команда list

– Для просмотра строк с 12 по 15 основного файла используйте list с параметрами “list 12,15”:



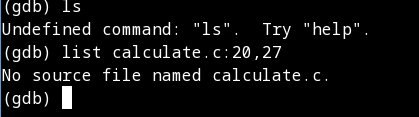
Команда list c указанием

– Для просмотра определённых строк не основного файла используйте list с параметрами “list calculate.c:20,29”:

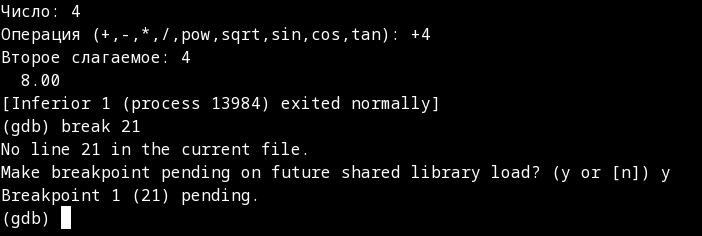


Команда list по файлу

– Установите точку останова в файле calculate.c на строке номер 21 “list calculate.c:20,27” и “break 21”:

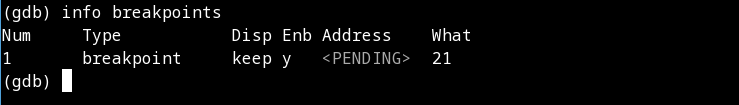


Команда list по файлу 2



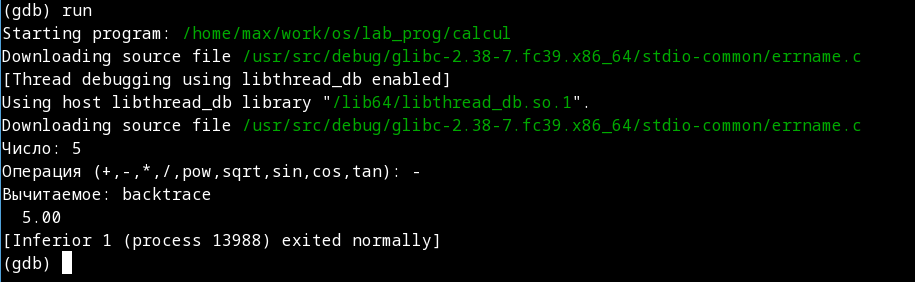
Устоновка точки остановы

– Выведите информацию об имеющихся в проекте точка останова “info breakpoints”:



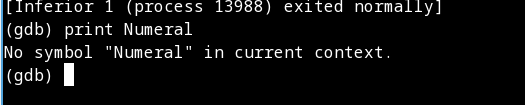
Просмотр breakpoints

– Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения точки останова “run” “5” “-” “backtrace”:



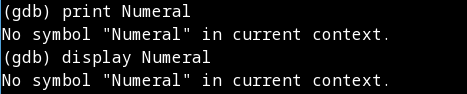
Запуск по условию

– Посмотрите, чему равно на этом этапе значение переменной Numeral, введя “print Numeral”, На экран должно быть выведено число 5:



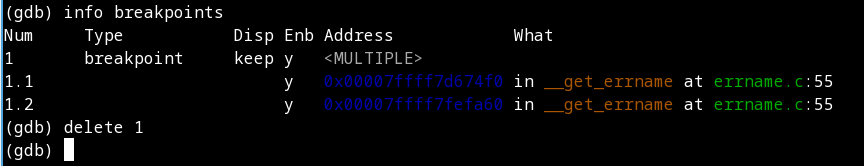
Вывод значения

– Сравните с результатом вывода на экран после использования команды “display Numeral”:



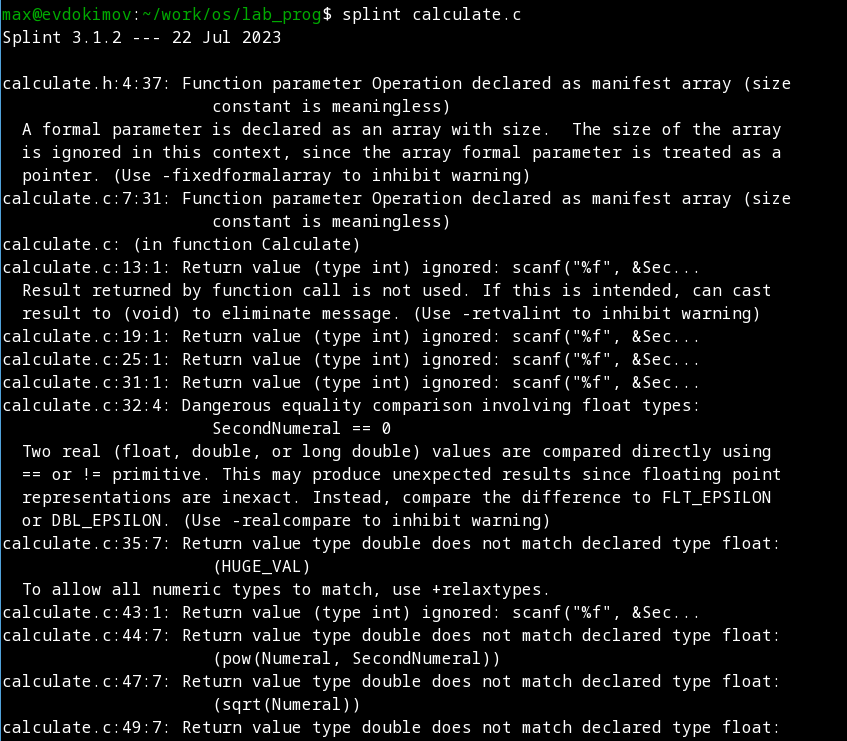
Показ имеющегося в системе значения

– Уберите точки останова “info breakpoints” и “delete 1”:



Удаление точки остановы

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



Команда splint по calculate



Команда splint по main

# Контрольные вопросы

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

Дополнительную информацию об этих программах можно получить с помощью функций info и man.

1. 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;
* компиляция исходного текста и построение исполняемого модуля;
* тестирование и отладка;
* проверка кода на наличие ошибок;
* сохранение всех изменений, выполняемых при тестировании и отладке.

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.

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

Использование суффикса “.с” 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.о” 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 и новых 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”.

1. Каково основное назначение компилятора языка С в UNIX?

Основное назначение компилятора с языка Си заключается в компиляции всей программы в целом и получении исполняемого модуля.

1. Для чего предназначена утилита make?

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.

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

makefile для некой программы расщирения “.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 OC UNIX, которые необходимо выполнить для обновления целевого файла.

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

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 отлаженную 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.

1. 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;
* 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. Опишите по шагам схему отладки программы, которую Вы использовали при выполнении лабораторной работы.

* Выполнили компиляцию программы;
* Увидели ошибки в программе;
* Открыли редактор и исправили программу;
* Загрузили программу в отладчик gdb run — отладчик выполнил программу, ввели требуемые значения;
* Программа завершена, gdb не видит ошибок.

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

Отладчику не понравился формат “%s” для “&Operation”, так как “%s” — символьный формат, а значит необходим только Operation.

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

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. Каковы основные задачи, решаемые программой splint?

Проверка корректности задания аргументов всех исполняемых функций , а также типов возвращаемых ими значений; Поиск фрагментов исходного текста, корректных с точки зрения синтаксиса языка Си, но малоэффективных с точки зрения их реализации или содержащих в себе семантические ошибки; Общая оценка мобильности пользовательской программы.

# Выводы

В ходе выполнения лабораторной работы были изучены новые способы написания программ на языке C, а так же ознокомился с системой gdb.

# Список литературы

1. [Лабораторная работа №13](https://esystem.rudn.ru/mod/resource/view.php?id=970842)