**Лабораторная работа 4 Инструментальные средства разработки Linux**

**Цель**: изучить инструментальные средства разработки Linux: компиляторы сс/g++/gcc

**Теория**:

Средствами, традиционно используемыми для создания программ для открытых операционных систем, являются инструменты разработчика GNU. Целью проекта GNU было создание комплекта программного обеспечения под единой лицензией, которая не допускала бы возможности присваивания кем-то эксклюзивных прав на это ПО. Частью этого комплекта и является набор инструментов для разработчика, которым мы будем пользоваться, и который должен входить во все дистрибутивы Linux.

Одним из этих инструментов является компилятор GCC. Первоначально эта аббревиатура расшифровывалась, как GNU C Compiler. Сейчас она означает – GNU Compiler Collection и поддерживает множество языков программирования (С, C++, Objective C++, Java, Fortran и др.).

Файлы с исходными кодами программ, которые мы будем создавать, это обычные текстовые файлы, и создавать их можно с помощью любого текстового редактора (например GEdit, KWrite, Kate, а также более традиционные для пользователей Linux – vi и emacs).

Обычно для создания запускаемого файла создается отдельный каталог с исходным кодом, в котором и будет находится исполняемый файл.

Создадим отдельный в домашнем каталоге каталог hello. Это будет каталог нашего первого проекта. Создадим текстовый файл с исходным кодом hello.c со следующим текстом:

#include <stdio.h>

int main(void){

printf("Hello world!\n");

return(0);

}

Теперь в консоли необходимо зайти в каталог проекта и набрать

gcc hello.c

В каталоге должен появиться новый файл a.out. Это и есть исполняемый файл. Запустим его, набрав в консоли:

./a.out

Программа должна запуститься, то есть должен появиться текст:

Hello world!

Компилятор gcc по умолчанию присваивает всем созданным исполняемым файлам имя a.out. Если хотите назвать его по-другому, нужно к команде на компиляцию добавить флаг –o и имя, которым вы хотите его назвать. Наберите такую команду:

gcc hello.c -o hello

В каталоге должен появиться исполняемый файл с названием hello. Запустите его.

./hello

Флаг -o является лишь одним из многочисленных флагов компилятора gcc. Чтобы просмотреть все возможные флаги, можно воспользоваться справочной системой man. Наберите в командной строке:

man gcc

Перед вами предстанет справочная система по этой программе. Просмотрите, что означает каждый флаг.

При запуске программ из нашего каталога разработки, мы перед названием файла набираем точку и слэш. Зачем же мы это нужно? Дело в том, что, если мы наберем только название исполняемого файла, операционная система будет искать его в каталогах /usr/bin и /usr/local/bin, и, естественно, не найдет. Каталоги /usr/bin и /usr/local/bin – системные каталоги размещения исполняемых программ. Первый из них предназначен для размещения стабильных версий программ, как правило, входящих в дистрибутив Linux. Второй – для программ, устанавливаемых самим пользователем (за стабильность которых никто не ручается). Такая система нужна, чтобы отделить их друг от друга. По умолчанию при сборке программы устанавливаются в каталог /usr/local/bin. Крайне нежелательно помещать что-либо лишнее в /usr/bin или удалять что-то оттуда вручную, потому что это может привести к краху системы. Там должны размещаться программы, за стабильность которых отвечают разработчики дистрибутива.

Чтобы запустить программу, находящуюся в другом месте, надо прописать полный путь к ней, например так:

/home/dima/projects/hello/hello

Или другой вариант: прописать путь относительно текущего каталога, в котором вы в данной момент находитесь в консоли. При этом одна точка означает текущий каталог, две точки – родительский. Например, команда ./hello запускает программу hello, находящуюся в текущем каталоге, команда ../hello – программу hello, находящуюся в родительском каталоге, команда ./projects/hello/hello – программу во вложенных каталогах, находящихся внутри текущего.

Есть возможность добавлять в список системных путей к программам дополнительные каталоги. Для этого надо добавить новый путь в системную переменную PATH.

Рассмотрим принцип работы программы gcc, он включает три этапа: обработка препроцессором, компиляция и компоновка (или линковка).

Препроцессор включает в основной файл содержимое всех заголовочных файлов, указанных в директивах #include. В заголовочных файлах обычно находятся объявления функций, используемых в программе, но не определенных в тексте программы. Их определения находятся где-то в другом месте: или в других файлах с исходным кодом или в бинарных библиотеках (ключ -E).

Вторая стадия – компиляция. Она заключается в превращении текста программы на языке C/C++ в набор машинных команд. Результат сохраняется в объектном файле.

Разумеется, на машинах с разной архитектурой процессора двоичные файлы получаются в разных форматах, и на одной машине невозможно запустить бинарник, собранный на другой машине (разве только, если у них одинаковая архитектура процессора и одинаковые операционные системы). Вот почему программы для UNIX-подобных систем распространяются в виде исходных кодов: они должны быть доступны всем пользователям, независимо от того, у кого какой процессор и какая операционная система (ключ -с).

Последняя стадия – компоновка. Она заключается в связывании всех объектных файлов проекта в один, связывании вызовов функций с их определениями, и присоединением библиотечных файлов, содержащих функции, которые вызываются, но не определены в проекте. В результате формируется запускаемый файл – наша конечная цель. Если какая-то функция в программе используется, но компоновщик не найдет место, где эта функция определена, он выдаст сообщение об ошибке, и откажется создавать исполняемый файл (ключ -o).

Если создается объектный файл из исходника, уже обработанного препроцессором (например, такого, какой мы получили выше), то мы должны обязательно указать явно, что компилируемый файл является файлом исходного кода, обработанный препроцессором, и имеющий теги препроцессора. В противном случае он будет обрабатываться, как обычный файл C++, без учета тегов препроцессора, а значит связь с объявленными функциями не будет устанавливаться. Для явного указания на язык и формат обрабатываемого файла служит опция -x. Файл C++, обработанный препроцессором обозначается cpp-output

.

gcc -x cpp -output -c prog1.cpp

Наконец, последний этап – компоновка. Получаем из объектного файла исполняемый.

gcc prog1.o -o proga

Можно его запускать.

./ prog1

Как правило исходных файлов несколько, и они объединены в проект. И в некоторых исключительных случаях программу приходится компоновать из нескольких частей, написанных на разных языка. В этом случае приходится запускать компиляторы разных языков, чтобы каждый получил объектный файл из своего исходника, а затем уже эти полученные объектные файлы компоновать в исполняемую программу.

**Пример проекта из нескольких файлов**

Напишем теперь программу, состоящую из двух исходных файлов и одного заголовочного. Для этого создадим программу калькулятор, в которой после введения первого числа надо сразу вводить действие. Если действие оперирует только с одним числом (как в случае синуса, косинуса, тангенса, квадратного корня), результат сразу будет выведен. Если понадобится второе число, оно будет специально запрашиваться.

Создадим каталог проекта kalkul. В нем создадим три файла: calculate.h, calculate.c, main.c.

Файл calculate.h:

///////////////////////////////////////

// calculate.h

#ifndef CALCULATE\_H\_

#define CALCULATE\_H\_

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

#endif /\*CALCULATE\_H\_\*/

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);

}

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);

}

}

Файл 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("Арифметическое действие (+,-,\*,/,pow,sqrt,sin,cos,tan): ");

scanf("%s",&Operation);

Result = Calculate(Numeral, Operation);

printf("%6.2f\n",Result);

return 0;

}

Таким образом у нас есть два файла исходного кода (c-файлы) и один заголовочный (h-файл). Заголовочный файл включается в оба c-файла.

Компилируем calculate.c.

gcc -c calculate.c

Получили calculate.o. Затем компилируем main.c.

gcc -c main.c

Теперь, надо из этих двух объектных файлов сделать запускаемый.

gcc calculate.o main.o -o kalkul

Если выполнить все эти действия, вместо столь желаемого запускаемого файла, в консоли будут выведены ошибки:

calculate.o(.text+0x1b5): In function `Calculate':

calculate.c: undefined reference to `pow'

calculate.o(.text+0x21e):calculate.c: undefined reference to

`sqrt'

calculate.o(.text+0x274):calculate.c: undefined reference to `sin'

calculate.o(.text+0x2c4):calculate.c: undefined reference to `cos'

calculate.o(.text+0x311):calculate.c: undefined reference to `tan'

collect2: ld returned 1 exit status

Undefined reference означает ссылку на функцию, которая не определена. В данном случае компилятор не смог найти определения функций pow, sqrt, sin, cos, tan. Как уже говорилось раньше, определения функций могут находиться в библиотеках. Это скомпилированные двоичные файлы, содержащие коллекции однотипных операций, которые часто вызываются из многих программ, а потому нет смысла многократно писать их код в программах. Стандартное расположение файлов библиотек – каталоги /usr/lib и /usr/local/lib (при желании можно добавить путь). Если библиотечный файл имеет расширение .a, то это статическая библиотека, то есть при компоновке весь ее двоичный код включается в исполняемый файл. Если расширение .so, то это динамическая библиотека. Это значит в исполняемый файл программы помещается только ссылка на библиотечный файл, а уже из него и запускается функция.

Когда мы писали программу hello, мы использовали функцию printf для вывода текстовой строки. Однако, мы нигде не писали определения этой функции. Откуда же она тогда вызывается? Просто при компоновке любой программы компилятор gcc по умолчанию включает в запускаемый файл библиотеку libc. Это стандартная библиотека языка C. Она содержит функции, необходимые абсолютно во всех программах, написанных на C, в том числе и функцию printf. Поскольку библиотека libc нужна во всех программах, она включается по умолчанию, без необходимости давать отдельное указание на ее включение.

Остальные библиотеки надо включать явно. Для нашей программы необходима библиотека libm. Именно она содержит все основные математические функции. Она требует включения в текст программы заголовочного файла <math.h>.

Помимо этого дистрибутивы Linux содержат и другие библиотеки, например:

● libGL Вывод трехмерной графики в стандарте OpenGL. Требуется заголовочный файл <GL/gl.h>.

● libcrypt Криптографические функции. Требуется заголовочный файл <crypt.h>.

● libcurses Псевдографика в символьном режиме. Требуется заголовочный файл <curses.h>.

● libform Создание экранных форм в текстовом режиме. Требуется заголовочный файл <form.h>.

● libgthread Поддержка многопоточного режима. Требуется заголовочный файл <glib.h>.

● libgtk Графическая библиотека в режиме X Window. Требуется заголовочный файл <gtk/gtk.h>.

● libhistory Работы с журналами. Требуется заголовочный файл <readline/readline.h>.

● libjpeg Работа с изображениям в формате JPEG. Требуется заголовочный файл <jpeglib.h>.

● libncurses Работа с псевдографикой в символьном режиме. Требуется заголовочный файл <ncurses.h>.

● libpng Работа с графикой в формате PNG. Требуется заголовочный файл <png.h>.

● libpthread Многопоточная библиотека POSIX. Стандартная многопоточная библиотека для Linux. Требуется заголовочный файл <pthread.h>.

● libreadline Работа с командной строкой. Требуется заголовочный файл <readline/readline.h>.

● libtiff Работа с графикой в формате TIFF. Требуется заголовочный файл <tiffio.h>.

● libvga Низкоуровневая работа с VGA и SVGA. Требуется заголовочный файл <vga.h>.

А также многие-многие другие.

Обратите внимание, что названия всех этих библиотек начинаются с буквосочетания lib-. Для их явного включения в исполняемый файл, нужно добавить к команде gcc опцию -l, к которой слитно прибавить название библиотеки без lib-. Например, чтобы включить библиотеку libvga надо указать опцию -lvga.

Нам нужны математические функции pow, sqrt, sin, cos, tan. Они, как уже было сказано, находятся в математической библиотеке libm. Следовательно, чтобы подключить эту библиотеку, мы должны указать опцию -lm.

gcc calculate.o main.o -o kalkul -lm

Итак запускаемый файл создан!

Практическая часть:

1. В соответствии со своим вариантом разработать программу вычисления значения функции **b=f(x,y,z).** Значения x, y и z должны вводиться пользователем. При выводе информации предусмотреть форматирование документа. Описание функции и ее реализацию представить в отдельных файлах.

| Вариант | Вид функции | Вариант | Вид функции |
| --- | --- | --- | --- |
| 1 |  | 16 |  |
| 2 |  | 17 |  |
| 3 |  | 18 |  |
| 4 |  | 19 |  |
| 5 |  | 20 |  |
| 6 |  | 21 |  |
| 7 |  | 22 |  |
| 8 |  | 23 |  |
| 9 |  | 24 |  |
| 10 |  | 25 |  |
| 11 |  | 26 |  |
| 12 |  | 27 |  |
| 13 |  | 28 |  |
| 14 |  | 29 |  |
| 15 |  | 30 |  |

1. Разработать программу, в которой используется класс в соответствии с вариантом. Описание и реализация методов класса должны быть в разных файлах

***Вариант 1***

Разработайте класс *Окружность*. Свойство: радиус. Методы: длина окружности и площадь круга, ограниченного окружностью.

На основе разработанного класса решите следующую задачу: для заданных радиусов двух окружностей определите, у какого круга большая площадь и насколько длина одной окружности отличается от другой. Ответ выведите на форму.

Формулы для расчета:



где *r* – радиус окружности;

*C* – длина окружности;

*S* – площадь круга.

***Вариант 2***

Разработайте класс *Прямоугольный\_Треугольник*. Свойства: длины двух катетов. Методы: длина гипотенузы и площадь треугольника.

На основе разработанного класса решите следующую задачу: для заданных длин катетов двух треугольников определите, у какого треугольника большая гипотенуза, а у какого большая площадь. Ответ выведите на форму.

Формулы для расчета:



где *a* и *b* – длины катетов;

*c* – длина гипотенузы;

*S* – площадь прямоугольного треугольника.

***Вариант 3***

Разработайте класс *Куб*. Свойство: длина стороны. Методы: площадь поверхности и объем.

На основе разработанного класса решите следующую задачу: для заданных длин сторон двух кубов определите, у какого куба большая поверхность, и насколько объем одного куба больше другого. Ответ выведите на форму.

Формулы для расчета:



где *a* – длина стороны куба;

*S* – площадь поверхности куба;

*V* – объем куба.

***Вариант 4***

Разработайте класс *Треугольник*. Свойства: длина одной стороны, величины двух прилежащих к заданной стороне углов. Методы: площадь треугольника, величина третьего угла.

На основе разработанного класса решите следующую задачу: для заданных длин сторон и величин углов двух треугольников определите, у какого треугольника большая площадь, а у какого самый большой угол. Ответ выведите на форму.

Формулы для расчета:



где *a* – длина стороны треугольника;

*α* и *β* – углы, прилежащие к стороне *a*, град.;

*γ* – третий угол треугольника, град.;

*S* – площадь треугольника.

***Вариант 5***

Разработайте класс *Отрезок*. Свойство: координаты концов отрезка. Методы: длина отрезка, проверка параллельности оси ОX.

На основе разработанного класса решите следующую задачу: для заданных координат концов двух отрезков определите, у какого отрезка большая длина, а какой из отрезков параллелен оси OX. Ответ выведите на форму.

Формула для расчета:



где (*x*1, *y*1) и (*x*2, *y*2) – координаты концов отрезка;

*r* – длина отрезка.

Условие параллельности отрезка оси ОХ:

.

***Вариант 6***

Разработайте класс *Треугольник*. Свойства: координаты вершин треугольника. Методы: площадь треугольника, длина большей стороны.

На основе разработанного класса решите следующую задачу: для заданных координат вершин двух треугольников определите, у какого треугольника большая площадь, а у какого самая большая длина стороны. Ответ выведите на форму.

Формулы для расчета:

где (*x*1, *y*1), (*x*2, *y*2) и (*x*3, *y*3) – координаты вершин треугольника;

*a*, *b* и *c* – длины сторон треугольника;

*p* – полупериметр треугольника;

*S* – площадь треугольника.

***Вариант 7***

Разработайте класс *Окружность*. Свойство: диаметр. Методы: длина окружности и площадь круга, ограниченного окружностью.

На основе разработанного класса решите следующую задачу: для заданных диаметров двух окружностей определите, у какого круга большая площадь и насколько длина одной окружности отличается от другой. Ответ выведите на форму.

Формулы для расчета:



где *d* – диаметр окружности;

*C* – длина окружности;

*S* – площадь круга.

***Вариант 8***

Разработайте класс *Прямоугольник*. Свойства: координаты трех вершин прямоугольника. Методы: площадь прямоугольника и длина диагонали.

На основе разработанного класса решите следующую задачу: для заданных координат вершин двух прямоугольников определите, у какого прямоугольника большая диагональ, а у какого большая площадь. Ответ выведите на форму.

Формулы для расчета:



где (*x*1, *y*1), (*x*2, *y*2) и (*x*3, *y*3) – координаты вершин прямоугольника;

*a* и *b* – длины сторон прямоугольника;

*d* – длина диагонали прямоугольника;

*S* – площадь прямоугольника.

***Вариант 9***

Разработайте класс *Параллелограмм*. Свойства: длины сторон параллелограмма и острый угол. Методы: площадь и периметр параллелограмма.

На основе разработанного класса решите следующую задачу: для заданных сторон и острого угла двух параллелограммов определите, у какого параллелограмма больший периметр, а у какого большая площадь. Ответ выведите на форму.

Формулы для расчета:

, P = 2(a+b)

где *a* и *b* – длины сторон параллелограмма;

– острый угол параллелограмма;

*P* – периметр параллелограмма;

*S* – площадь параллелограмма.

***Вариант 10***

Разработайте класс *Шар*. Свойства: радиус. Методы: площадь поверхности и объём шара.

На основе разработанного класса решите следующую задачу: для заданных радиусов двух шаров определите, у какого шара больший объём, а у какого большая площадь поверхности. Ответ выведите на форму.

Формулы для расчета:

 

где *R* – радиус шара;

***Вариант 11***

Разработайте класс *Окружность*. Свойство: радиус. Методы: длина окружности и площадь круга, ограниченного окружностью.

На основе разработанного класса решите следующую задачу: для заданных радиусов двух окружностей определите, у какого круга большая площадь и насколько длина одной окружности отличается от другой. Ответ выведите на форму.

Формулы для расчета:



где *r* – радиус окружности;

*C* – длина окружности;

*S* – площадь круга.

***Вариант 12***

Разработайте класс *Прямоугольный\_Треугольник*. Свойства: длины двух катетов. Методы: длина гипотенузы и площадь треугольника.

На основе разработанного класса решите следующую задачу: для заданных длин катетов двух треугольников определите, у какого треугольника большая гипотенуза, а у какого большая площадь. Ответ выведите на форму.

Формулы для расчета:



где *a* и *b* – длины катетов;

*c* – длина гипотенузы;

*S* – площадь прямоугольного треугольника.

***Вариант 13***

Разработайте класс *Куб*. Свойство: длина стороны. Методы: площадь поверхности и объем.

На основе разработанного класса решите следующую задачу: для заданных длин сторон двух кубов определите, у какого куба большая поверхность, и насколько объем одного куба больше другого. Ответ выведите на форму.

Формулы для расчета:



где *a* – длина стороны куба;

*S* – площадь поверхности куба;

*V* – объем куба.

***Вариант 14***

Разработайте класс *Треугольник*. Свойства: длина одной стороны, величины двух прилежащих к заданной стороне углов. Методы: площадь треугольника, величина третьего угла.

На основе разработанного класса решите следующую задачу: для заданных длин сторон и величин углов двух треугольников определите, у какого треугольника большая площадь, а у какого самый большой угол. Ответ выведите на форму.

Формулы для расчета:



где *a* – длина стороны треугольника;

*α* и *β* – углы, прилежащие к стороне *a*, град.;

*γ* – третий угол треугольника, град.;

*S* – площадь треугольника.

***Вариант 15***

Разработайте класс *Отрезок*. Свойство: координаты концов отрезка. Методы: длина отрезка, проверка параллельности оси ОX.

На основе разработанного класса решите следующую задачу: для заданных координат концов двух отрезков определите, у какого отрезка большая длина, а какой из отрезков параллелен оси OX. Ответ выведите на форму.

Формула для расчета:



где (*x*1, *y*1) и (*x*2, *y*2) – координаты концов отрезка;

*r* – длина отрезка.

Условие параллельности отрезка оси ОХ:

.