Learning C
by Example

Agus Kurniawan

Copyright

Learning C by Example

Agus Kurniawan

1st Edition, 2015

Copyright © 2015 Agus Kurniawan

Table of Contents

Copyright

Preface

1. Development Environment

1.1 Getting Started

1.2 Compilers

1.2.1 Linux

1.2.2 Windows

1.2.3 Mac

1.3 Development Tools

1.4 Hello World

2. Basic C Programming Language

2.1 Common Rule

2.2 Declaring Variable

2.3 Assigning Variables

2.4 Comment

2.5 Input & Output on Console

2.6 Arithmetic Operations

2.7 Mathematical Functions

2.8 Comparison Operators

2.9 Logical Operators

2.10 Increment and Decrement

2.11 Decision

2.11.1 if..then

2.11.2 switch..case

2.12 Iterations

2.12.1 For

2.12.2 While

2.13 Struct

3. Array and Pointer

3.1 Array
3.1.1 Defining An Array
3.1.2 Basic Array Operations
3.2 Multidimensional Array
3.3 Pointer
3.3.1 Basic Pointer
3.3.2 Dynamic Array
4. Functions
4.1 Creating Function
4.2 Function with Parameters and Returning Value
4.3 Function with Array Parameters
4.4 Function and Pointer
5. I/O Operations
5.1 Getting Started
5.2 Reading Input from Keyboard
5.2.1 getchar() and putchar() functions
5.2.2 gets() and puts() functions
5.2.3 scanf() function
5.3 Reading Program Arguments
5.4 Writing Data Into A File
5.5 Reading Data From A File
6. String Operations
6.1 Concatenating Strings
6.2 String To Numeric
6.3 Numeric to String
6.4 String Parser
6.5 Check String Data Length
6.6 Copy Data
6.7 Exploring Characters
7. Building C Library
7.1 Getting Started

7.2 Writing C Library
7.3 Compiling and Testing
7.3.1 Static Library
7.3.2 Shared Library
8. Threading
8.1 Creating Thread
8.2 Thread ID
8.3 Terminating Thread
8.3.1 Terminating Itself
8.3.2 Terminating Others
8.4 Joining Thread
8.5 Thread Mutex
8.6 Condition Variables
8.6.1 Signaling
8.6.2 Broadcasting
9. Database Programming
9.1 Database Library for C
9.2 MySQL
9.3 Connection Test
9.4 CRUD (Create, Read, Update and Delete) Operations
9.4.1 Creating Data
9.4.2 Reading Data
9.4.3 Updating Data
9.4.4 Deleting Data
10. Socket Programming
10.1 Getting Local Hostname
10.2 Creating and Connecting
10.2.1 Server
10.2.2 Client
10.2.3 Testing
10.3 Data Transfer

10.3.1 Server

10.3.2 Client

10.3.3 Testing

Contact and Source Code

Contact

Preface

This book is a practical book to get started with C language. It describes all the elements of the language and illustrates their use with code examples.

Agus Kurniawan

Depok, March 2015

1. Development Environment

This chapter explains how to start with development environment using C.

1.1 Getting Started

C is a general-purpose, imperative computer programming language. It supports structured programming, lexical variable scope and recursion, while a static type system prevents many unintended operations.

In this chapter, we prepare development environment to develop C program. You can write C program with your program. We focus on Windows, Linux and Mac platforms.

1.2 Compilers

There are many C compilers to develop C program. In this book, we use GCC as C compiler for Linux, Mac and Windows. For Windows users, you also can use Visual C++ compiler.

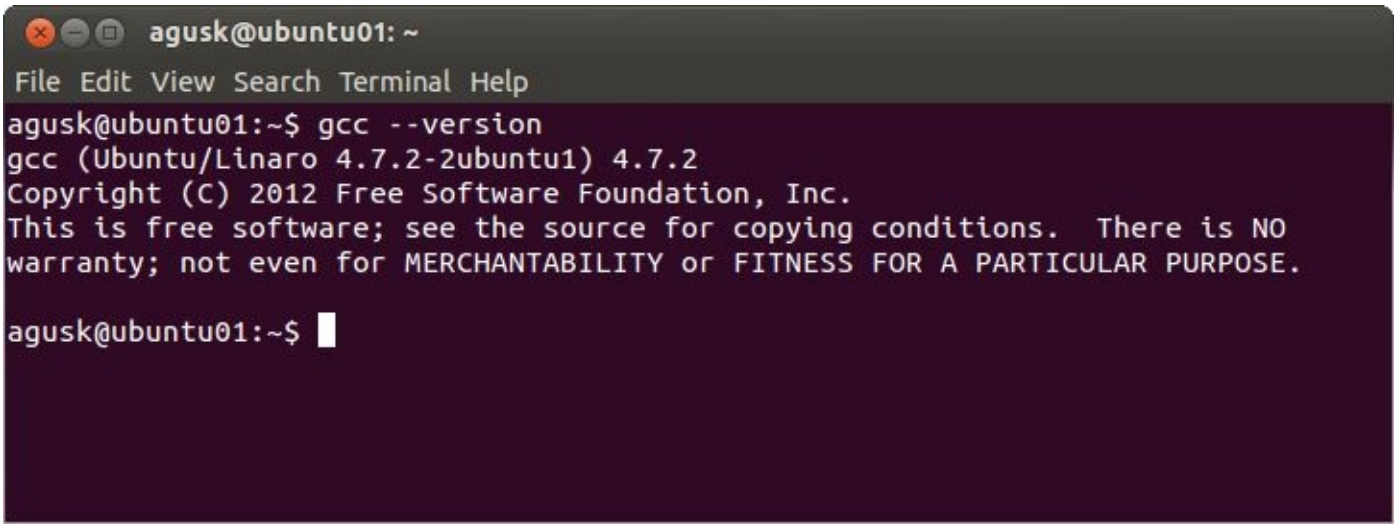
1.2.1 Linux

Installation of GCC C is easy. For Ubuntu Linux, you can do it using console and write this script

```
$ sudo apt-get install build-essential
```

If installation already finished, you can check GCC version on console and write script as below

```
gcc --version
```

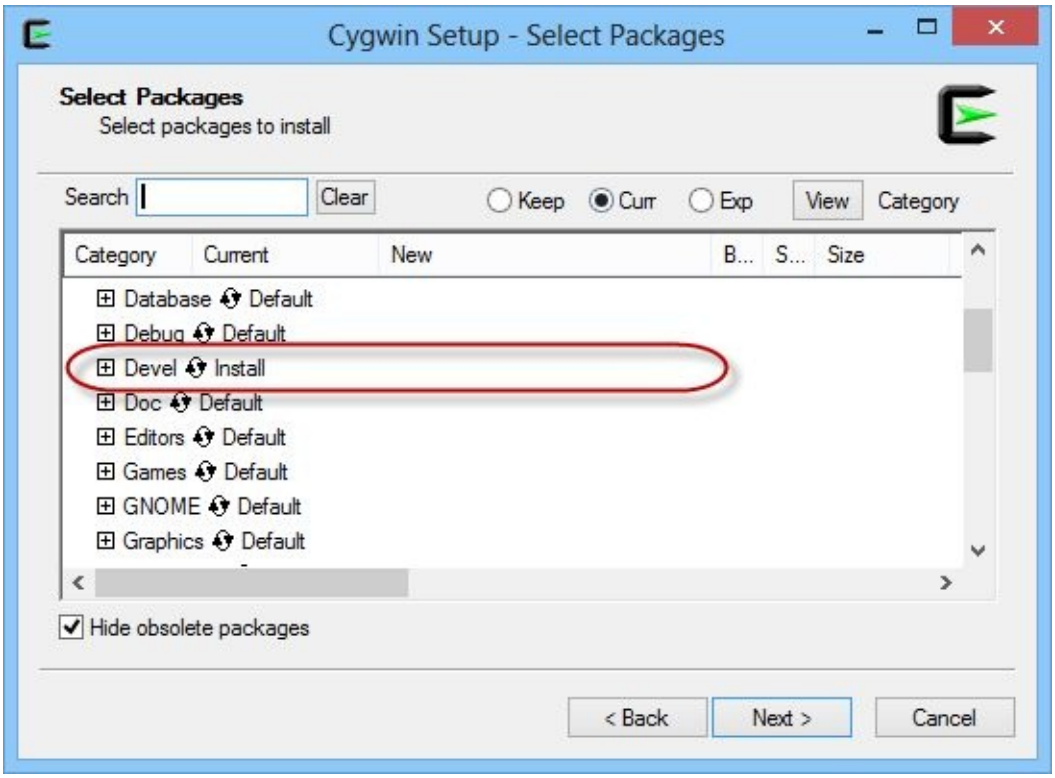


You will see gcc version, for instance shown in Figure above.

1.2.2 Windows

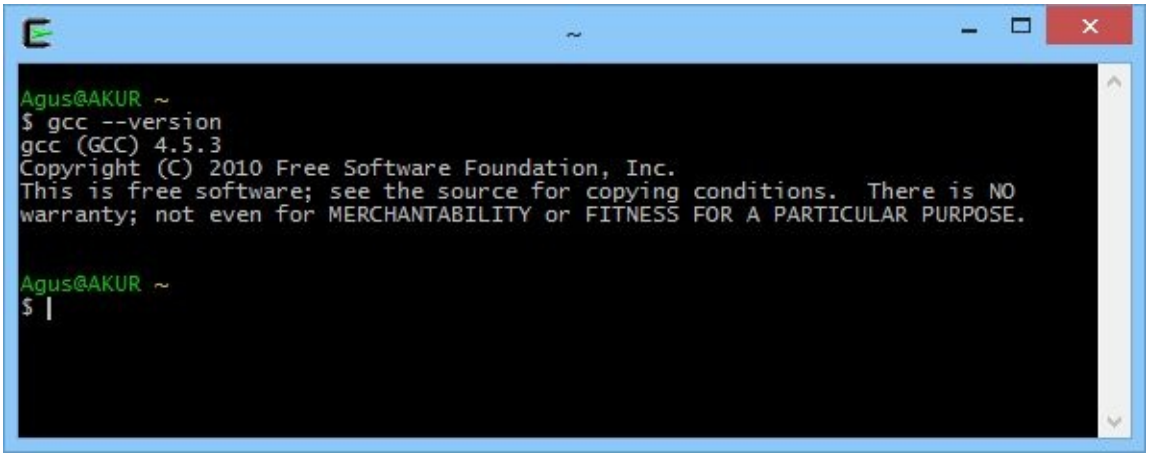
If your computer has Windows platform, you can use **Cygwin**. Download it on http://cygwin.com/setup.exe and then run it.

On setup dialog, you choose **Devel** package as follows.



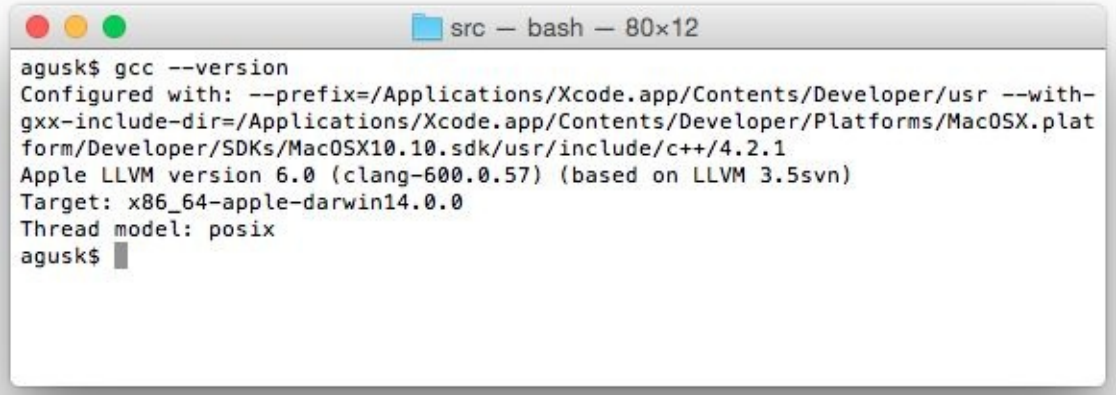
After installed, you will get Cygwin terminal. Run it.

You can check GCC version. The following is a sample of output for GCC version



1.2.3 Mac

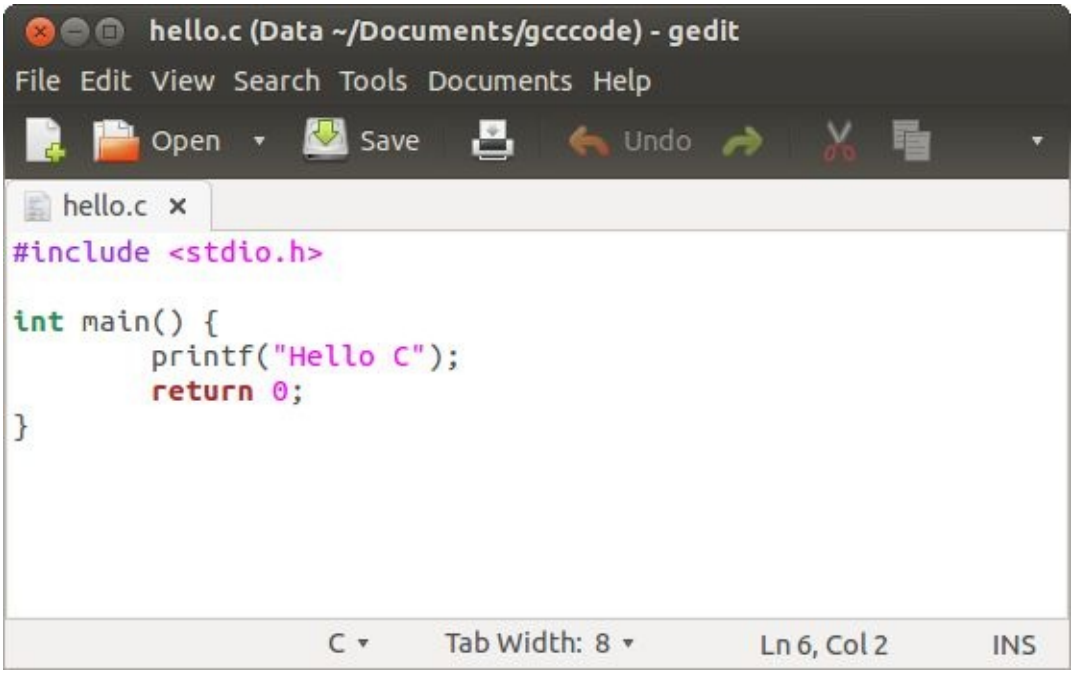
To install GCC, you can install Xcode so you get GCC. After installed, you can verify it by checking GCC version.


src — bash — 80×12
agusk$ gcc --version
Configured with: --prefix=/Applications/Xcode.app/Contents/Developer/usr --with-gxx-include-dir=/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.10.sdk/usr/include/c++/4.2.1
Apple LLVM version 6.0 (clang-600.0.57) (based on LLVM 3.5svn)
Target: x86_64-apple-darwin14.0.0
Thread model: posix
agusk$

1.3 Development Tools

Basically you can use any editor tool for instance, vi, vim, gedit, Eclipse. The following is vi editor in Ubuntu.

Here is gedit editor



1.4 Hello World

Now we start to write the first program using C.

Firstly, open your text editor and create new file, called **hello.c**

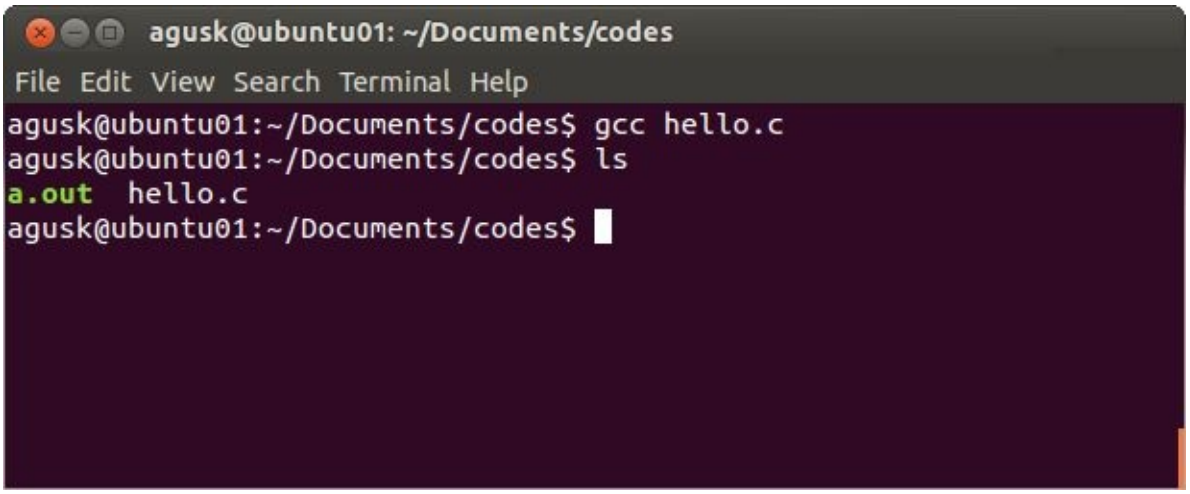
Let's write this code

```
#include <stdio.h>

int main() {
        printf("Hello C\n");
        return 0;
}
```

Save this file.

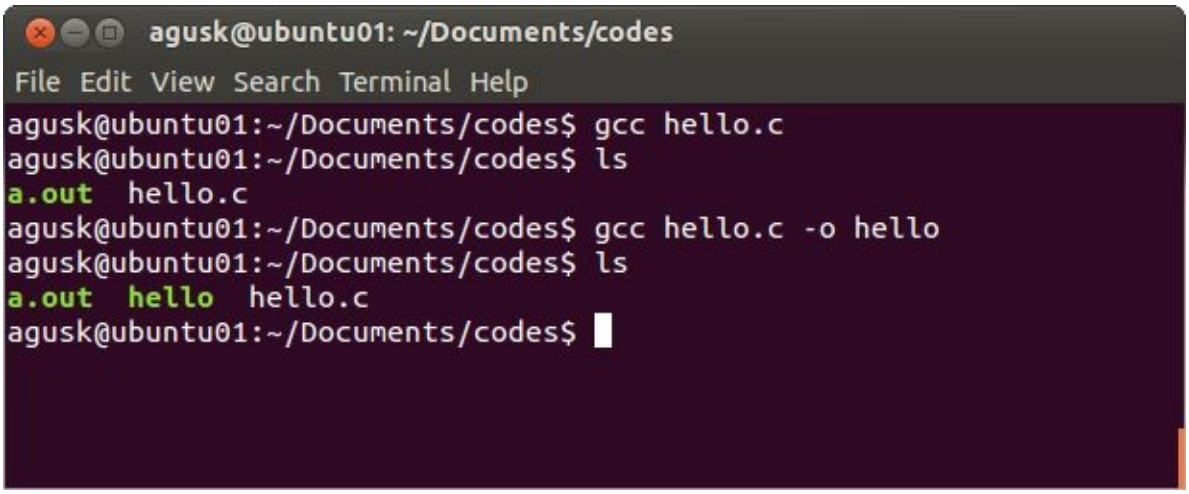
Now open your Terminal and compile this file.

```
sudo gcc hello.c
```



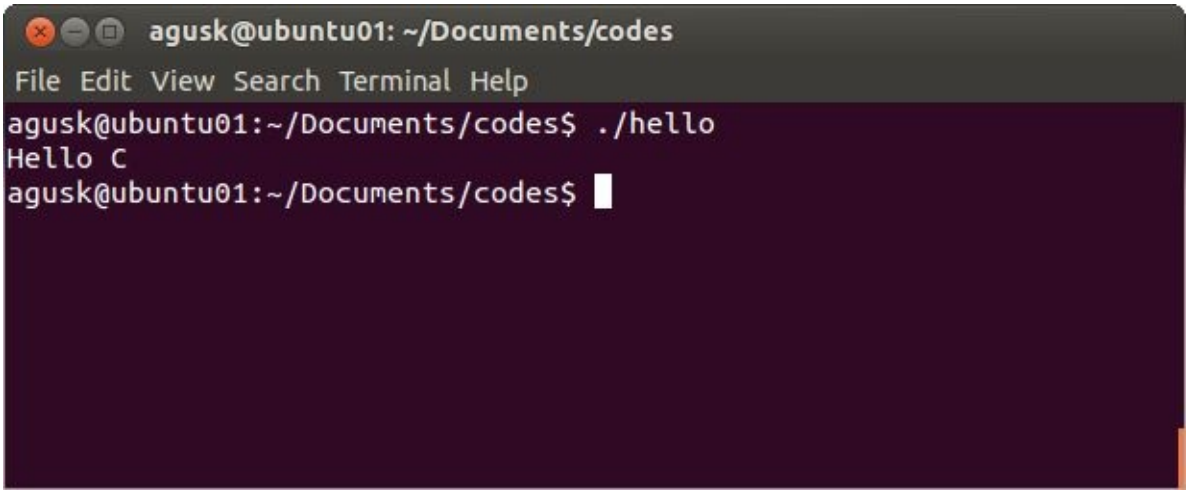
If you check, you will get file **a.out**. GCC will generate a.out file if you don't specify an output file name. You can define the output compiled file as follows

```
sudo gcc hello.c -o hello
```

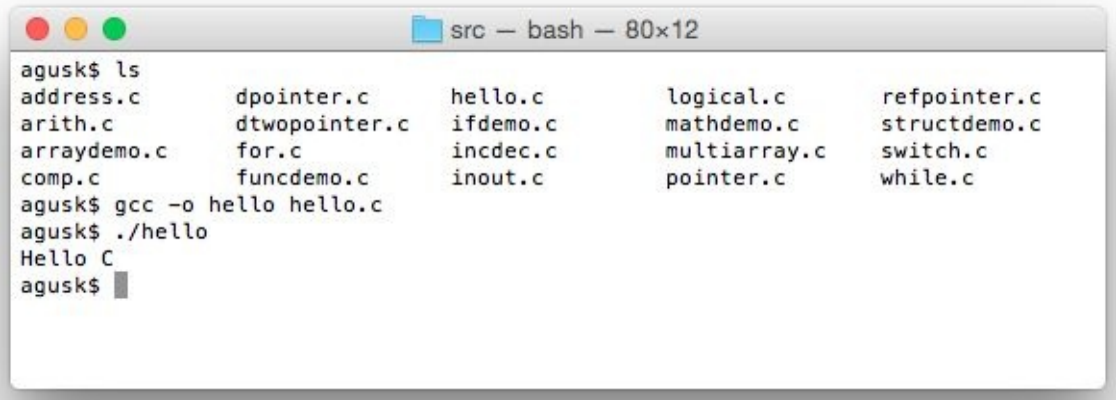


After compiled, you can run the compiled file by writing this command (for instance, the compiled file is hello.out)

```
./hello
```



A sample output of program is run on Mac platform.



2. Basic C Programming Language

This chapter explains the basic of C programming language.

2.1 Common Rule

In C language if you write a line of code we must write semicolon (;) at the end of code. Here is the syntax rule:

```
syntax_code;
```

2.2 Declaring Variable

To declare a variable called *myvar1*, write the following:

```
int myvar1;
```

int is data type.

The following is the list of common data type you can use in C language.

- int
- long
- float
- char
- double

We can also write many variables as follows:

```
int myvar1, myvar2;
```

Once declared, these variables can be used to store data based on its data type.

2.3 Assigning Variables

Variable that you already declare can be assigned by a value. It can done using the equals sign (=). For example, variable **myvar1** will assign the number 100, you would write this:

```
int myvar1 = 100;
```

You also declare as below

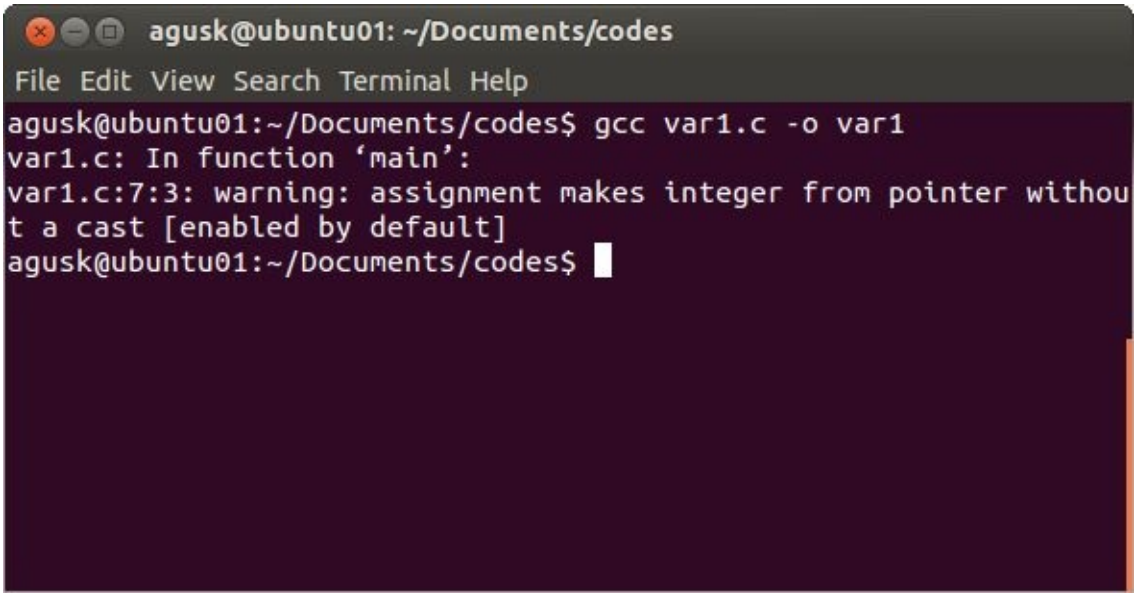
```
int myvar1;
myvar1 = 100;
```

You must assign a value on a variable properly based on data type. If not, you will get error on compiling process, for instance, write this code

```
#include <stdio.h>

int main() {
        int n;
        int i,j;

        n="hello";

        return 0;
}
```

Save this code as file, called **var1.c**.

Try to compile this file. You will get a warning message as shown Figure as below



2.4 Comment

You may explain how to work on your code with writing C. To do it, you can use // and /* */ syntax. Here is sample code:

```
// bank account
char accountCode;

/* parameters*/
int p1, p2, p3, p4;
```

2.5 Input & Output on Console

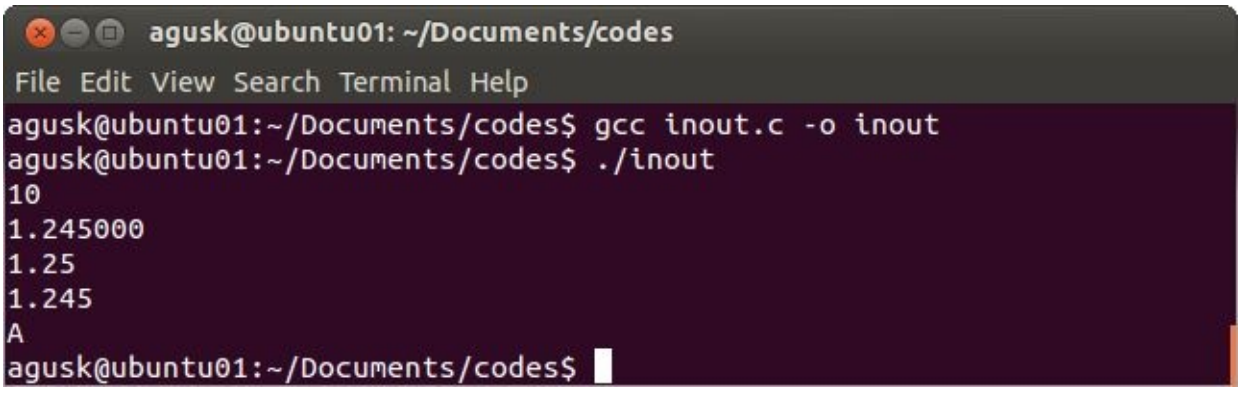
You may want to show message on console using C. You can use **printf()** to write message on console. printf() also can be passed parameters inside using *%d* for integer numeric and *%f* for floating numeric. For char data type, we can use *%c* for passing parameter.

Here is a sample code

```c
#include <stdio.h>

int main() {
        int n = 10;
        float m = 1.245;
        char c = 'A';

        printf("%d \n",n);
        printf("%f \n",m);
        printf("%.2f \n",m);
        printf("%.3f \n",m);
        printf("%c \n",c);

        return 0;
}
```

Save this code into a file, called **inout.c**.

Compile and run this code



You can see on *%f* that wrote floating data with 6 decimal digits. You can specify the number of decimal digit by passing numeric after dot (.), for instance, *%.2f* and *%.3f*.

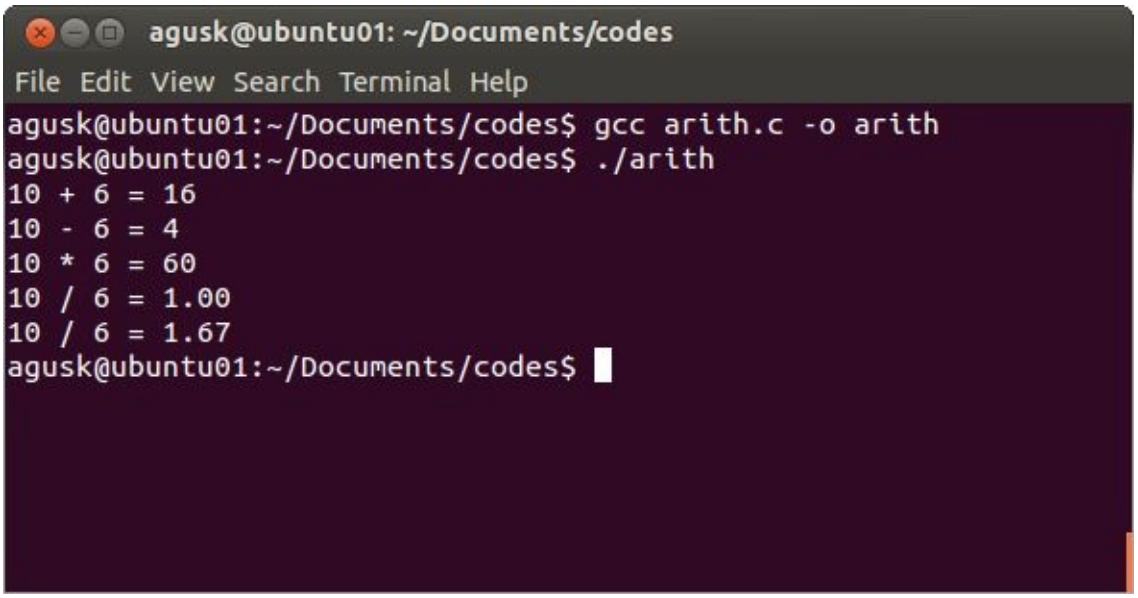
2.6 Arithmetic Operations

C supports the four basic arithmetic operations such as addition, subtraction, multiplication, and division. The following is the code illustration for basic arithmetic:

```c
#include <stdio.h>

int main() {
        int a,b,c;
        a = 10;
        b = 6;

        c = a + b;
        printf("%d + %d = %d\n",a,b,c);

        c = a - b;
        printf("%d - %d = %d\n",a,b,c);

        c = a * b;
        printf("%d * %d = %d\n",a,b,c);

        float d = a / b;
        printf("%d / %d = %.2f\n",a,b,d);

        float e =(float)a / b;
        printf("%d / %d = %.2f\n",a,b,e);

        return 0;
}
```

Save this program into a file, called **arith.c**. Compile and run it.

Program run:



2.7 Mathematical Functions

C provides math library to manipulate mathematical operations. You must include **math.h** to implement mathematical operations.

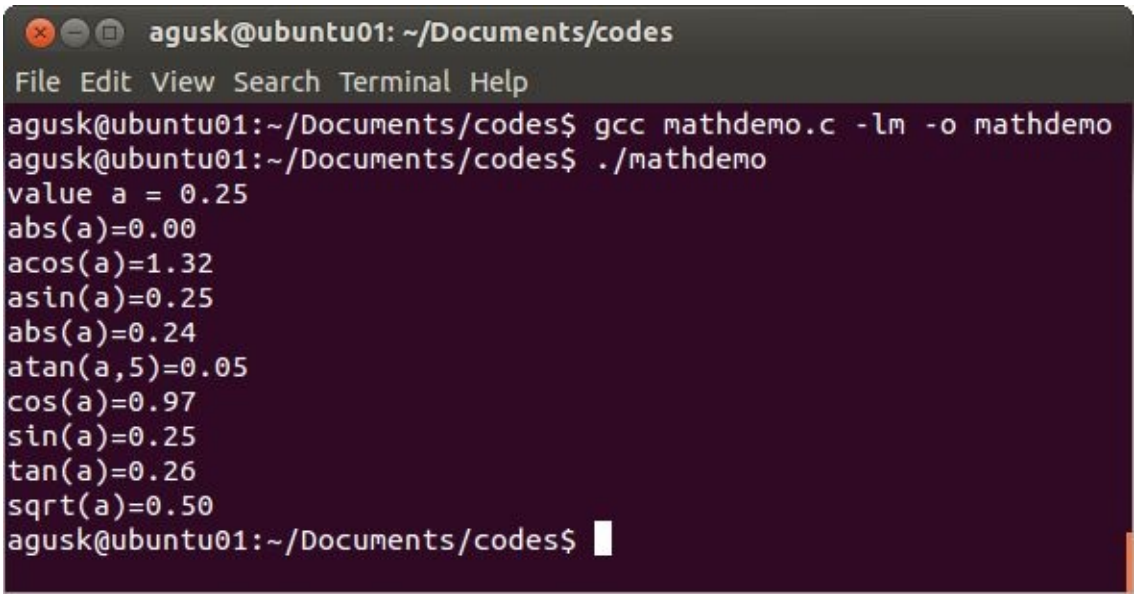
Here is an illustration code for mathematical functions usage:

```c
#include <stdio.h>
#include <stdlib.h>

#include <math.h>

int main() {
        float a;
        float b;
        float pi = 3.14;

        a = 0.25;
        printf("value a = %.2f \n",a);

        b = abs(-a);
        printf("abs(a)=%.2f \n",b);

        b = acos(a);
        printf("acos(a)=%.2f \n",b);

        b = asin(a);
        printf("asin(a)=%.2f \n",b);

        b = atan(a);
        printf("abs(a)=%.2f \n",b);

        b = atan2(a,5);
        printf("atan(a,5)=%.2f \n",b);

        b = cos(a);
        printf("cos(a)=%.2f \n",b);

        b = sin(a);
        printf("sin(a)=%.2f \n",b);

        b = tan(a);
        printf("tan(a)=%.2f \n",b);

        b = sqrt(a);
        printf("sqrt(a)=%.2f \n",b);

        return 0;
}
```

Save this code into a file, called **mathdemo.c**. Then compile and run it. Because we use math.h, we add -lm parameter on compiling.

```
$ gcc mathdemo.c -lm -o mathdemo
$ ./mathdemo
```

A program output can be seen in Figure below.



2.8 Comparison Operators

You may determine equality or difference among variables or values. Here is the list of comparison operatos:

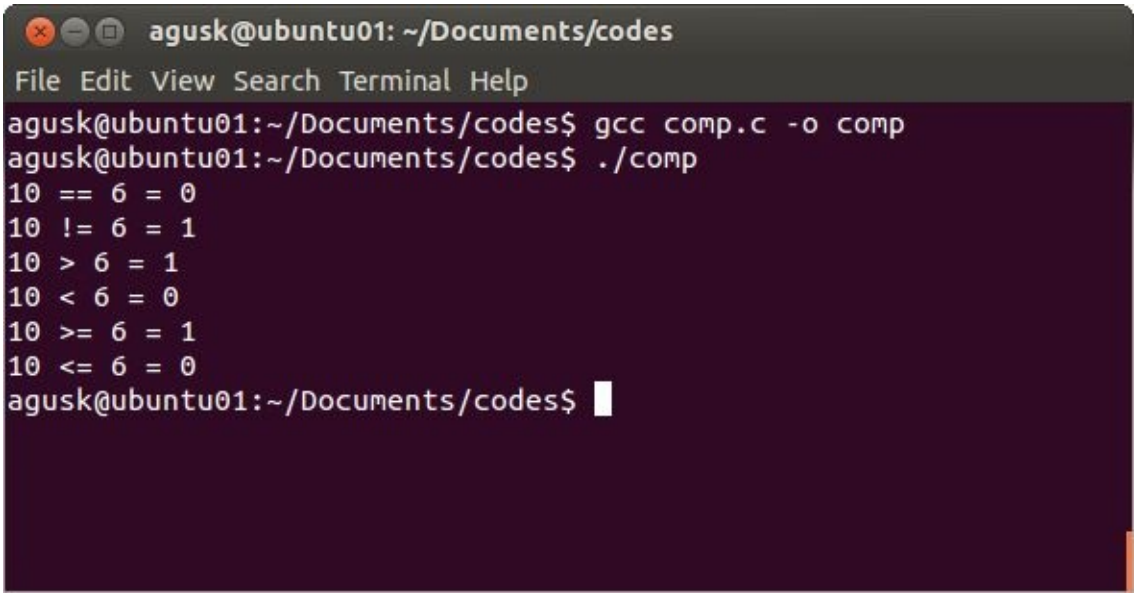
```
==        is equal to
!=        is not equal
>         is greater than
<         is less than
>=        is greater than or equal to
<=        is less than or equal to
```

This is sample code for comparison usage

```
#include <stdio.h>

int main() {
        int a,b;
        a = 10;
        b = 6;

        printf("%d == %d = %d\n",a,b,a==b);
        printf("%d != %d = %d\n",a,b,a!=b);
        printf("%d > %d = %d\n",a,b,a>b);
        printf("%d < %d = %d\n",a,b,a<b);
        printf("%d >= %d = %d\n",a,b,a>=b);
        printf("%d <= %d = %d\n",a,b,a<=b);

        return 0;
}
```

Save this code as file, called **comp.c**.

Here is a sample of program output.



2.9 Logical Operators

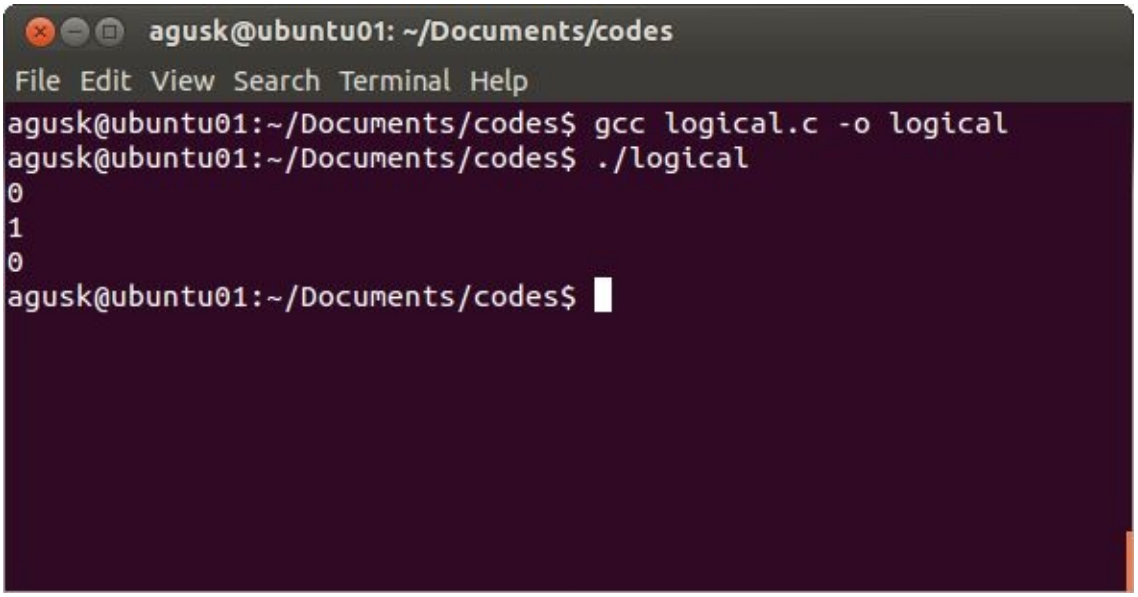
These operators can be used to determine the logic between variables or values.

```
&&      and
||      or
!        not
```

To illustrate how to use logical operators in C code, you can write this code.

```c
#include <stdio.h>

int main() {
        int a,b;
        a = 5;
        b = 8;

        printf("%d \n",a>b && a!=b);
        printf("%d \n",!(a>=b));
        printf("%d \n",a==b || a>b);

        return 0;
}
```

Save this code into a file, **logical.c**. Compile and run it.



2.10 Increment and Decrement

Imagine you have operation as below

```
num = num + 1;
```

you can simply this syntax use ++.

```
num++;
```

using the same approach for this case

```
num = num - 1;
```

you can use — syntax.

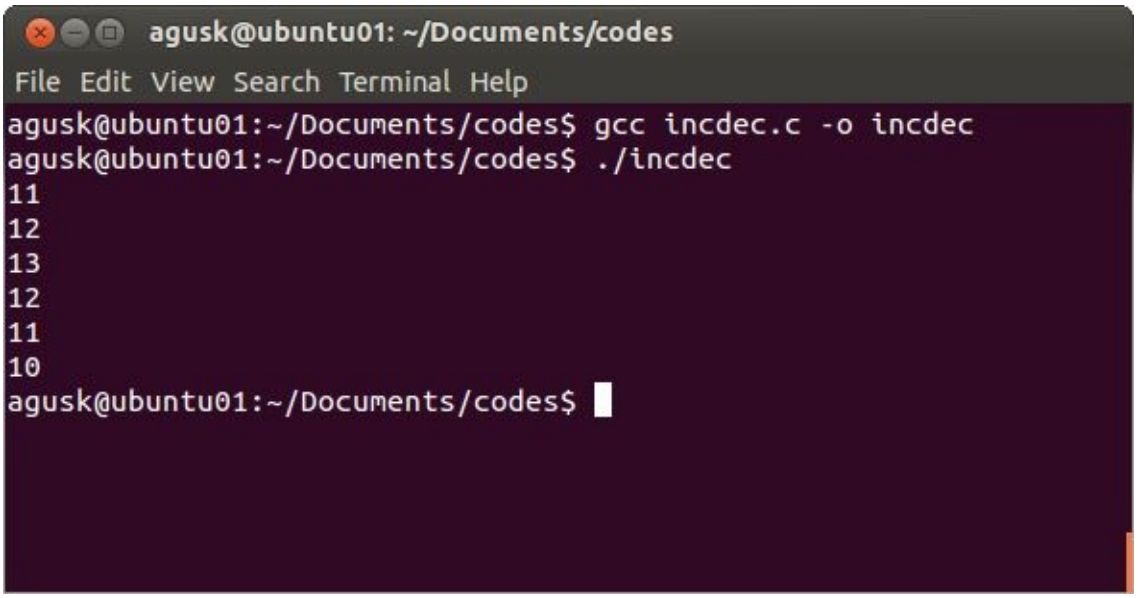
```
num--;
```

Now how to implement them in code. Let’s write this code.

```
#include <stdio.h>

int main() {
        int a = 10;

        a++;
        printf("%d \n",a);

        a++;
        printf("%d \n",a);

        ++a;
        printf("%d \n",a);

        a--;
        printf("%d \n",a);

        a--;
        printf("%d \n",a);

        --a;
        printf("%d \n",a);

        return 0;
}
```

Save this code into a file, called **incdec.c**.

Compile and run it. Here is sample of program output.


agusk@ubuntu01: ~/Documents/codes
File Edit View Search Terminal Help
agusk@ubuntu01:~/Documents/codes$ gcc incdec.c -o incdec
agusk@ubuntu01:~/Documents/codes$ ./incdec
11
12
13
12
11
10
agusk@ubuntu01:~/Documents/codes$

2.11 Decision

There are two approaches to build decision on C. We can use *if..then* and *switch..case*.

2.11.1 if..then

Syntax model for *if..then* can be formulated as below:

```
if (conditional) {
   // do something
}else{
   // do another job
}
```

conditional can be obtained by logical or/and comparison operations.

Here is the sample code:

```
#include <stdio.h>

int main() {
        int a, b;
        a = 5;
        b = 8;

        if(a>b || a-b<a){
                printf("conditional-->a>b || a-b<a \n");
        }else{
                printf("..another \n");
        }
        return 0;
}
```

Save this program into a file, called i**fdemo.c**. Then, run this file.

Program run:

2.11.2 switch..case

switch..case can be declared as below:

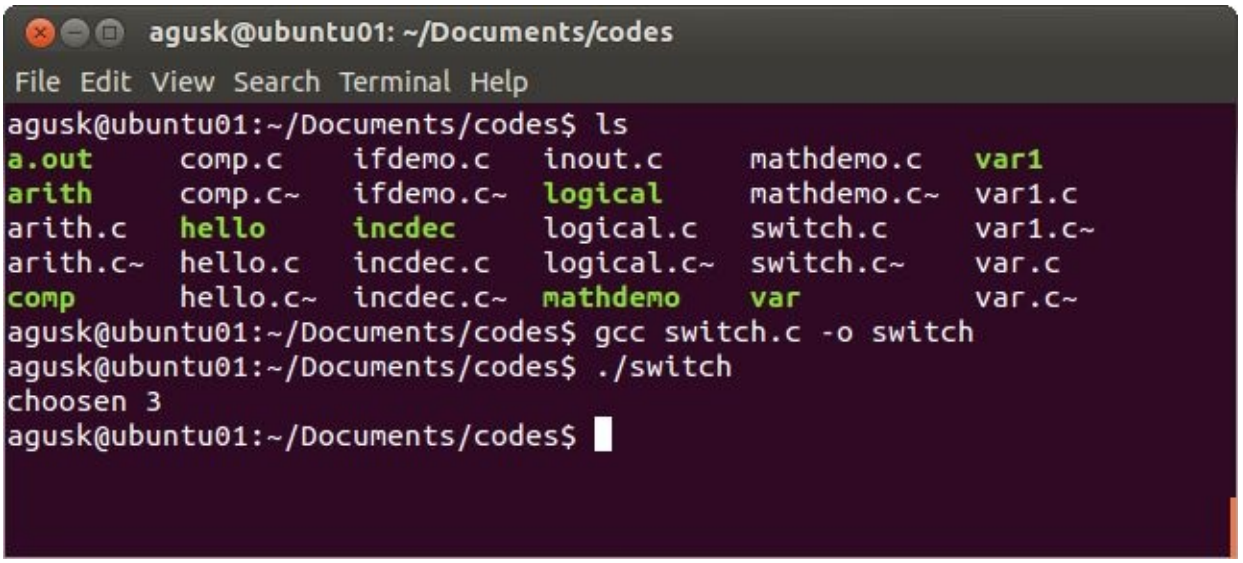
```
switch(option){
        case option1:
                // do option1 job
                break;
        case option2:
                // do option2 job
                break;
}
```

Here is the sample code of *switch..case* usage:

```
#include <stdio.h>

int main() {
        // you can obtain input value from keyboard
        // or any input device
        int input = 3;

        switch(input){
        case 1:
                printf("choosen 1\n");
                break;
        case 2:
                printf("choosen 2\n");
                break;
        case 3:
        case 4:
                printf("choosen 3\n");
                break;
        }
        return 0;
}
```

Save this code into a file, called **switch.c**. Try to build and run it.

Program run:


agusk@ubuntu01: ~/Documents/codes
File Edit View Search Terminal Help
agusk@ubuntu01:~/Documents/codes$ ls
a.out comp.c ifdemo.c inout.c mathdemo.c var1
arith comp.c~ ifdemo.c~ logical mathdemo.c~ var1.c
arith.c hello incdec logical.c switch.c var1.c~
arith.c~ hello.c incdec.c logical.c~ switch.c~ var.c
comp hello.c~ incdec.c~ mathdemo var var.c~
agusk@ubuntu01:~/Documents/codes$ gcc switch.c -o switch
agusk@ubuntu01:~/Documents/codes$ ./switch
choosen 3
agusk@ubuntu01:~/Documents/codes$

2.12 Iterations

Iteration operation is useful when we do repetitive activities. We use for and *while* syntax

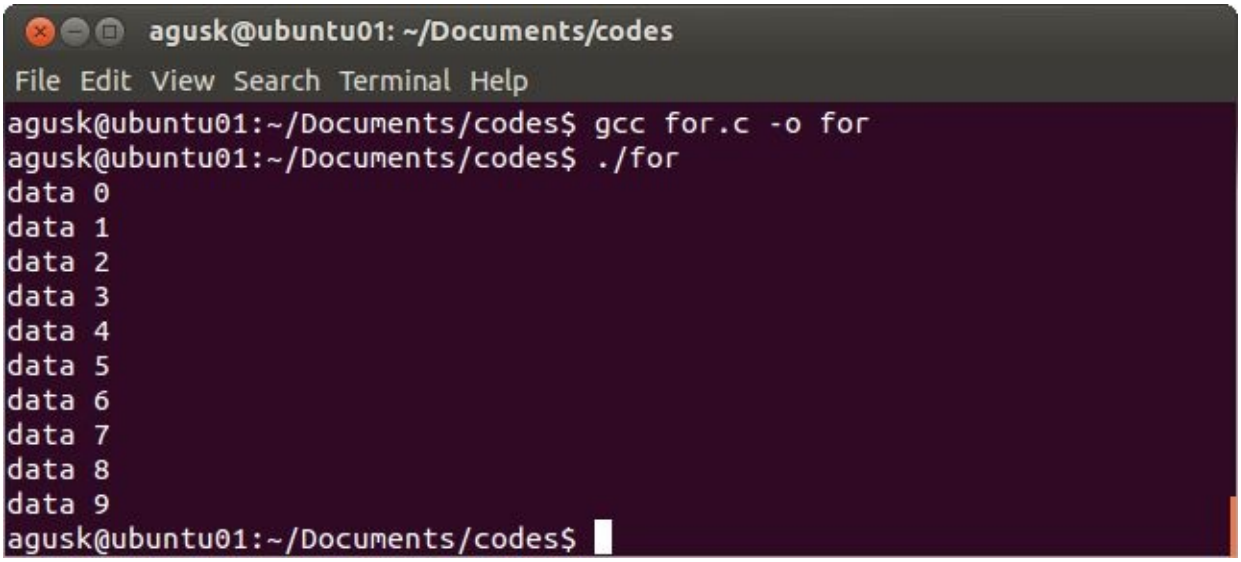
2.12.1 For

The simple scenario that illustrates iteration scenario is to show list of number. We do iteration until the number value less than 10.

```
#include <stdio.h>

int main() {

        int i;
        for(i=0;i<10;i++){
                printf("data %d\n",i);
        }
        return 0;
}
```

Save this code into a file, called f**or.c**. Compile and run it.

Here is a program output:



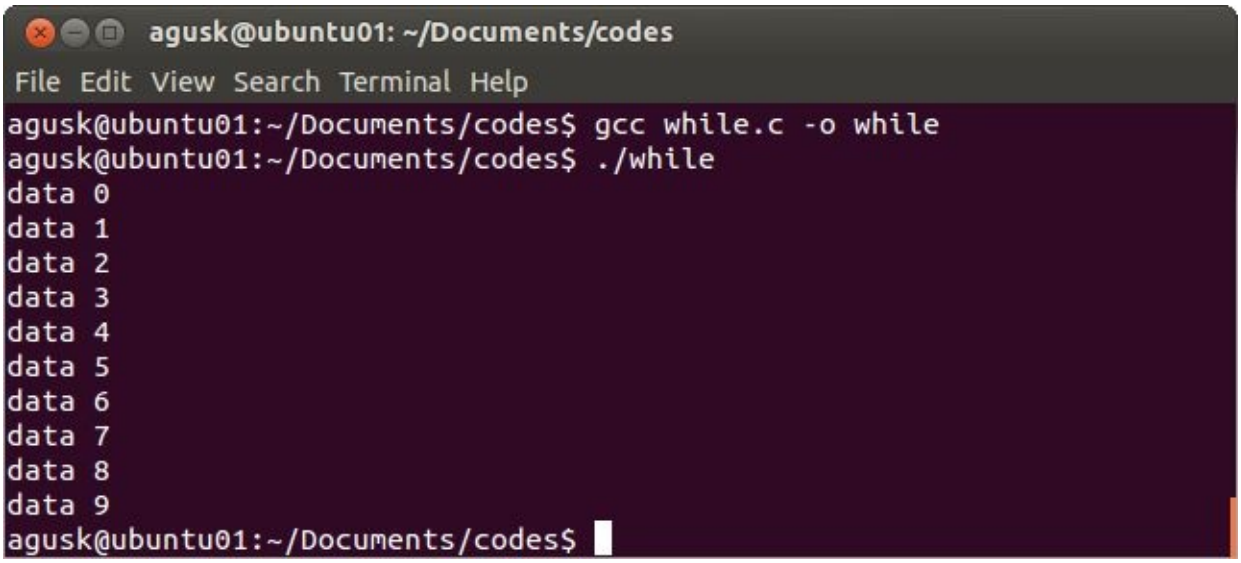
2.12.2 While

The following is the same code like 2.12.1 scenario but it uses *while* syntax.

```
#include <stdio.h>

int main() {
        int num = 0;
```

```
        while(num<10){
                printf("data %d\n",num);
                num++;
        }
        return 0;
}
```

Save this code into a file, called **while.c**. Try to compile and run it.



2.13 Struct

We can define a struct to declare a new data type. We use struct keyword. For illustration, we define new data type, called employee.

Create a file, called **structdemo.c**, and write this code.

```
#include <stdio.h>

// define a struct
struct employee{
    int id;
    char name[10];
    char country[5];
};

int main() {

    // declare struct variable
    struct employee emp;

    // set values
    emp.id = 10;
    sprintf(emp.name,"jane");
    sprintf(emp.country,"DE");

    // display
    printf("id: %d, name: %s, country: %s\n",emp.id,emp.name,emp.country

    return 0;
}
```

Save this code.

Now you compile and run this program.

```
$ gcc -o structdemo structdemo.c
$ ./structdemo
```

A sample output of program can be seen in Figure below, Mac platform.

src — bash — 80×12
agusk$ ls
address.c dtwopointer.c ifdemo.c multiarray.c while.c
arith.c for.c incdec.c pointer.c
arraydemo.c funcdemo.c inout.c refpointer.c
comp.c hello logical.c structdemo.c
dpointer.c hello.c mathdemo.c switch.c
agusk$ gcc -o structdemo structdemo.c
agusk$ ./structdemo
id: 10, name: jane, country: DE
agusk$

3. Array and Pointer

This chapter explains how to work with array and Pointer.

3.1 Array

In this section, we build an array using C. For illustration, we develop single and multi dimensional array.

3.1.1 Defining An Array

We can define an array using [] syntax. For instance, we define array of int and char.

```
    int numbers[5];
    char chars[10];
```

We also can construct array from struct.

```
struct employee{
    int id;
    char name[10];
    char country[5];
};


struct employee list[5];
```

3.1.2 Basic Array Operations

After declared an array, we can set and get data on array. For illustration, create a file, called **arraydemo.c**, and write this code.

```
#include <stdio.h>

struct employee{
    int id;
    char name[10];
    char country[5];
};

int main() {
    // define array
    int numbers[5];
    char chars[10];
    struct employee list[5];

    // insert data
    int i;
```

```
    for(i=0;i<5;i++){
        numbers[i] = i;

        list[i].id = i;
        sprintf(list[i].name,"usr %d",i);
        sprintf(list[i].country,"DE");
    }
    sprintf(chars,"hello c");

    // display data
    for(i=0;i<5;i++){
        printf("%d %c\n",numbers[i],chars[i]);
        printf("struct. id: %d, name: %s, country: %s \n",
                list[i].id,list[i].name,list[i].country);
    }
    printf("%s\n",chars);

    return 0;
}
```

Save this code and try to compile and run.

```
$ gcc -o arraydemo arraydemo.c
$ ./arraydemo
```

A sample output can be seen in Figure below.

3.2 Multidimensional Array

We can create multidimensional array, for instance two dimensional, we can use [][].

For testing, you can create a file, called **multiarray.c** and write this code.

```
#include <stdio.h>

int main() {
    // define Multidimensional demenarray
    int matrix[3][5];

    // insert data
    int i,j;
    for(i=0;i<3;i++){
        for(j=0;j<5;j++){
            matrix[i][j] = i+j;
        }
    }

    // display data
    for(i=0;i<3;i++){
        for(j=0;j<5;j++){
            printf("%d ",matrix[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Save this. Then, try to compile and run.

```
$ gcc -o multiarray multiarray.c
$ ./multiarray
```

A sample output can be seen in Figure below.

src — bash — 80×17
agusk$ ls
address.c dpointer.c hello.c mathdemo.c structdemo.c
arith.c dtwopointer.c ifdemo.c multiarray.c switch.c
arraydemo for.c incdec.c pointer.c while.c
arraydemo.c funcdemo.c inout.c refpointer.c
comp.c hello logical.c structdemo
agusk$ gcc -o multiarray multiarray.c
agusk$ ./multiarray
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
agusk$

3.3 Pointer

A pointer is a programming language object, whose value refers to (or “points to”) another value stored elsewhere in the computer memory using its address. A pointer references a location in memory, and obtaining the value stored at that location is known as dereferencing the pointer (source: http://en.wikipedia.org/wiki/Pointer_(computer_programming)).

To obtain memory address of a variable, we can use & syntax. For instance, create a file, called **address.c**, and write this code.

```
#include <stdio.h>

int main(int argc, const char* argv[]) {

  int n;
  n = 10;

  printf("value n: %d \n",n);
  printf("address n: %x \n",&n);

  return 0;
}
```

Save this file. Then, try to compile and run.

```
$ gcc -o address address.c
$ ./address
```

If success, we obtain memory address of variable n. A sample output of program can be seen in Figure below.

3.3.1 Basic Pointer

To declare a pointer of a specific data type, we can use * syntax. This variable consists of memory address of our pointer variable.

For illustration, create a file, called **pointer.c**, and write this code.

```c
#include <stdio.h>

int main(int argc, const char* argv[]) {

  int n;
  int* nPtr;


  n = 10;
  nPtr = &n;

  printf("value n: %d \n",n);
  printf("address n: %x \n",(unsigned int)&n);

  printf("value nPtr: %x \n",(unsigned int)nPtr);
  printf("address nPtr: %x \n",(unsigned int)&nPtr);
  printf("value pointer nPtr: %d \n",*nPtr);

  return 0;
}
```

Save this file. Then, compile and run this file.

```
$ gcc -o pointer pointer.c
$ ./pointer
```

You can see a value of nPtr is memory address of variable n. If n is set value 10, then value of pointer nPtr which we declare as *nPtr is the same value with value of variable n.

A sample of program output is shown in Figure below.

3.3.2 Dynamic Array

In this section, we will create a dynamic array using pointer. In general, we can declare an array using [] with size. Now we can declare our array with dynamic size.

Basically, when we add a new value, we allocate memory and then attach it to array. For illustration, we define array with size 10. Then, we add some values to this array.

Create a file, called **dpointer.c**, and write this code.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, const char* argv[]) {

  // define dynamic array of pointer
  int *numbers; // single array pointer

  // a number of array
  int N = 10;
```

```
  // allocate memory
  numbers = malloc( N * sizeof(int));

  // set values
  int i;
  for(i=0;i<N;i++){
    numbers[i] = i+3;
  }

  // display values
  for(i=0;i<N;i++){
      printf("%d ",numbers[i]);
  }
  printf("\n");

  // free memory
  free(numbers);

  return 0;
}
```

Save this code. Try to compile and run it.

```
$ gcc -o dpointer dpointer.c
$ ./dpointer
```

A program output can be seen in Figure below.

Another sample, we also define dynamic array with multidimensional. For instance, we create two dimensional dynamic array. Create a file, called **dtwopointer.c**, and write this

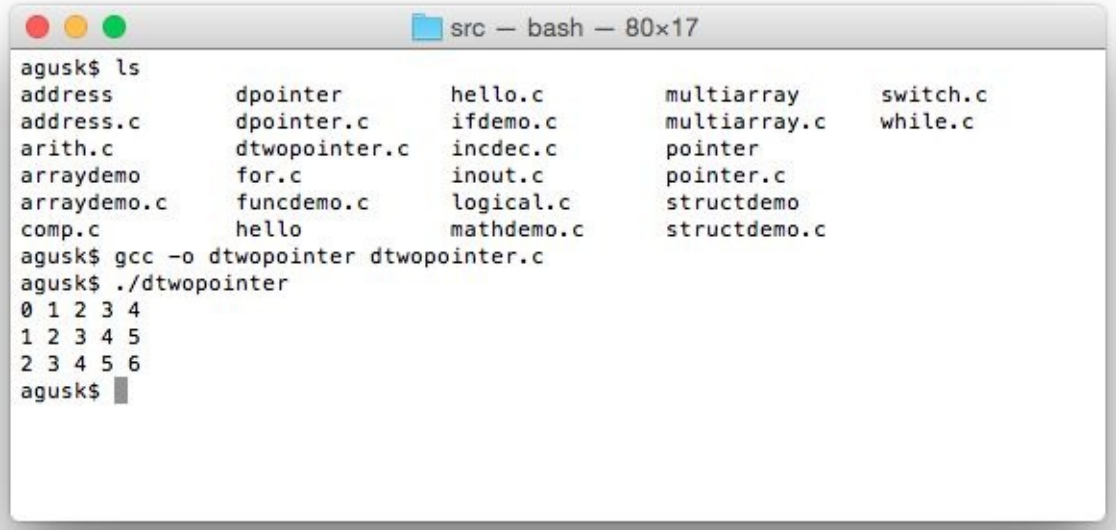
code.

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, const char* argv[]) {

  // define dynamic array of pointer
  int **matrix; // two dimensional array pointer

  // a number of array
  int M = 3;
  int N = 5;

  // allocate memory
  matrix = malloc( M * sizeof(int*));

  // set values
  int i,j;
  for(i=0;i<M;i++){
    matrix[i] = malloc( N * sizeof(int));
    for(j=0;j<N;j++){
        matrix[i][j] = i + j;
    }
  }

  // display values
  for(i=0;i<M;i++){
      for(j=0;j<N;j++){
          printf("%d ",matrix[i][j]);
      }
      printf("\n");
  }

  // free memory
  free(matrix);

  return 0;
}
```

Save this code. Compile and run this file.

```
$ gcc -o dtwopointer dtwopointer.c
$ ./dtwopointer
```

A sample output is shown in Figure below.


src — bash — 80×17
agusk$ ls
address dpointer hello.c multiarray switch.c
address.c dpointer.c ifdemo.c multiarray.c while.c
arith.c dtwopointer.c incdec.c pointer
arraydemo for.c inout.c pointer.c
arraydemo.c funcdemo.c logical.c structdemo
comp.c hello mathdemo.c structdemo.c
agusk$ gcc -o dtwopointer dtwopointer.c
agusk$ ./dtwopointer
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
agusk$

4. Functions

This chapter explains how to create function using C.

4.1 Creating Function

Declaring function in C can be written as follows

```
void foo(){
    printf("foo() was called\n");
}
```

We put this function on above main() function. Then, we can call this function, for instance foo().

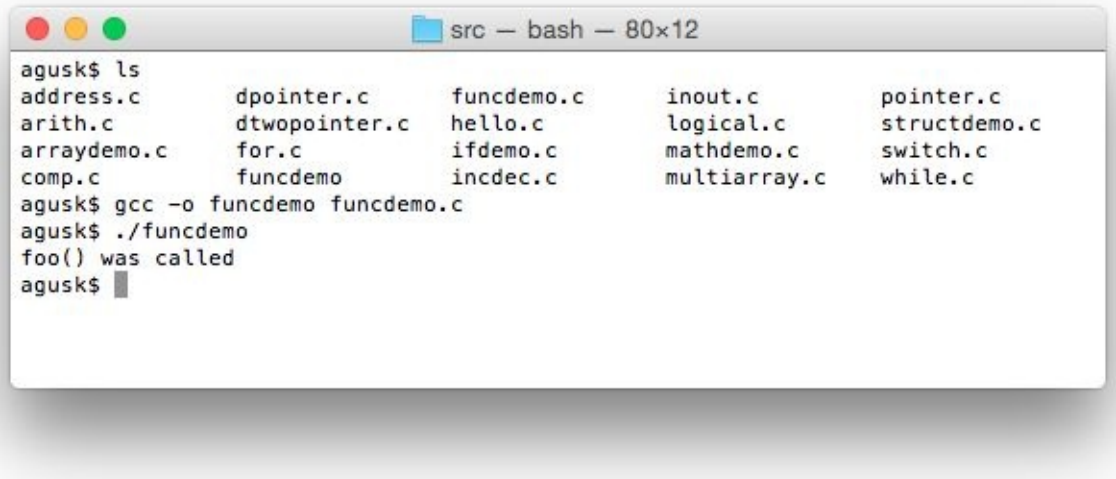
For illustration, we can create a file, called **funcdemo.c** , and write this code.

```
#include <stdio.h>

void foo(){
    printf("foo() was called\n");
}


int main(int argc, const char* argv[]) {

  foo();
  return 0;
}
```

Save this file. Compile and run this program.

```
$ gcc -o funcdemo funcdemo.c
$ ./funcdemo
```

A sample of program output can be seen in Figure below.



We also can declare a function on below of main() function but we must declare our function name.

Add this code on **funcdemo.c** file.

```c
#include <stdio.h>

// implicit declaration for functions
void boo();

int main(int argc, const char* argv[]) {

  boo();
  return 0;
}

void boo(){
    printf("boo() was called\n");
}
```

If we compile and run this code, we will get a response, shown in Figure below.

4.2 Function with Parameters and Returning Value

You may want to create a function that has parameters and a return value. It is easy because you just call *return* into your function.

Add this code in funcdemo.c file. Write this code.

```c
#include <stdio.h>

// implicit declaration for functions
int add(int a, int b);

int main(int argc, const char* argv[]) {

  int result = add(10,5);
  printf("result: %d\n",result);

  return 0;
}

int add(int a, int b){
    return a + b;
}
```

Try to compile and run this file.

4.3 Function with Array Parameters

We also can declare a function with array as parameters. To know how array size, our function should declare array size.

Write this code into your program for illustration.

```c
#include <stdio.h>

// implicit declaration for functions
double mean(int numbers[],int size);

int main(int argc, const char* argv[]) {

  int numbers[8] = {8,4,5,1,4,6,9,6};
  double ret_mean = mean(numbers,8);
  printf("mean: %.2f\n",ret_mean);

  return 0;
}

double mean(int numbers[],int size){
    int i, total;
    double temp;

    for (i = 0; i < size; ++i){
        total += numbers[i];
    }

    temp = (double)total / (double)size;
    return temp;
}
```

Now you can compile and run this program.

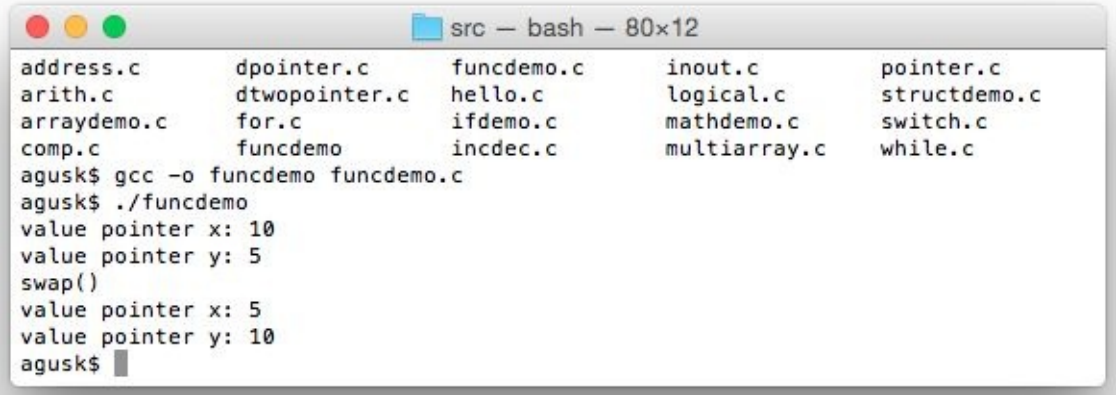
4.4 Function and Pointer

We can pass pointer as parameters in our function. For illustration, we can create swap() to swap our values.

Try to run this program.

```c
#include <stdio.h>

// implicit declaration for functions
void swap(int *px, int *py);

int main(int argc, const char* argv[]) {

  int *x, *y;
  int a, b;

  a = 10;
  b = 5;

  // set value
  x = &a;
  y = &b;

  printf("value pointer x: %d \n",*x);
  printf("value pointer y: %d \n",*y);

  swap(x,y);
  printf("swap()\n");
  printf("value pointer x: %d \n",*x);
  printf("value pointer y: %d \n",*y);
  return 0;
}

void swap(int *px, int *py){
    int temp;

    // store pointer px value to temp
    temp = *px;
    // set pointer px by py value
    *px = *py;
    // set pointer py by temp value
    *py = temp;
}
```

A sample of program output can be seen in Figure below.


src — bash — 80×12
address.c dpointer.c funcdemo.c inout.c pointer.c
arith.c dtwopointer.c hello.c logical.c structdemo.c
arraydemo.c for.c ifdemo.c mathdemo.c switch.c
comp.c funcdemo incdec.c multiarray.c while.c
agusk$ gcc -o funcdemo funcdemo.c
agusk$ ./funcdemo
value pointer x: 10
value pointer y: 5
swap()
value pointer x: 5
value pointer y: 10
agusk$

5. I/O Operations

This chapter explains how to work with I/O operations.

5.1 Getting Started

In this section, we learn how to work with I/O operation. The following is our scenario to illustrate how deal with I/O using C:

- Reading input from keyboard
- Reading program arguments
- Writing data into a file
- Reading data from a file

5.2 Reading Input from Keyboard

We can read input from keyboard in many ways. The following is a list of function for reading input from keyboard:

- getchar()
- gets()
- scanf()
-

We will implement theses functions on next section.

5.2.1 getchar() and putchar() functions

getchar() function is used to get a character from keyboard and putchar() function is used to print a character to Terminal.

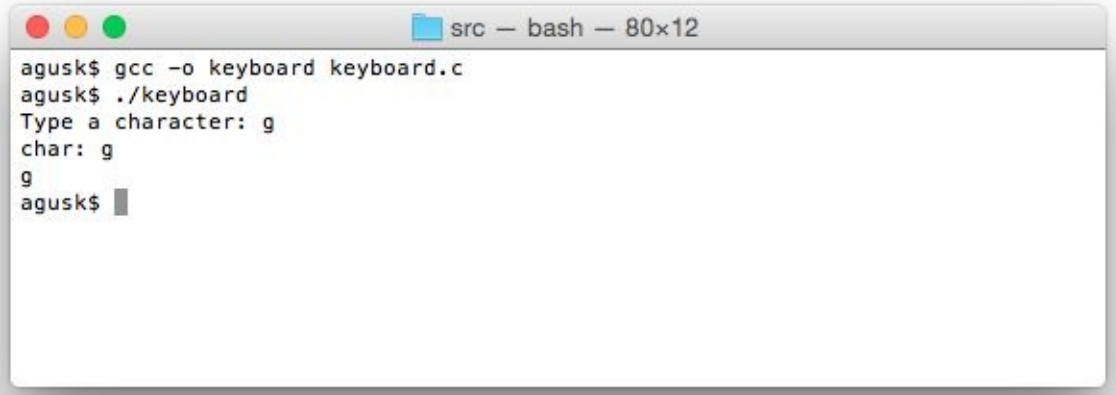
For illustration, create a file, called **keyboard.c**, and write this code.

```
#include <stdio.h>

void getchar_putchar();

int main() {

    getchar_putchar();

    return 0;
}

void getchar_putchar(){
    int c;

    printf ("Type a character: ");
    c = getchar();
    printf ("char: %c\n",c);
    putchar(c);
    printf("\n");
}
```

Compile and run this program.

```
$ gcc -o keyboard keyboard.c
$ ./keyboard
```

You can see the program output in the following Figure below.



5.2.2 gets() and puts() functions

gets() is used to read a text. It will stop to read if you type newline (ENTER key). Unfortunately, gets() is unsafe function(). We can use fgets() to read text with length limitation.

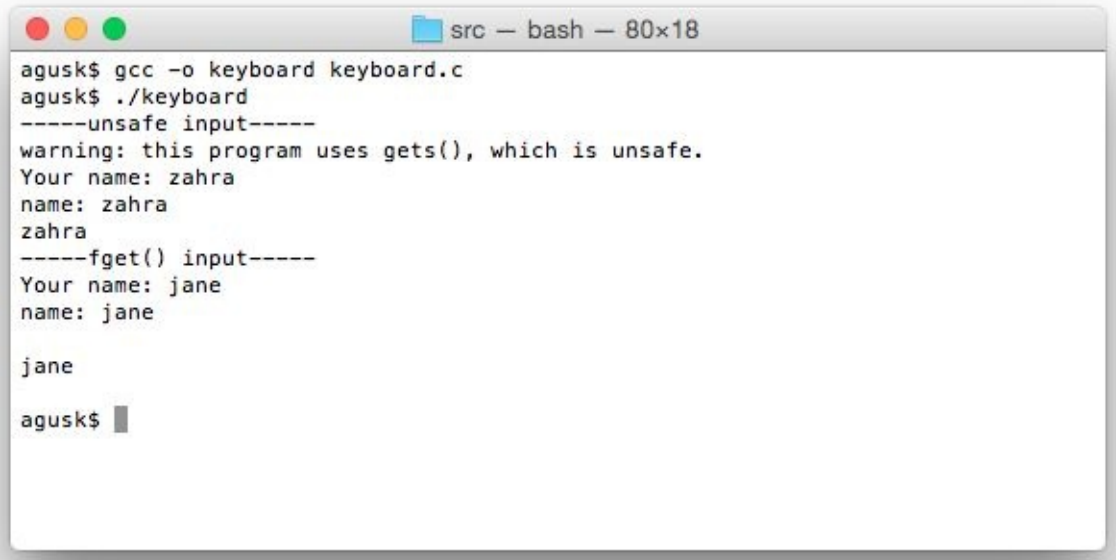
For testing, we add get_puts() function on **keyboard.c** file, and write this code.

```
#include <stdio.h>

void gets_puts();

int main() {

    gets_puts();

    return 0;
}

void gets_puts(){
    printf("-----unsafe input-----\n");
    char name[256];

    printf ("Your name: ");
    gets (name);
    printf ("name: %s\n",name);
    puts(name);

    printf("-----fget() input-----\n");
    name[0] = '\0'; // clear
    printf ("Your name: ");
    fgets(name,256,stdin);
```

```
    printf ("name: %s\n",name);
    puts(name);
}
```

Save this code. Try to compile and run this program.

A sample output can be seen in Figure below.



5.2.3 scanf() function

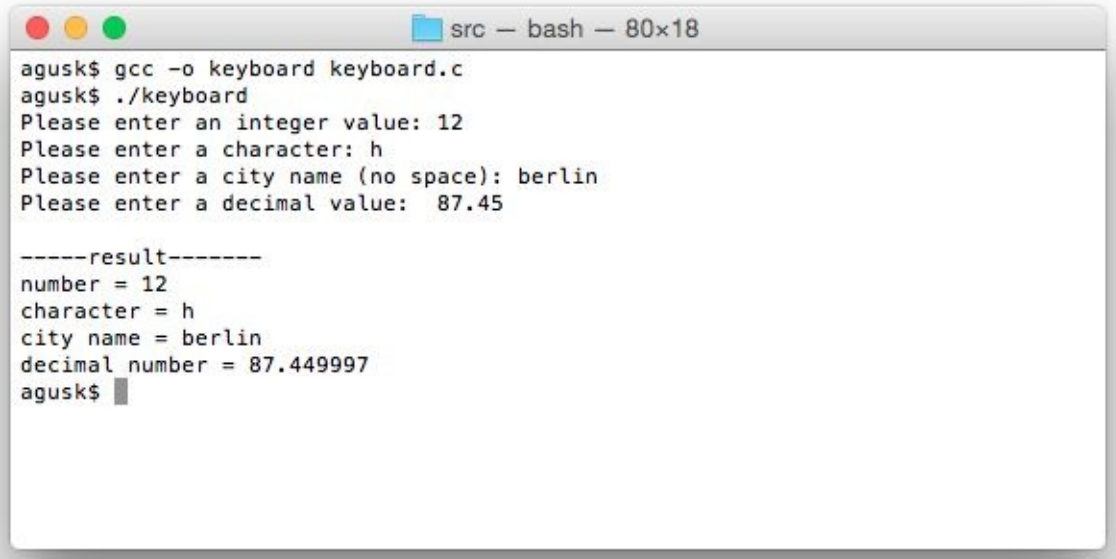
Another approach, we can use scanf() function to read a text.

For illustration, we add scanf() function on **keyboard.c** file, and write this code.

```
#include <stdio.h>

void scanf_demo();

int main() {

    scanf_demo();

    return 0;
}

void scanf_demo(){
    int num;
    char c;
    char city[15];
```

```
    float dec;

    printf("Please enter an integer value: ");
    scanf("%d", &num );

    // %c ignores space characters
    printf("Please enter a character: ");
    scanf(" %c", &c );

    printf("Please enter a city name (no space): ");
    scanf("%s", city );

    printf("Please enter a decimal value:  ");
    scanf("%f", &dec );

    printf("\n-----result-------\n");
    printf("number = %d\n", num );
    printf("character = %c\n", c );
    printf("city name = %s\n", city );
    printf("decimal number = %f\n", dec );
}
```

Save this code. Compile and run this program



5.3 Reading Program Arguments

We can read arguments and its length from our program. This information already pass to main() function.

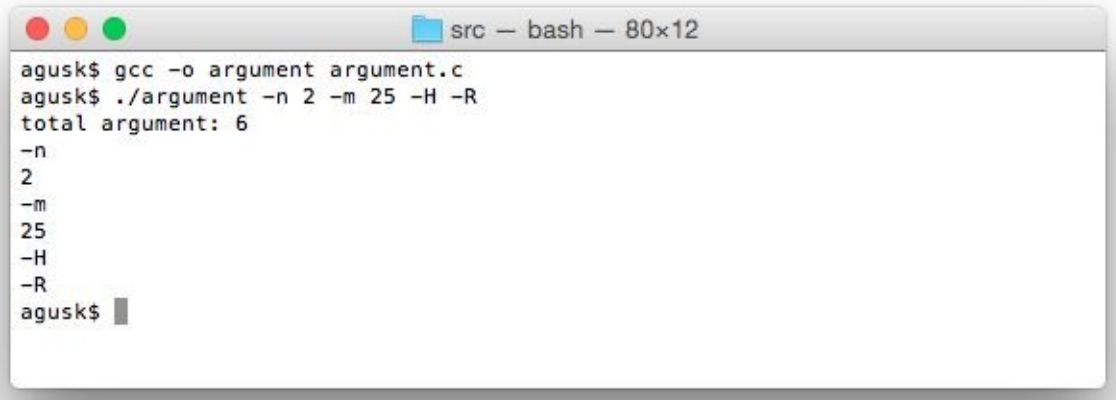
For illustration, create a file, called **argument.c**, and write this code.

```
#include <stdio.h>

int main(int argc, const char* argv[]) {

    int i;

    printf("total argument: %d\n",argc-1);
    if(argc>1){
        for(i=1;i<argc;i++){
            printf("%s\n",argv[i]);
        }
    }


    return 0;
}
```

Compile and run this program.

```
$ gcc -o argument argument.c
$ ./argument
```

A sample output can be seen in Figure below.



5.4 Writing Data Into A File

We can write data into a file. The following is the algorithm for writing data:

- Create a file using fopen() function
- Write data into a file using fprintf() and fputs() functions
- Close a file using fclose() function

For testing, we create data into a file, **demo.txt**. For implementation, create a file, called **filewrite.c**, and write this code.

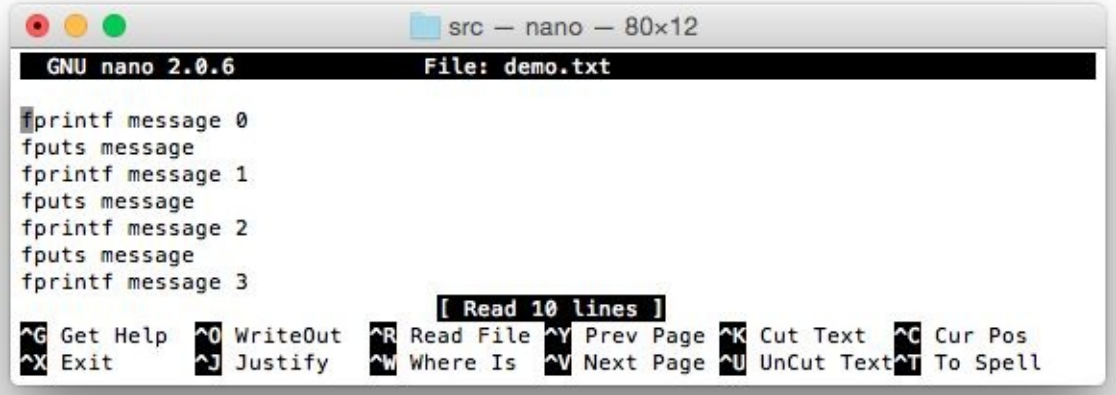
```
#include <stdio.h>

int main(int argc, const char* argv[]) {

    int i;
    FILE *f;

    f = fopen("demo.txt", "w+");

    for(i=0;i<5;i++){
        fprintf(f, "fprintf message %d\n",i);
        fputs("fputs message\n", f); // no format
    }

    fclose(f);
    printf("Data was written into a file\n");

    return 0;
}
```

Save this code. Compile and run this program.

```
$ gcc -o filewrite filewrite.c
$ ./filewrite
```

A sample output can be seen in Figure below.

If success, it will generate **demo.txt**. You can open this file using text editor.



5.5 Reading Data From A File

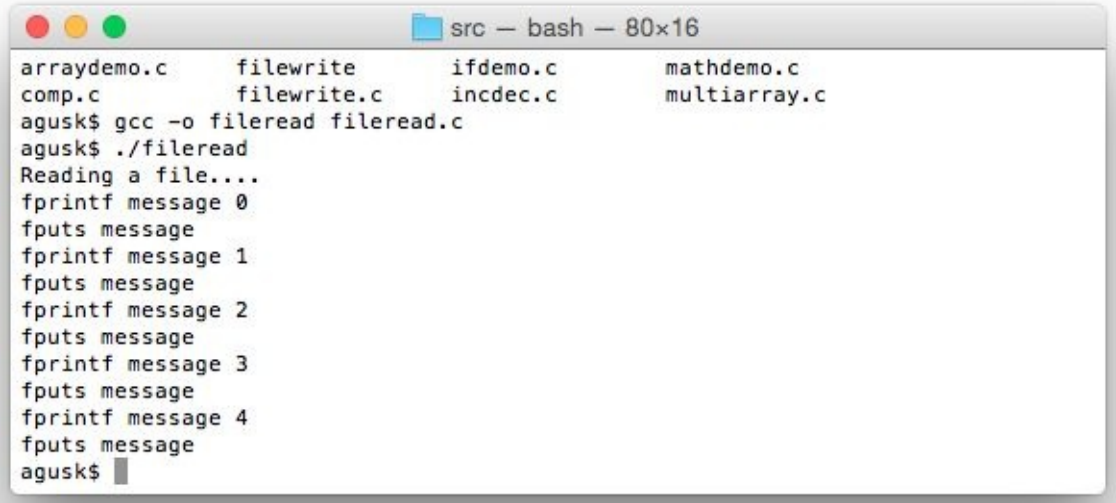
We also can read data from a file using fgetc() function. For testing, create file, called **fileread.c**, and write this code.

```
#include <stdio.h>

int main(int argc, const char* argv[]) {

    char ch;
    FILE *f;

    printf("Reading a file….\n");
    f = fopen("demo.txt", "r");
    if(f==NULL){
        printf("Failed to read file\n");
        return 0;
    }

    while((ch = fgetc(f)) != EOF )
          printf("%c",ch);

    fclose(f);

    return 0;
}
```

Compile and this program.

```
$ gcc -o fileread fileread.c
$ ./fileread
```

A sample output can be seen in Figure below.


src — bash — 80×16
arraydemo.c filewrite ifdemo.c mathdemo.c
comp.c filewrite.c incdec.c multiarray.c
agusk$ gcc -o fileread fileread.c
agusk$ ./fileread
Reading a file....
fprintf message 0
fputs message
fprintf message 1
fputs message
fprintf message 2
fputs message
fprintf message 3
fputs message
fprintf message 4
fputs message
agusk$

6. String Operations

This chapter explains how to work with String operations in C.

6.1 Concatenating Strings

If you have a list of string, you can concatenate into one string. You can use strcat() function from string.h header. For illustration, create a file, called **stringdemo.c**, and write the following code.

```
#include <stdio.h>
#include <string.h>

void concatenating();

int main(int argc, const char* argv[]) {

    concatenating();

    return 0;
}

void concatenating(){
    printf("====concatenating===\n");

    char str1[30] = "hello";
    char str2[10] = "wolrd";

    strcat(str1,str2);
    printf("result: %s\n",str1);

}
```

Compile and run this program.

```
$ gcc -o stringdemo stringdemo.c
$ ./stringdemo
```

A program output can be seen in Figure below.

src — bash — 80×11
address.c demo.txt filewrite.c incdec.c multiarray.c
argument dpointer.c for.c inout.c pointer.c
argument.c dtwopointer.c funcdemo keyboard stringdemo.c
arith.c fileread funcdemo.c keyboard.c structdemo.c
arraydemo.c fileread.c hello.c logical.c switch.c
comp.c filewrite ifdemo.c mathdemo.c while.c
agusk$ gcc -o stringdemo stringdemo.c
agusk$ ./stringdemo
====concatenating===
result: hellowolrd
agusk$

6.2 String To Numeric

Sometime you want to do math operations but input data has String type. To convert String type into numeric, you can use *sscanf()* function for String to numeric. Add string_to_numeric() function on **stringdemo.c** and write this code.

```
#include <stdio.h>
#include <string.h>

void string_to_numeric();

int main(int argc, const char* argv[]) {

    string_to_numeric();

    return 0;
}

void string_to_numeric(){
    printf("====string_to_numeric===\n");
    char str1[10] = "10";
    char str2[10] = "28.74";

    int num1;
    float num2;

    sscanf(str1,"%d",&num1);
    sscanf(str2,"%f",&num2);

    printf("num1: %d\n",num1);
    printf("num2: %f\n",num2);

}
```

Compile and run this program. If success, you get the following of program output.

6.3 Numeric to String

It is easy to convert numeric to String type, you can use sprintf() function. You can get String type automatically. For illustration, add numeric_to_string() function and write this code.

```c
#include <stdio.h>
#include <string.h>

void numeric_to_string();


int main(int argc, const char* argv[]) {

    numeric_to_string();

    return 0;
}

void numeric_to_string(){
    printf("====numeric_to_string===\n");
    int n = 10;
    float m = 23.78;

    char num1[10];
    char num2[10];

    sprintf(num1,"%d",n);
    sprintf(num2,"%.2f",m);

    printf("num1: %s\n",num1);
    printf("num2: %s\n",num2);

}
```

Compile and run this program.

src — bash — 80×11
agusk$ gcc -o stringdemo stringdemo.c
agusk$ ./stringdemo
====numeric_to_string===
num1: 10
num2: 23.78
agusk$

6.4 String Parser

The simple solution to parsing String uses *strtok()* function with delimiter parameter. For example, you have String data with ; delimiter and want to parse it. Here is sample code by adding string_parser() function.

```
#include <stdio.h>
#include <string.h>

void string_parser();


int main(int argc, const char* argv[]) {

    string_parser();

    return 0;
}

void string_parser(){
    char cities[40] = "Tokyo;Berlin;London;New York";
    char token[2]=";";
    char* city;

    printf("cities: %s\n",cities);

    city = strtok(cities, token);
    while(city != NULL){
        printf("%s\n", city );
        city = strtok(NULL, token);
    }

}
```

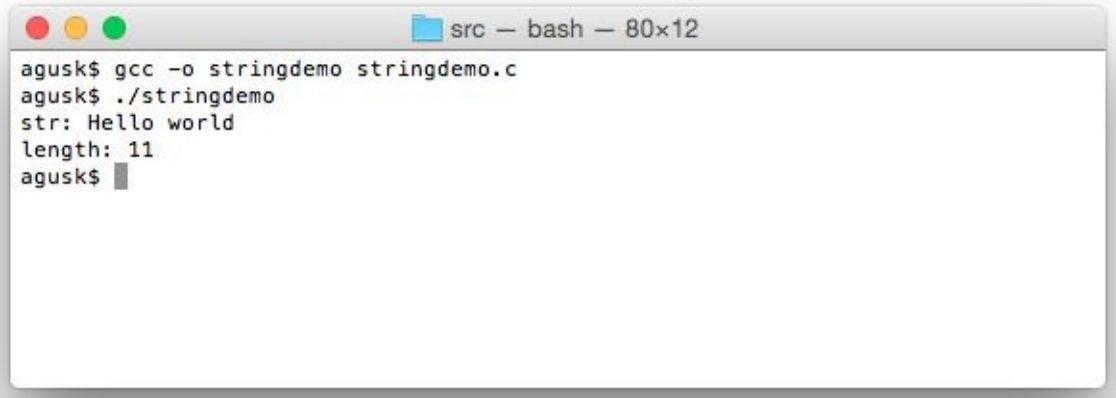
Compile and run this program.

6.5 Check String Data Length

You can use *strlen()* function from string.h header to get the length of string data. For testing, add string_length() function and write this code.

```c
#include <stdio.h>
#include <string.h>

void string_length();

int main(int argc, const char* argv[]) {

    string_length();

    return 0;
}

void string_length(){
    char str[20] = "Hello world";

    printf("str: %s\n",str);
    printf("length: %d\n",(int)strlen(str));

}
```

Compile and run this program.

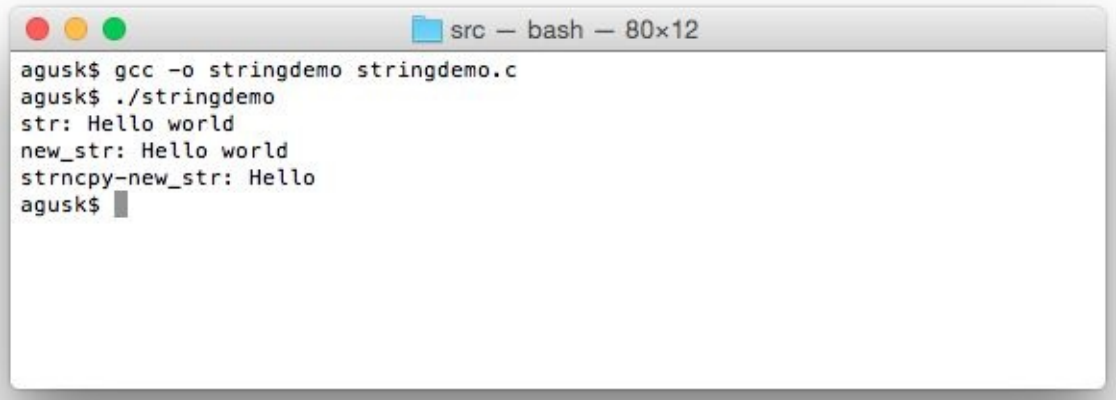


6.6 Copy Data

You may copy some characters from String data. To do it, you can use *strcpy()* and *strncpy()* functions. For illustration, add string_copy() function and write the following code.

```
#include <stdio.h>
#include <string.h>

void string_copy();


int main(int argc, const char* argv[]) {

    string_copy();

    return 0;
}

void string_copy(){
    char str[15] = "Hello world";
    char new_str[20];

    strcpy(new_str,str);
    printf("str: %s\n",str);
    printf("new_str: %s\n",new_str);

    memset(new_str, '\0', sizeof(new_str));
    strncpy(new_str,str,5);
    printf("strncpy-new_str: %s\n",new_str);

}
```

Compile and run this program.



6.7 Exploring Characters

You may get a character by position index. *string_var[index]* syntax provides this feature. Note 0 is the first index.

For testing, add string_explore() function and write this code.

```c
#include <stdio.h>
#include <string.h>

void string_explore();

int main(int argc, const char* argv[]) {

    string_explore();

    return 0;
}

void string_explore(){
    char str[15] = "Hello world";
    int index;

    for(index=0;index<strlen(str);index++){
        printf("%c\n",str[index]);
    }

}
```

Compile and run this program.

7. Building C Library

This chapter explains how to build a library in C.

7.1 Getting Started

Sometimes we create some functions which are used for any application. We can create a library and use it in our C program. In this chapter, we try to build a library using C. There are two library type of library in C, static library and shared library.

7.2 Writing C Library

For illustration, we create a library by creating a file, called **mysimplelib.c**, and write this code.

```c
#include <stdio.h>


int add(int a,int b){

    return a+b;
}
int subtract(int a,int b){

    return a-b;
}
int multiply(int a,int b){

    return a*b;
}
```

Save this program.

After that, we create a header file for our library. Create a file, called **mysimple.h**, and write this code.

```c
/*
header of libmysimple for static and shared libraries
*/

extern int add(int a,int b);
extern int subtract(int a,int b);
extern int multiply(int a,int b);
```

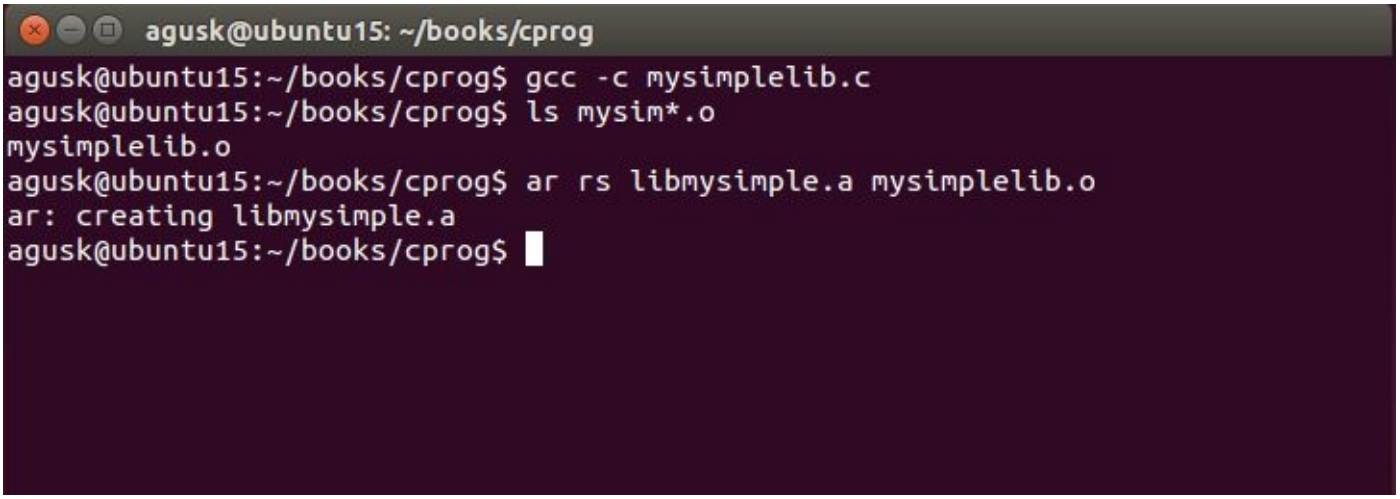
7.3 Compiling and Testing

In this section, we try to compile our library, **mysimplelib.c**, to be static library and shared library.

7.3.1 Static Library

Firstly, we compile our library, **mysimplelib.c**, to be a static library, called **libmysimple.a** .

```
$ gcc -c mysimplelib.c
$ ar rs libmysimple.a mysimplelib.o
```



To access our static library, we create a simple app. Create a file, called **mysimpletest.c**, and write this code.

```
#include <stdio.h>
#include "mysimple.h"

int main(int argc, const char* argv[]) {

    int a,b,c;

    a = 5;
    b = 3;

    c = add(a,b);
    printf("%d + %d = %d\n",a,b,c);

    c = subtract(a,b);
    printf("%d - %d = %d\n",a,b,c);
```

```
    c = multiply(a,b);
    printf("%d * %d = %d\n",a,b,c);

    return 0;
}
```

Consider our static library, **libmysimple.a** , and header file for library, **mysimple.h** are located into the same folder. Now you can compile and run this program.

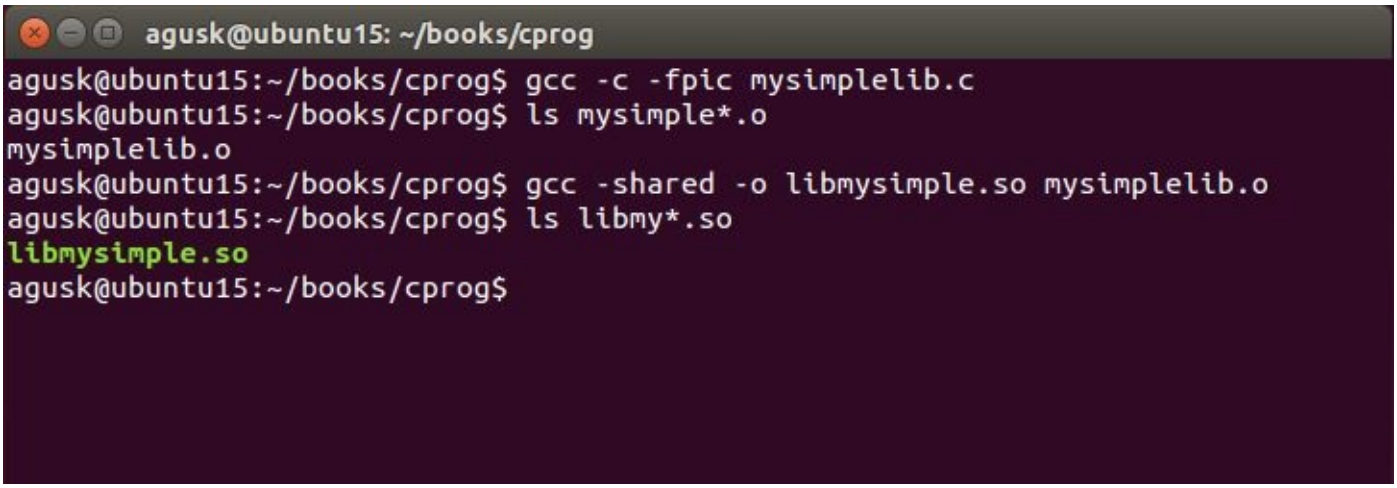
```
$ gcc -I./ -L./ -o mysimpletest mysimpletest.c ./libmysimple.a
$ ./mysimpletest
```

A program output can be seen in Figure below.

7.3.2 Shared Library

We compile our library, **mysimplelib.c**, to be a shared library, called **libmysimple.so** . You can type the following commands.

```
$ gcc -c -fpic mysimplelib.c
$ gcc -shared -o libmysimple.so mysimplelib.o
```



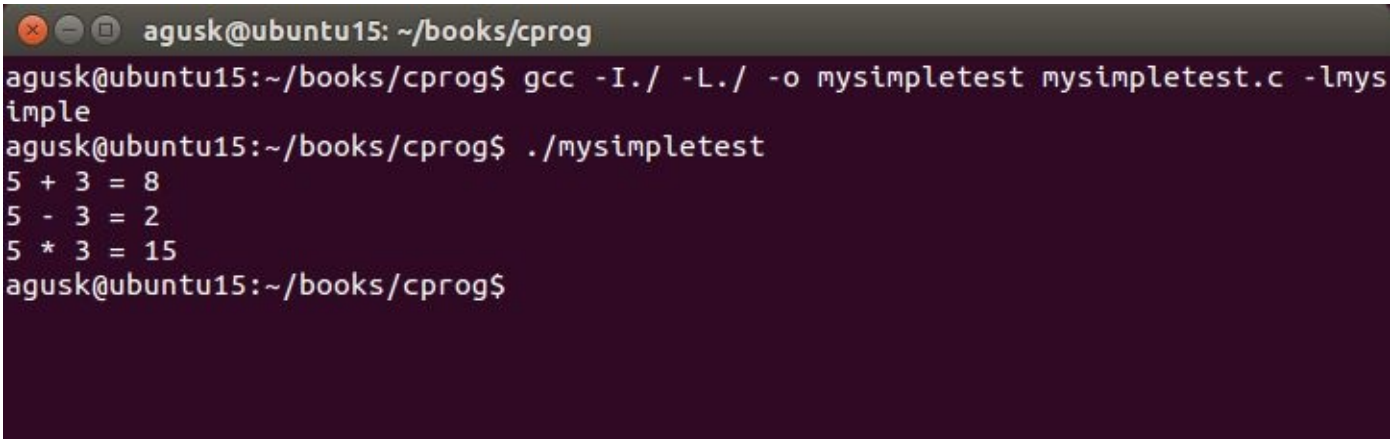
To test our shared library, we can use the same program from **mysimpletest.c** .

Consider share library, **libmysimple.so**, and header file for shared library, **mysimple.h**, are located on the same folder.

You can compile and run our program.

```
$ gcc -I./ -L./ -o mysimpletest mysimpletest.c -lmysimple
$ ./mysimpletest
```

Program output:



You can modify shared library file location. For instance, we put our shared library file into /home/agusk/lib. We must add our library folder into LD_LIBRARY_PATH. Type this command.

```
export LD_LIBRARY_PATH=/home/agusk/lib:$LD_LIBRARY_PATH
```

Then, you can compile and run our program, mysimpletest .

8. Threading

This chapter explains how to work with threading using C.

8.1 Creating Thread

A thread of execution is the smallest unit of processing that a scheduler works on. A process can have multiple threads of execution which are executed asynchronously.

On Linux/Unix, we can create a thread using pthread_create() that can be defined as follows.

```
#include <pthread.h>

int pthread_create (pthread_t *thread,
                const pthread_attr_t *attr,
                void *(*start_routine) (void *),
                void *arg);
```

Note:

- pthread_attr_t is thread attributes
- start_routine is a function that will be executed
- arg is argument that passes to function

For illustration, we create a thread by creating a file, called **createthread.c**. Write this code.

```
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <errno.h>

void* perform(void *arg)
{
  int i;
  int *n = (int *)arg;

  printf("processing from thread\r\n");
  for(i=0;i<(*n);i++)
  {
    printf("%d ",i);
  }
  printf("\r\n");

}

int main(int argc, char* argv[])
{
  pthread_t thread;
  int ret;
  errno = 0;
  int n = 10;
```

```
    ret = pthread_create (&thread, NULL, perform, &n);
    if (ret)
    {
        printf("\n pthread_create() failed with error [%s]\n",strerror(err
        return -1;
    }
    int c = getchar(); // hold app to exit
    return 0;
}
```

You can see we pass parameter n=10 to function perform(). On function perform(), we just do looping.

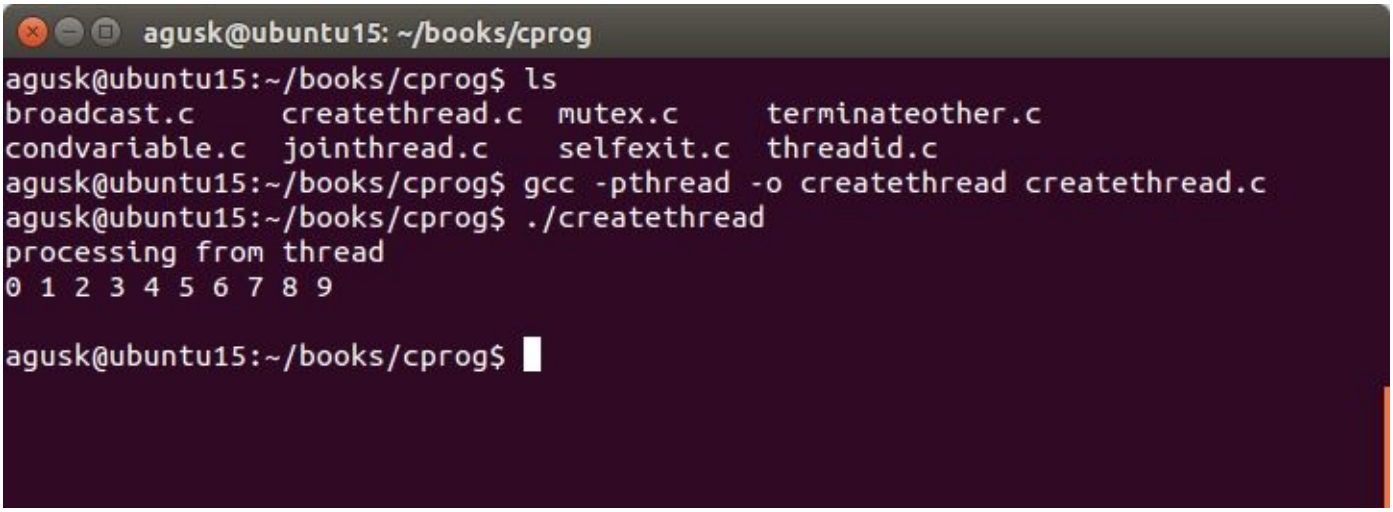
Now save this code. To compile and link, you can type the following command.

```
$ gcc -pthread -o createthread createthread.c
```

If success, you can run this app.

```
$ ./createthread
```

The following is sample output.



8.2 Thread ID

We can obtain the running thread using pthread_self() and following is its syntax.

```
#include <pthread.h>

pthread_t pthread_self (void);
```

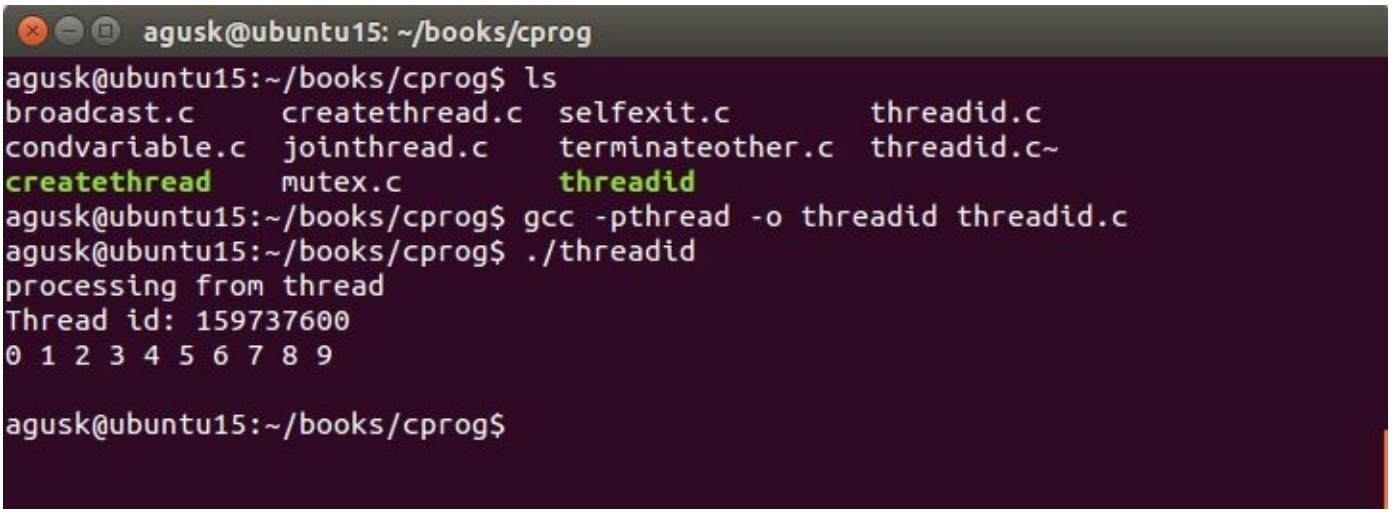
To use this function, you can call pthread_self() inside thread function.

For illustration, create file, called **threadid.c**, and write this code.

```
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <errno.h>

void* perform(void *arg)
{
  int i;
  int *n = (int *)arg;
  pthread_t tid;

  printf("processing from thread\r\n");
  /* get the calling thread's ID */
  tid = pthread_self();
  printf("Thread id: %d \r\n",(int)tid);
  for(i=0;i<(*n);i++)
  {
    printf("%d ",i);
  }
  printf("\r\n");

}

int main(int argc, char* argv[])
{
  pthread_t thread;
  int ret;
  errno = 0;
  int n = 10;

  ret = pthread_create (&thread, NULL, perform, &n);
  if (ret)
  {
      printf("\n pthread_create() failed with error [%s]\n",strerror(err
      return -1;
  }
  int c = getchar(); // hold app to exit
  return 0;
}
```

Save this code. Then try to compile and run it.

The following is sample output.



8.3 Terminating Thread

We can terminate a thread using two approaches, terminating itself and terminating others. We are going to explore these approaches on next section.

8.3.1 Terminating Itself

A thread can stop its processing using pthread_exit(). The following is a syntax of pthread_exit().

```
#include <pthread.h>

void pthread_exit (void *retval);
```

For illustration, we build app based threading and call function perform(). On function perform(), we do looping and will exit from internal thread after looped index 3 by calling pthread_exit().

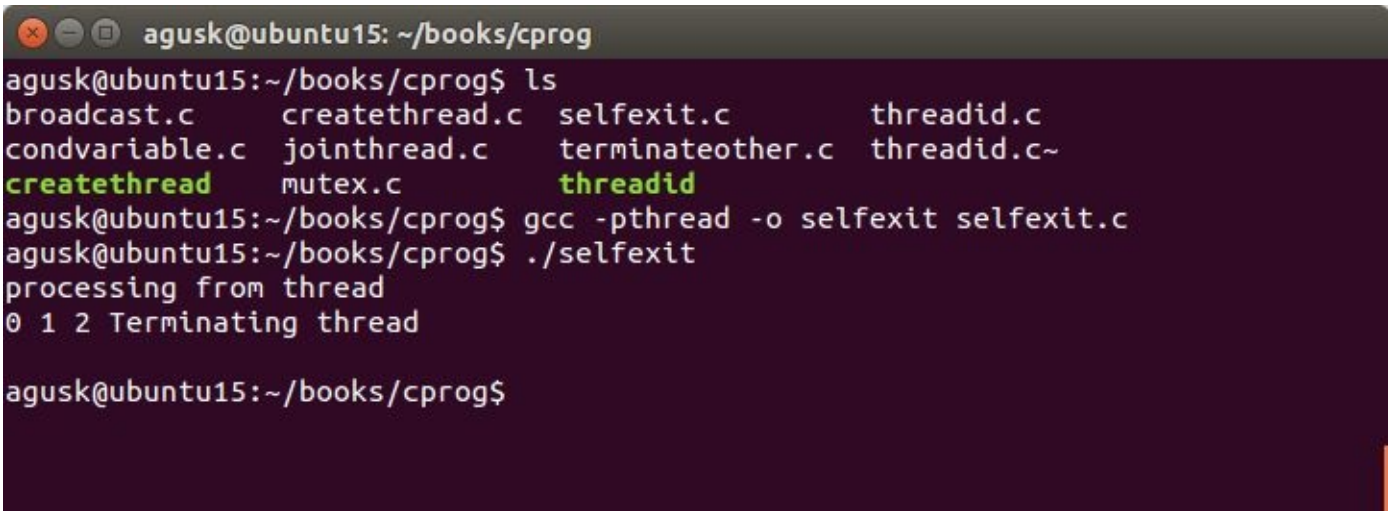
To implement, we create a file, called **selfexit.c**, and write this code.

```
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <errno.h>

void* perform(void *arg)
{
  int i;
  int *n = (int *)arg;

  printf("processing from thread\r\n");
  for(i=0;i<(*n);i++)
  {
    if(i==3)
    {
        printf("Terminating thread\r\n");
        /* exit this thread */
        pthread_exit((void *)0);
    }
    printf("%d ",i);
  }
  printf("\r\n");

}

int main(int argc, char* argv[])
{
```

```
    pthread_t thread;
    int ret;
    errno = 0;
    int n = 10;

    ret = pthread_create (&thread, NULL, perform, &n);
    if (ret)
    {
        printf("\n pthread_create() failed with error [%s]\n",strerror(err
        return -1;
    }
    int c = getchar(); // hold app to exit
    return 0;
}
```

Save this code. Compile and run this code.

The following is a sample output.



8.3.2 Terminating Others

We also can terminate a thread by calling pthread_cancel() with passing thread object.

```
#include <pthread.h>

int pthread_cancel (pthread_t thread);
```

A thread can be configured to enable/disable for canceling thread using pthread_setcancelstate(). pthread_testcancel() function creates a cancellation point in the calling thread. pthread_setcancelstate() and pthread_testcancel() functions can be defined as follows

```
#include <pthread.h>
```

```
int pthread_setcancelstate(int state, int *oldstate);
void pthread_testcancel(void);
```

For illustration, we build app to create a thread and try to terminate it using pthread_cancel().

Create a file, called **terminateother.c**, and write this code.

```
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>

void* perform(void *arg)
{
  int n = 0;

  // I'm not ready to be canceled
  pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, NULL);
  printf("processing from thread\r\n");
  while(1)
  {
    printf("%d ",n);
    n++;

    pthread_testcancel();
    if(n>5)
    {
        // I'm ready to be canceled
        pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
    }

    sleep(1);
  }
  printf("\r\n");

}

int main(int argc, char* argv[])
{
    pthread_t thread;
    int ret, status;
    errno = 0;

    ret = pthread_create (&thread, NULL, perform, NULL);
    if (ret)
    {
      printf("\n pthread_create() failed with error [%s]\n",strerror(err
      return -1;
    }
    sleep(10);
```

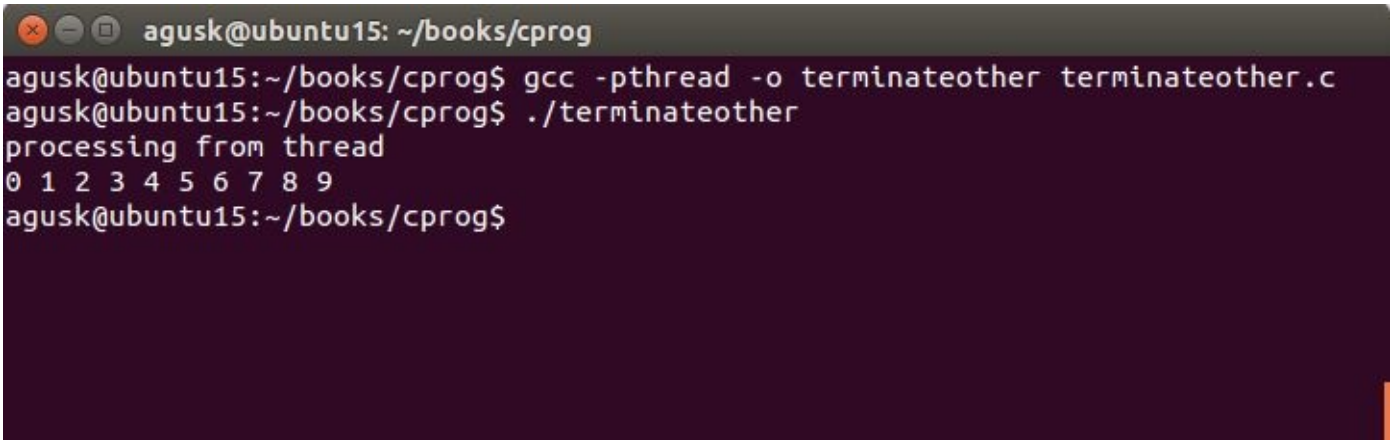
```
    errno = 0;
    status = pthread_cancel(thread);
    if (status)
    {
        printf("\n pthread_cancel() failed with error [%s]\n",strerror(e
        return -1;
    }

    int c = getchar(); // hold app to exit
    return 0;
}
```

Explanation:

- On main entry main(), we create a thread with passing perform() function
- We hold current process using sleep() for 10 seconds
- After that, we try to terminate a thread by calling pthread_cancel()
- pthread_cancel() function returns status value. You can verify this return value
- On perform() function, firstly we call pthread_setcancelstate() to disable the thread cancelling
- After looping n>5, we enable the thread cancelling
- pthread_testcancel() function is called to create a cancellation point

Save this code. Compile and run it.

The following is a sample output.



8.4 Joining Thread

Joining thread is one of thread synchnonization to terminate the executing thread. We can use pthread_join() and defined as follows.

```
#include <pthread.h>

int pthread_join (pthread_t thread, void **retval);
```

We need to pass thread attribute to enable joining thread. We can use pthread_attr_init(), pthread_attr_setdetachstate(), and pthread_attr_destroy() that can be defined as follows.

```
#include <pthread.h>

int pthread_attr_init(pthread_attr_t *attr);
int pthread_attr_destroy(pthread_attr_t *attr);
int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
```

For illustration, we build a simple thread app to compute a simple math equation.

Create a file, called **jointhread.c**, and write this code.

```
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <math.h>


void* compute(void *arg)
{
  int i;
  pthread_t tid;
  double result = 0;

  tid = pthread_self();
  printf("processing from thread %ld \r\n",tid);

  // do something
  for(i=0;i<50;i++)
  {
    result = result + sin(i/2);
  }
  printf("Thread %ld. Completed, result=%e \r\n",tid,result);

  pthread_exit((void *)0);
}

int main(int argc, char* argv[])
```

```
{
    pthread_t thread[5];
    pthread_attr_t attr;
    void *status;
    int i, ret;
    errno = 0;

    // thread attribute
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

    for(i=0;i<5;i++)
    {
        ret = pthread_create (&thread[i], &attr, compute, NULL);
        if (ret)
        {
          printf("\n pthread_create() failed with error [%s]\n",strerror
          return -1;
        }
    }
    sleep(2);

    // joining thread
    printf("joining thread\r\n");
    for(i=0;i<5;i++)
    {
        ret = pthread_join(thread[i], &status);
        if (ret)
        {
          printf("\n pthread_join() failed with error [%s]\n",strerror(e
          return -1;
        }
        printf("Completed join with thread %ld. Status: %ld\n", i,(int)s
    }

    int c = getchar(); // hold app to exit

    return 0;
}
```

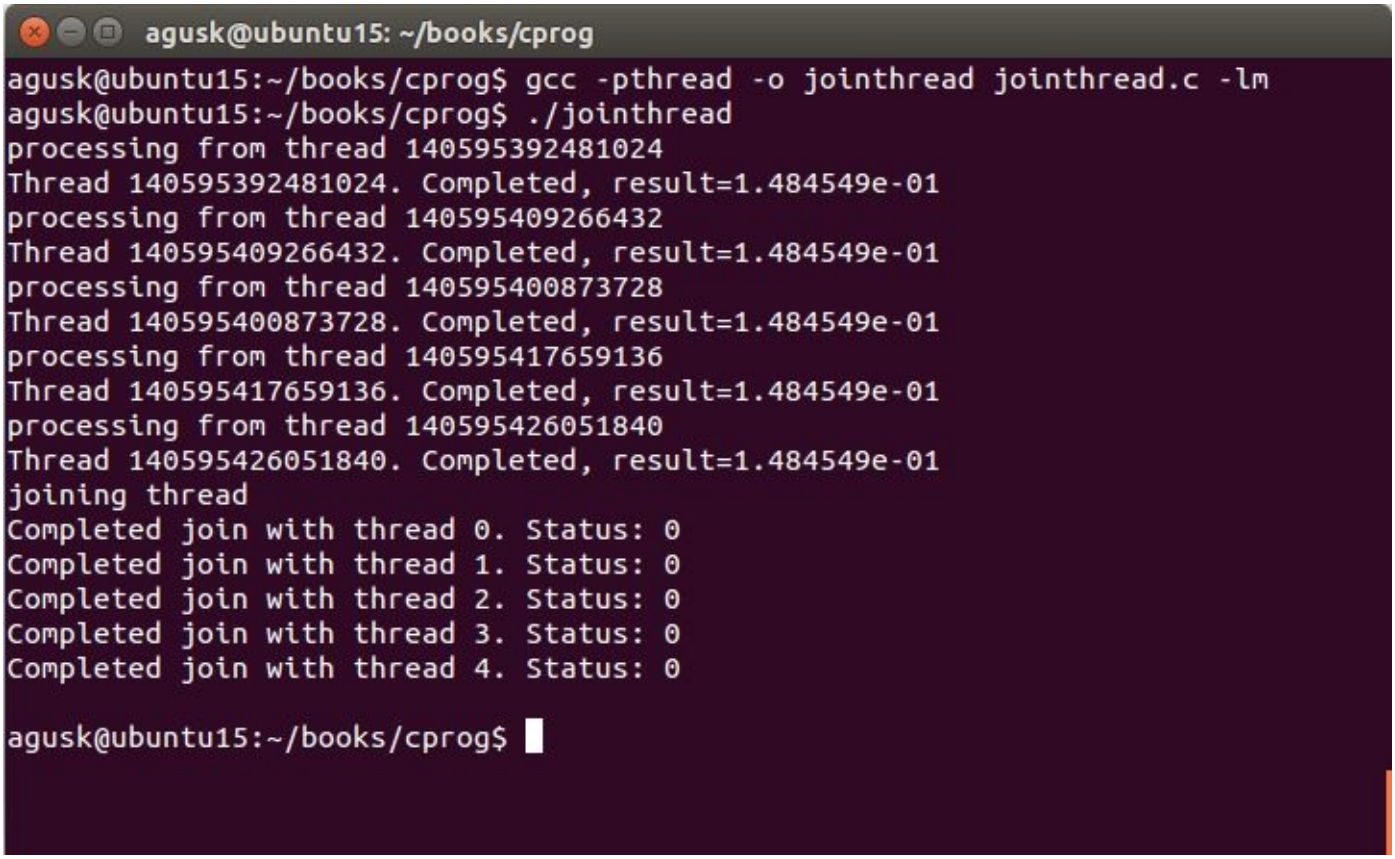
Explanation:

- On main() entry point, we initialize thread attribute using pthread_attr_init() and pthread_attr_setdetachstate() to activate joining thread
- We create 5 threads using pthread_create()
- After 2 seconds, we call pthread_join() to wait 5 exiting threads
- On computer() function, we calculate simple math
- If finished, it call pthread_exit()

Save this code. Because we use math.h, we add library -lm on compiling and linking. The following is a syntax to compile this code.

```
$ gcc -pthread -o jointhread jointhread.c -lm
$ ./jointhread
```

Now you can run it. The sample output is shown Figure below.



8.5 Thread Mutex

You have resources such as data variable, file, database which can be accessed only by a thread. It's called mutex. For illustration, you have data which is stored on variable n. You also have 5 threads to access this data. The first rule is data variable can be accessed by one thread. It means data variable will be locked so another thread cannot access it.

Mutex is one of thread synchronization mechanism to control accessing resource.

You can implement thread mutex using the following functions.

```
#include <pthread.h>

int pthread_mutex_init(pthread_mutex_t *restrict mutex,
                const pthread_mutexattr_t *restrict attr);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Note:

- pthread_mutex_init() initialize mutex object
- pthread_mutex_destroy() destroy mutex object
- pthread_mutex_lock() locak mutex object
- pthread_mutex_lock() unlock/release mutex object

For implementation, we create a file, called **mutex.c**. Firstly we define headers and global variables. Write this code.

```
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>


pthread_t thread[5];
int total_finished_jobs;
pthread_mutex_t lock;
```

Variable total_finished_jobs can be accessed by only one thread.

On main() point entry, we instantiate mutex object and create 5 threads.

```
int main(int argc, char* argv[])
{
    int i, ret;
    errno = 0;
```

```
    total_finished_jobs = 0;

    if (pthread_mutex_init(&lock, NULL) != 0)
    {
        printf("\n pthread_mutex_init() failed with error [%s]\n",strerr
        return 1;
    }

    for(i=0;i<5;i++)
    {
        ret = pthread_create (&thread[i], NULL, perform_job, NULL);
        if (ret)
        {
          printf("\n pthread_create() failed with error [%s]\n",strerror
          return -1;
        }
    }

    int c = getchar(); // hold app to exit
    pthread_mutex_destroy(&lock);

    return 0;
}
```

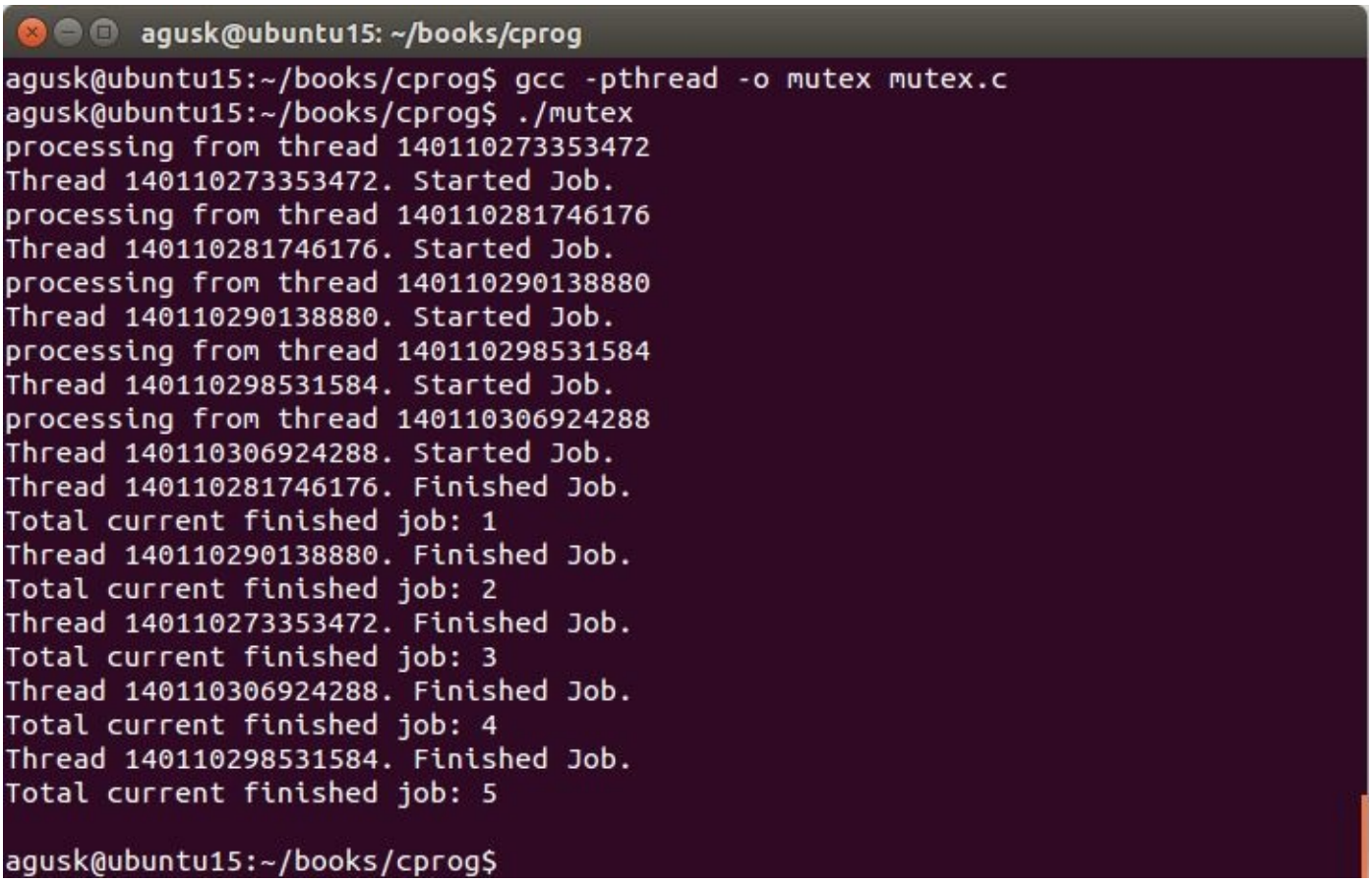
Each thread calls perform_job() function. When access variable total_finished_jobs, we try to lock object lock. After that, we update a value of total_finished_jobs.

The following is implementation of perform_job() function.

```
void* perform_job(void *arg)
{
  int i;
  pthread_t tid;

  tid = pthread_self();
  printf("processing from thread %ld \r\n",tid);
  printf("Thread %ld. Started Job. \r\n",tid);

  // do something for doing job
  for(i=0;i<20;i++)
  {
    sleep(1);
  }

  // update job counter
  pthread_mutex_lock(&lock);
  total_finished_jobs++;
  printf("Thread %ld. Finished Job. \r\n",tid);
  printf("Total current finished job: %d \r\n",total_finished_jobs);
  pthread_mutex_unlock(&lock);

}
```

Now save all code. You can compile and run it.

You can see a sample output on the following Figure.



8.6 Condition Variables

We have learned thread synchronization using thread mutex. Now we explore another thread synchronization using condition variables in controlling accessing resource.

There are two methods to implement condition variables on thread synchronization:

- Signaling
- Broadcasting

Each method will be explained on next section.

8.6.1 Signaling

The idea of signaling is after you access a resource you must notify to another thread to start accessing resource. To implement it, we can use the following functions.

```
#include <pthread.h>

int pthread_cond_init(pthread_cond_t *restrict cond,
        const pthread_condattr_t *restrict attr);

int pthread_cond_wait(pthread_cond_t *restrict cond,
        pthread_mutex_t *restrict mutex);

int pthread_cond_destroy(pthread_cond_t *cond);
```

To notify another thread, you can use pthread_cond_signal() which is defined as below.

```
#include <pthread.h>

int pthread_cond_signal(pthread_cond_t *cond);
```

For illustration, we create two threads that act as producer and consumer. Producer can write data and consumer can only read data. The data can be accessed by one thread, either producer or consumer.

Let's start to create a file, called **condvariable.c**. Firstly we define headers and global variables.

```
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
```

```
#define BUFF_LEN 30

int buffer[BUFF_LEN];
int index_put=0, index_get=0;
int count = 0;
pthread_cond_t empty, get;
pthread_mutex_t lock;
```

On main() entry point, we initialize object mutex and condition variables (empty and fill). We also create 2 thread, producer and consumer.

```
int main(int argc, char* argv[])
{
    pthread_t thread_consumer, thread_producer;
    int ret;
    errno = 0;
    int loops = 15;

    // initialization
    if (pthread_mutex_init(&lock, NULL) != 0)
    {
        printf("\n pthread_mutex_init() failed with error [%s]\n",strerr
        return 1;
    }
    if (pthread_cond_init(&empty, NULL) != 0)
    {
        printf("\n pthread_cond_init() failed with error [%s]\n",strerro
        return 1;
    }
    if (pthread_cond_init(&get, NULL) != 0)
    {
        printf("\n pthread_cond_init() failed with error [%s]\n",strerro
        return 1;
    }

    // create thread
    ret = pthread_create (&thread_consumer, NULL, consumer, &loops);
    if (ret)
    {
      printf("\n pthread_create() failed with error [%s]\n",strerror(err
      return -1;
    }
    ret = pthread_create (&thread_producer, NULL, producer, &loops);
    if (ret)
    {
      printf("\n pthread_create() failed with error [%s]\n",strerror(err
      return -1;
    }

    int c = getchar(); // hold app to exit
    pthread_mutex_destroy(&lock);
    pthread_cond_destroy(&empty);
```

```
    pthread_cond_destroy(&get);

    return 0;
}
```

On producer() function we do looping until obtain signal empty. After obtained a signal, we add a value on array buffer. If done, it will notify signal get.

```
void *producer(void *arg)
{
    int i;
    int *loops = (int *)arg;

    for (i = 0; i < (*loops); i++)
    {
        pthread_mutex_lock(&lock);
        while (count == BUFF_LEN)
            pthread_cond_wait(&empty, &lock);

        buffer[index_put] = i;
        index_put = (index_put + 1) % BUFF_LEN;
        count++;

        pthread_cond_signal(&get);
        pthread_mutex_unlock(&lock);
    }
    printf("exit from producer\r\n");
}
```

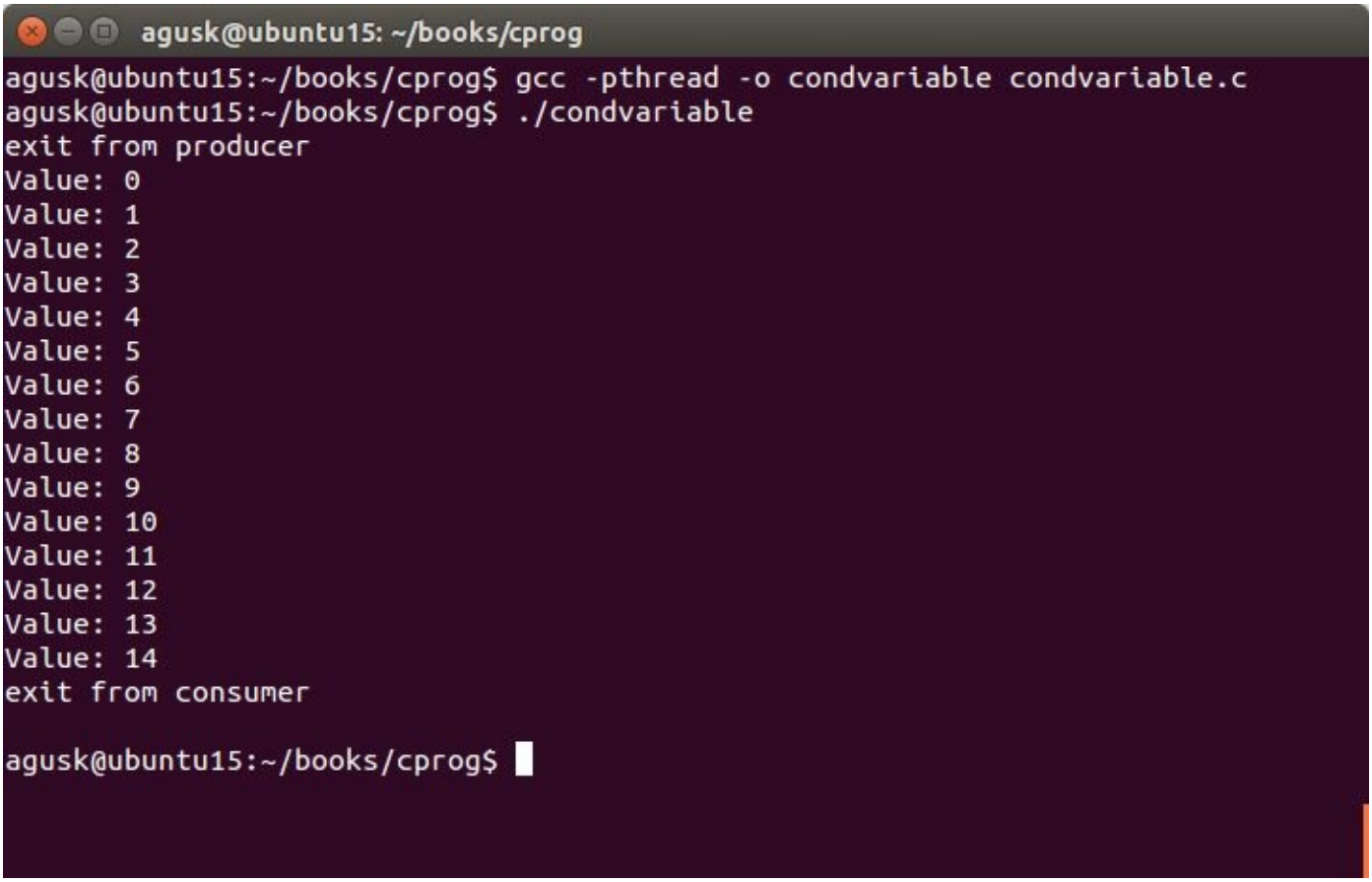
On consumer() function, firstly we wait signal fill. After obtained signal, we read data from array buffer. If done, we notify signal empty.

```
void *consumer(void *arg)
{
    int i;
    int *loops = (int *)arg;

    for (i = 0; i < (*loops); i++)
    {
        pthread_mutex_lock(&lock);
        while (count == BUFF_LEN)
            pthread_cond_wait(&get, &lock);

        int tmp = buffer[index_get];
        index_get = (index_get + 1) % BUFF_LEN;
        count--;
        printf("Value: %d\r\n", tmp);

        pthread_cond_signal(&empty);
        pthread_mutex_unlock(&lock);
```

```
    }
    printf("exit from consumer\r\n");
}
```

Save this code. Compile and run it.

The following is a sample output.



8.6.2 Broadcasting

We can notify some threads using broadcasting signal. It can use pthread_cond_broadcast() and defined as follows.

```
#include <pthread.h>

int pthread_cond_broadcast(pthread_cond_t *cond);
```

What's difference between pthread_cond_broadcast() and pthread_cond_signal()? pthread_cond_broadcast() notifies several threads but pthread_cond_signal() notifies a thread.

For illustration, we use same code on previous section (section 8.6.1). In this scenario, we have 2 consumer threads and 1 producer thread.

To start, we create a file, called **broadcast.c**. Firstly, we define headers and global variables. Write this code.

```
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>

#define BUFF_LEN 30

int buffer[BUFF_LEN];
int index_put=0, index_get=0;
int count = 0;
pthread_cond_t empty, get;
pthread_mutex_t lock;
```

On main() entry point, we initialize mutex and condition variables. Then we create a producer thread and 2 consumer threads.

```
int main(int argc, char* argv[])
{
    pthread_t thread_consumer[2], thread_producer;
    int ret;
    errno = 0;
    int loops = 15;

    // initialization
    if (pthread_mutex_init(&lock, NULL) != 0)
    {
        printf("\n pthread_mutex_init() failed with error [%s]\n",strerr
        return 1;
    }
    if (pthread_cond_init(&empty, NULL) != 0)
    {
        printf("\n pthread_cond_init() failed with error [%s]\n",strerro
        return 1;
    }
    if (pthread_cond_init(&get, NULL) != 0)
    {
        printf("\n pthread_cond_init() failed with error [%s]\n",strerro
        return 1;
    }

    // create thread
    ret = pthread_create (&thread_consumer[0], NULL, consumer, &loops);
    if (ret)
    {
      printf("\n pthread_create() failed with error [%s]\n",strerror(err
      return -1;
    }
    ret = pthread_create (&thread_consumer[1], NULL, consumer, &loops);
```

```
    if (ret)
    {
      printf("\n pthread_create() failed with error [%s]\n",strerror(err
      return -1;
    }
    ret = pthread_create (&thread_producer, NULL, producer, &loops);
    if (ret)
    {
      printf("\n pthread_create() failed with error [%s]\n",strerror(err
      return -1;
    }

    int c = getchar(); // hold app to exit
    pthread_mutex_destroy(&lock);
    pthread_cond_destroy(&empty);
    pthread_cond_destroy(&get);

    return 0;
}
```

On producer() function, we insert data after obtained signal empty. Then we call pthread_cond_broadcast() to notify all consumer threads.

```
void *producer(void *arg)
{
    int i;
    int *loops = (int *)arg;

    for (i = 0; i < (*loops); i++)
    {
        pthread_mutex_lock(&lock);
        while (count == BUFF_LEN)
            pthread_cond_wait(&empty, &lock);

        buffer[index_put] = i;
        index_put = (index_put + 1) % BUFF_LEN;
        count++;

        pthread_cond_broadcast(&get); //broadcast to consumer
        pthread_mutex_unlock(&lock);
    }
    printf("exit from producer\r\n");
}
```

On consumer() function, we get data from array after obtained signal get. After that, we notify producer thread using pthread_cond_signal().

```
void *consumer(void *arg)
{
    int i;
```

```
    int *loops = (int *)arg;
    pthread_t tid;

    tid = pthread_self();
    for (i = 0; i < (*loops); i++)
    {
        pthread_mutex_lock(&lock);
        while (count == BUFF_LEN)
            pthread_cond_wait(&get, &lock);

        int tmp = buffer[index_get];
        index_get = (index_get + 1) % BUFF_LEN;
        count--;
        printf("Thread consumer id: %ld. Value: %d\r\n", tid, tmp);

        pthread_cond_signal(&empty);
        pthread_mutex_unlock(&lock);
    }
    printf("exit from consumer\r\n");
}
```

Save all code. Now you can compile and run it.

The following is a sample output.

```
agusk@ubuntu15: ~/books/cprog
agusk@ubuntu15:~/books/cprog$ gcc -pthread -o broadcast broadcast.c
agusk@ubuntu15:~/books/cprog$ ./broadcast
exit from producer
Thread consumer id: 140317422507776. Value: 0
Thread consumer id: 140317422507776. Value: 1
Thread consumer id: 140317422507776. Value: 2
Thread consumer id: 140317422507776. Value: 3
Thread consumer id: 140317422507776. Value: 4
Thread consumer id: 140317422507776. Value: 5
Thread consumer id: 140317422507776. Value: 6
Thread consumer id: 140317422507776. Value: 7
Thread consumer id: 140317422507776. Value: 8
Thread consumer id: 140317422507776. Value: 9
Thread consumer id: 140317422507776. Value: 10
Thread consumer id: 140317422507776. Value: 11
Thread consumer id: 140317422507776. Value: 12
Thread consumer id: 140317422507776. Value: 13
Thread consumer id: 140317422507776. Value: 14
exit from consumer
Thread consumer id: 140317430900480. Value: 0
Thread consumer id: 140317430900480. Value: 0
Thread consumer id: 140317430900480. Value: 0
Thread consumer id: 140317430900480. Value: 0
Thread consumer id: 140317430900480. Value: 0
Thread consumer id: 140317430900480. Value: 0
Thread consumer id: 140317430900480. Value: 0
Thread consumer id: 140317430900480. Value: 0
Thread consumer id: 140317430900480. Value: 0
Thread consumer id: 140317430900480. Value: 0
Thread consumer id: 140317430900480. Value: 0
Thread consumer id: 140317430900480. Value: 0
Thread consumer id: 140317430900480. Value: 0
Thread consumer id: 140317430900480. Value: 0
Thread consumer id: 140317430900480. Value: 0
Thread consumer id: 140317430900480. Value: 0
exit from consumer

agusk@ubuntu15:~/books/cprog$
```

9. Database Programming

This chapter explains how to create database application using C.

9.1 Database Library for C

C can communicate with database server through database driver. We must install database driver for C before we start to develop database application.

In this chapter, I only focus on MySQL scenarios. Basically, I have already published these codes to Github, https://github.com/agusk/crud-mysql-c , last year . You can download and improve these codes.

9.2 MySQL

To install MySQL server and client in Ubuntu and its database driver for C, you can try to writ this command in terminal

```
$ sudo apt-get install mysql-server mysql-client
$ sudo apt-get install libmysqlclient-dev
```

After finished, we can do testing.

Firstly, we can start to create simple C application to check MySQL version. Write this code

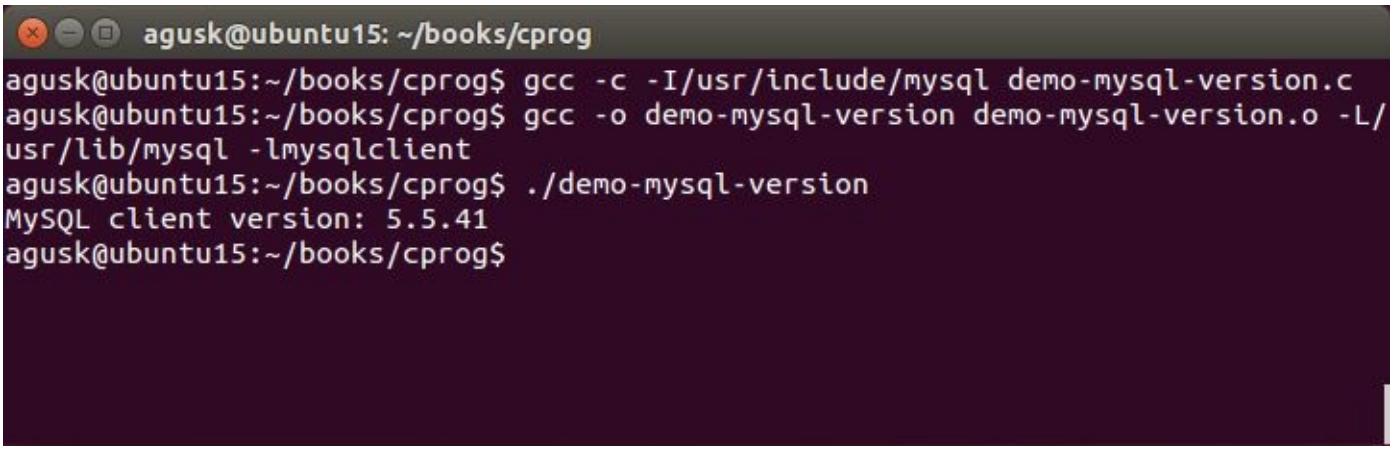
```
#include <my_global.h>
#include <mysql.h>

int main(int argc, char **argv)
{
        printf("MySQL client version: %s\n", mysql_get_client_info());
        exit(0);
}
```

Save this program into a file, called **demo-mysql-version.c** .

Now you can compile and run this file.

```
$ gcc -c -I/usr/include/mysql demo-mysql-version.c
$ gcc -o demo-mysql-version demo-mysql-version.o -L/usr/lib/mysql -lmysq
$ ./demo-mysql-version
```

If success, you will get MySQL client version.



9.3 Connection Test

In this section, we try to test a connection to MySQL server. Create a file, called **demo-connection.c** and write this code.

```
#include <my_global.h>
#include <mysql.h>

int main(int argc, char **argv)
{

        MYSQL *con = mysql_init(NULL);

        if (con == NULL)
        {
          fprintf(stderr, "%s\n", mysql_error(con));
          exit(1);
        }
        printf("connecting to mysql server…\r\n");
        // change host, username, and password
        if (mysql_real_connect(con, "localhost", "root", "password",
                  NULL, 0, NULL, 0) == NULL)
        {
                printf("error: %s\r\n", mysql_error(con));
                mysql_close(con);
                exit(1);
        }
        printf("connected.\r\n");
        printf("closing connection…\r\n");

        mysql_close(con);
        printf("closed.\r\n");

        exit(0);
}
```

Save this program.

Now you can compile and run this file.

```
$ gcc -c -I/usr/include/mysql demo-connection.c
$ gcc -o demo-connection demo-connection.o -L/usr/lib/mysql -lmysqlclien
$ ./demo-connection
```

Program output:

agusk@ubuntu15: ~/books/cprog
agusk@ubuntu15:~/books/cprog$ gcc -c -I/usr/include/mysql demo-connection.c
agusk@ubuntu15:~/books/cprog$ gcc -o demo-connection demo-connection.o -L/usr/li
b/mysql -lmysqlclient
agusk@ubuntu15:~/books/cprog$ ./demo-connection
connecting to mysql server...
connected.
closing connection...
closed.
agusk@ubuntu15:~/books/cprog$

9.4 CRUD (Create, Read, Update and Delete) Operations

In this section, we try to develop simple CRUD operations using MySQL and C.

For testing, we create a database, mydatabase, and a table, called product. Run this SQL script into your MySQL server.

```
CREATE SCHEMA `mydatabase` ;

CREATE  TABLE `mydatabase`.`product` (
  `idproduct` INT NOT NULL AUTO_INCREMENT ,
  `name` VARCHAR(45) NOT NULL ,
  `price` FLOAT NOT NULL ,
  `created` DATETIME NOT NULL ,
  PRIMARY KEY (`idproduct`) );
```

9.4.1 Creating Data

We create data into MySQL. In this case, we create a ten data. Create a file, called **demo-create-data.c** and write this code.

```
#include <my_global.h>
#include <mysql.h>
#include <time.h>
#include <string.h>

#define CREATE_DATA "insert into product(name,price,created) values(?,?,

int main(int argc, char **argv)
{

        MYSQL *con = mysql_init(NULL);

        if (con == NULL)
        {
          fprintf(stderr, "%s\n", mysql_error(con));
          exit(1);
        }
        printf("connecting to mysql server…\r\n");
        // change host, username, and password
        if (mysql_real_connect(con, "localhost", "root", "password",
                  "mydatabase", 0, NULL, 0) == NULL)
        {
                printf("error: %s\r\n", mysql_error(con));
                mysql_close(con);
                exit(1);
        }
        printf("connected.\r\n");
        printf("inserting 10 data…\r\n");
```

```
MYSQL_STMT  *stmt;

stmt = mysql_stmt_init(con);
if (!stmt)
{
        printf(" mysql_stmt_init(), out of memory\r\n");
        exit(0);
}

if (mysql_stmt_prepare(stmt, CREATE_DATA, strlen(CREATE_DATA)))
{
        printf("mysql_stmt_prepare(), INSERT failed\r\n");
        printf("error: %s\r\n", mysql_stmt_error(stmt));
        exit(0);
}

int i;
time_t now = time(NULL);
char name[10];
unsigned long str_length;
float price;
MYSQL_BIND  bind[3];
MYSQL_TIME  ts;

memset(bind, 0, sizeof(bind));

bind[0].buffer_type= MYSQL_TYPE_VAR_STRING;
bind[0].buffer= (char *)&name;
bind[0].is_null= 0;
bind[0].length= &str_length;

bind[1].buffer_type= MYSQL_TYPE_FLOAT;
bind[1].buffer= (char *)&price;
bind[1].is_null= 0;
bind[1].length= 0;

bind[2].buffer_type= MYSQL_TYPE_DATETIME;
bind[2].buffer= (char *)&ts;
bind[2].is_null= 0;
bind[2].length= 0;

// bind parameters
if (mysql_stmt_bind_param(stmt, bind))
{
        printf("mysql_stmt_bind_param() failed\r\n");
        printf("%s\r\n", mysql_stmt_error(stmt));
        exit(0);
}

for(i=1;i<=10;i++)
{
        // set values
```

```
                sprintf(name,"product-%d",i);
                str_length = strlen(name);
                price = 0.23*i;

                // time_t to MYSQL_TIME
                struct tm *now_struct = gmtime(&now);
                ts.year = now_struct->tm_year + 1900;
                ts.month = now_struct->tm_mon + 1;
                ts.day = now_struct->tm_mday;
                ts.hour = now_struct->tm_hour;
                ts.minute = now_struct->tm_min;
                ts.second = now_struct->tm_sec;

                printf("executing data %d…\r\n",i);
                if (mysql_stmt_execute(stmt))
                {
                        printf("mysql_stmt_execute(), 1 failed\r\n");
                        printf("%s\r\n", mysql_stmt_error(stmt));
                        exit(0);
                }

        }
        printf("done.\r\n");
        printf("closing connection…\r\n");

        /* Close the statement */
        if (mysql_stmt_close(stmt))
        {
                printf("failed while closing the statement\r\n");
                printf("%s\r\n", mysql_stmt_error(stmt));
                exit(0);
        }
        mysql_close(con);
        printf("closed.\r\n");

        exit(0);
}
```

Save this code.

Now you can compile and run this file.

```
$ gcc -c -I/usr/include/mysql demo-create-data.c
$ gcc -o demo-create-data demo-create-data.o -L/usr/lib/mysql -lmysqlcli
$ ./demo-create-data
```

Program output:

```
agusk@ubuntu15: ~/books/cprog
agusk@ubuntu15:~/books/cprog$ gcc -c -I/usr/include/mysql demo-create-data.c
agusk@ubuntu15:~/books/cprog$ gcc -o demo-create-data demo-create-data.o -L/usr/
lib/mysql -lmysqlclient
agusk@ubuntu15:~/books/cprog$ ./demo-create-data
connecting to mysql server...
connected.
inserting 10 data...
executing data 1...
executing data 2...
executing data 3...
executing data 4...
executing data 5...
executing data 6...
executing data 7...
executing data 8...
executing data 9...
executing data 10...
done.
closing connection...
closed.
agusk@ubuntu15:~/books/cprog$
```

9.4.2 Reading Data

In this section, we try to read data from MySQL. For illustration, create a file, called **demo-read-data.c** and write this code.

```c
#include <my_global.h>
#include <mysql.h>

int main(int argc, char **argv)
{

        MYSQL *con = mysql_init(NULL);

        if (con == NULL)
        {
          fprintf(stderr, "%s\n", mysql_error(con));
          exit(1);
        }
        printf("connecting to mysql server…\r\n");
        // change host, username, and password
        if (mysql_real_connect(con, "localhost", "root", "password",
                  "mydatabase", 0, NULL, 0) == NULL)
        {
                printf("error: %s\r\n", mysql_error(con));
                mysql_close(con);
                exit(1);
        }
        printf("connected.\r\n");
```

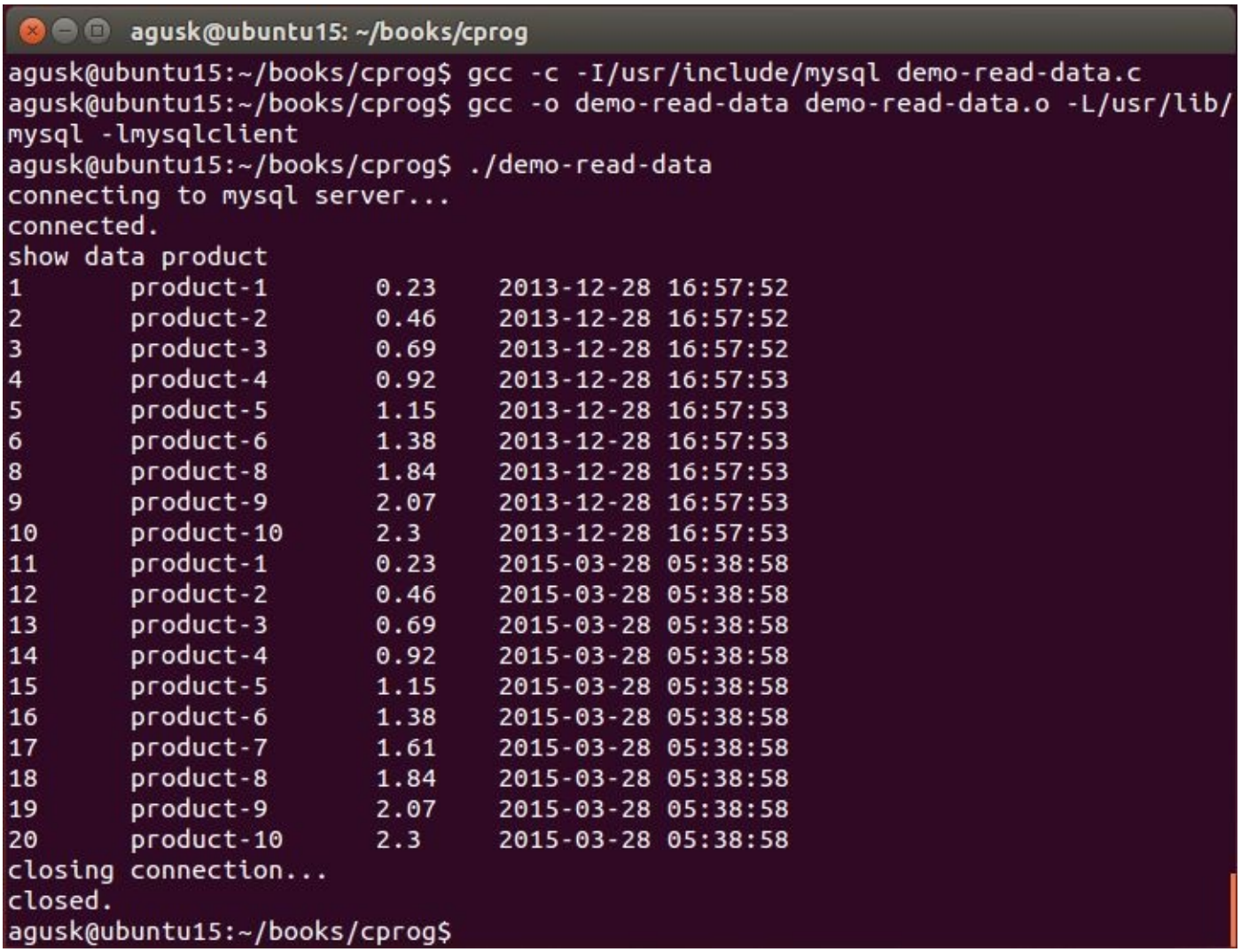
```
        printf("show data product\r\n");
        if (mysql_query(con, "SELECT * FROM product"))
        {
                printf("error: %s\r\n", mysql_error(con));
                mysql_close(con);
                exit(1);
        }

        MYSQL_RES *result = mysql_store_result(con);
        if (result == NULL)
        {
                printf("error: %s\r\n", mysql_error(con));
                mysql_close(con);
                exit(1);
        }

        int num_fields = mysql_num_fields(result);
        MYSQL_ROW row;
        int i;
        while ((row = mysql_fetch_row(result)))
        {
          for(i = 0; i < num_fields; i++)
          {
                  printf("%s\t", row[i] ? row[i] : "NULL");
          }
          printf("\r\n");
        }

        mysql_free_result(result);

        printf("closing connection…\r\n");

        mysql_close(con);
        printf("closed.\r\n");

        exit(0);
}
```

Save this code.

Now you can compile and run this file.

```
$ gcc -c -I/usr/include/mysql demo-read-data.c
$ gcc -o demo-read-data demo-read-data.o -L/usr/lib/mysql -lmysqlclient
$ ./demo-read-data
```

Program output:



9.4.3 Updating Data

In this section, we try to update data, for instance, we update data with id=5. Create a file, called **demo-update-data.c** and write this code.

```
#include <my_global.h>
#include <mysql.h>
#include <string.h>

#define UPDATE_DATA "update product set name = ?, price=? where idproduc

int main(int argc, char **argv)
{

        MYSQL *con = mysql_init(NULL);

        if (con == NULL)
        {
          fprintf(stderr, "%s\n", mysql_error(con));
          exit(1);
        }
```

```
        printf("connecting to mysql server…\r\n");
        // change host, username, and password
        if (mysql_real_connect(con, "localhost", "root", "password",
                "mydatabase", 0, NULL, 0) == NULL)
        {
                printf("error: %s\r\n", mysql_error(con));
                mysql_close(con);
                exit(1);
        }
        printf("connected.\r\n");

        MYSQL_STMT  *stmt;

        stmt = mysql_stmt_init(con);
        if (!stmt)
        {
                printf(" mysql_stmt_init(), out of memory\r\n");
                exit(0);
        }

        if (mysql_stmt_prepare(stmt, UPDATE_DATA, strlen(UPDATE_DATA)))
        {
                printf("mysql_stmt_prepare(), UPDATE failed\r\n");
                printf("error: %s\r\n", mysql_stmt_error(stmt));
                exit(0);
        }

        int i;
        char name[10];
        unsigned long str_length;
        float price;
        MYSQL_BIND  bind[3];
        int product_id;

        memset(bind, 0, sizeof(bind));

        bind[0].buffer_type= MYSQL_TYPE_VAR_STRING;
        bind[0].buffer= (char *)&name;
        bind[0].is_null= 0;
        bind[0].length= &str_length;

        bind[1].buffer_type= MYSQL_TYPE_FLOAT;
        bind[1].buffer= (char *)&price;
        bind[1].is_null= 0;
        bind[1].length= 0;

        bind[2].buffer_type= MYSQL_TYPE_LONG;
        bind[2].buffer= (char *)&product_id;
        bind[2].is_null= 0;
        bind[2].length= 0;

        // bind parameters
        if (mysql_stmt_bind_param(stmt, bind))
```

```
        {
                printf("mysql_stmt_bind_param() failed\r\n");
                printf("%s\r\n", mysql_stmt_error(stmt));
                exit(0);
        }

        // set updated values
        // change these values!!
        sprintf(name,"product-updated");
        str_length = strlen(name);
        price = 10.33;
        product_id = 5;

        printf("updating data…\r\n");
        if (mysql_stmt_execute(stmt))
        {
                printf("mysql_stmt_execute(), 1 failed\r\n");
                printf("%s\r\n", mysql_stmt_error(stmt));
                exit(0);
        }
        printf("done.\r\n");
        printf("closing connection…\r\n");

        /* Close the statement */
        if (mysql_stmt_close(stmt))
        {
                printf("failed while closing the statement\r\n");
                printf("%s\r\n", mysql_stmt_error(stmt));
                exit(0);
        }
        mysql_close(con);
        printf("closed.\r\n");

        exit(0);
}
```

Save this code.

Now you can compile and run this file.

```
$ gcc -c -I/usr/include/mysql demo-update-data.c
$ gcc -o demo-update-data demo-update-data.o -L/usr/lib/mysql -lmysqlcli
$ ./demo-update-data
```

Program output:

```
agusk@ubuntu15: ~/books/cprog
agusk@ubuntu15:~/books/cprog$ gcc -c -I/usr/include/mysql demo-update-data.c
agusk@ubuntu15:~/books/cprog$ gcc -o demo-update-data demo-update-data.o -L/usr/
lib/mysql -lmysqlclient
agusk@ubuntu15:~/books/cprog$ ./demo-update-data
connecting to mysql server...
connected.
updating data...
done.
closing connection...
closed.
agusk@ubuntu15:~/books/cprog$ ./demo-read-data
connecting to mysql server...
connected.
show data product
1       product-1       0.23    2013-12-28 16:57:52
2       product-2       0.46    2013-12-28 16:57:52
3       product-3       0.69    2013-12-28 16:57:52
4       product-4       0.92    2013-12-28 16:57:53
5       product-updated 10.33   2013-12-28 16:57:53
6       product-6       1.38    2013-12-28 16:57:53
8       product-8       1.84    2013-12-28 16:57:53
9       product-9       2.07    2013-12-28 16:57:53
10      product-10      2.3     2013-12-28 16:57:53
11      product-1       0.23    2015-03-28 05:38:58
12      product-2       0.46    2015-03-28 05:38:58
13      product-3       0.69    2015-03-28 05:38:58
14      product-4       0.92    2015-03-28 05:38:58
15      product-5       1.15    2015-03-28 05:38:58
16      product-6       1.38    2015-03-28 05:38:58
17      product-7       1.61    2015-03-28 05:38:58
18      product-8       1.84    2015-03-28 05:38:58
19      product-9       2.07    2015-03-28 05:38:58
20      product-10      2.3     2015-03-28 05:38:58
closing connection...
closed.
agusk@ubuntu15:~/books/cprog$
```

9.4.4 Deleting Data

The last operation is to delete data, for instance, delete data with id=3. Create a file, called **demo-delete-data.c** and write this code.

```c
#include <my_global.h>
#include <mysql.h>
#include <string.h>

#define DELETE_DATA "delete from product where idproduct=?"

int main(int argc, char **argv)
{

        MYSQL *con = mysql_init(NULL);
```

```
if (con == NULL)
{
  fprintf(stderr, "%s\n", mysql_error(con));
  exit(1);
}
printf("connecting to mysql server…\r\n");
// change host, username, and password
if (mysql_real_connect(con, "localhost", "root", "password",
          "mydatabase", 0, NULL, 0) == NULL)
{
        printf("error: %s\r\n", mysql_error(con));
        mysql_close(con);
        exit(1);
}
printf("connected.\r\n");

MYSQL_STMT  *stmt;

stmt = mysql_stmt_init(con);
if (!stmt)
{
        printf(" mysql_stmt_init(), out of memory\r\n");
        exit(0);
}

if (mysql_stmt_prepare(stmt, DELETE_DATA, strlen(DELETE_DATA)))
{
        printf("mysql_stmt_prepare(), UPDATE failed\r\n");
        printf("error: %s\r\n", mysql_stmt_error(stmt));
        exit(0);
}

int i;
MYSQL_BIND  bind[1];
int product_id;

memset(bind, 0, sizeof(bind));

bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&product_id;
bind[0].is_null= 0;
bind[0].length= 0;

// bind parameters
if (mysql_stmt_bind_param(stmt, bind))
{
        printf("mysql_stmt_bind_param() failed\r\n");
        printf("%s\r\n", mysql_stmt_error(stmt));
        exit(0);
}

// select product id to be deleted
```

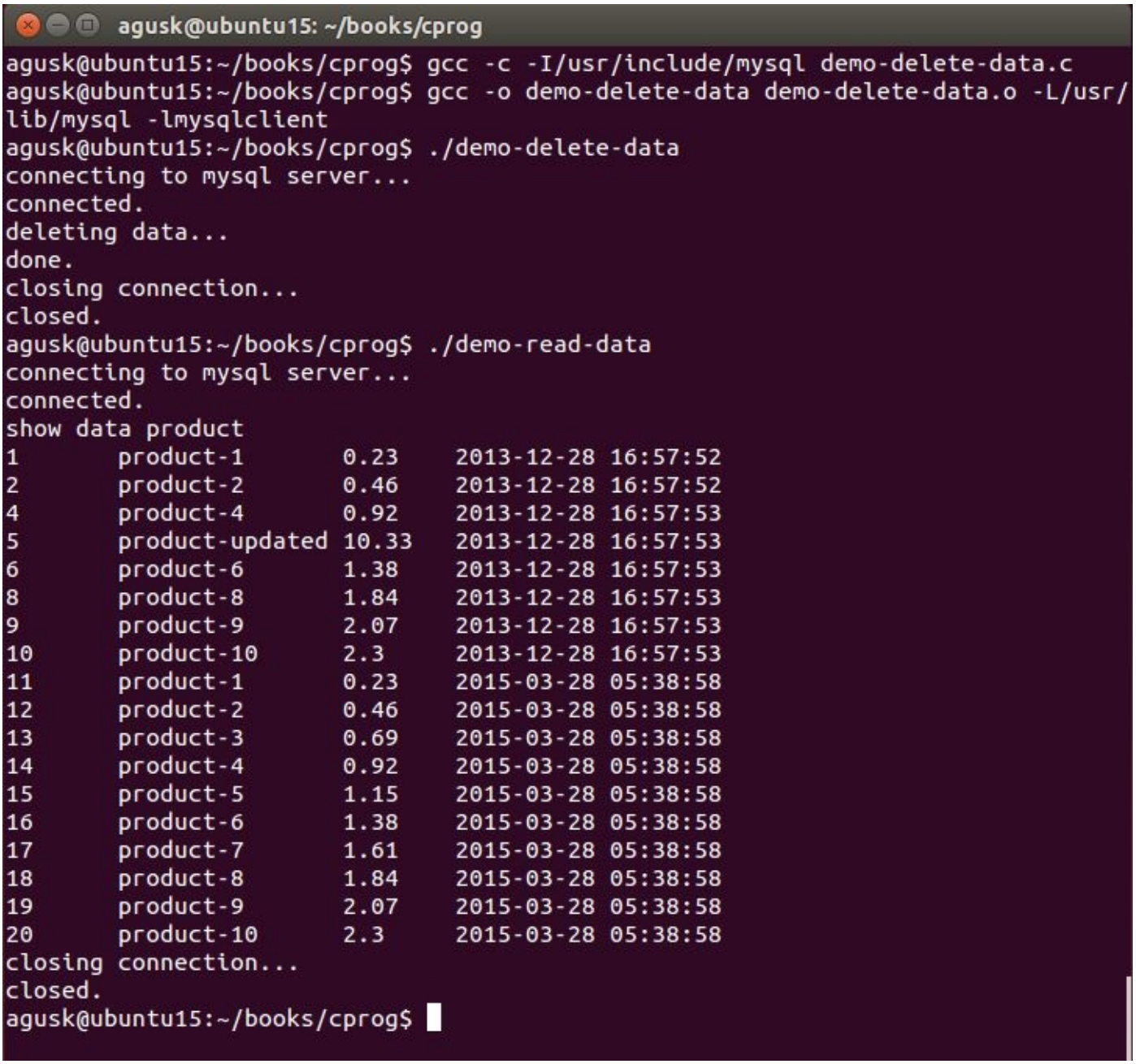
```
        // change these values!!
        product_id = 3;

        printf("deleting data…\r\n");
        if (mysql_stmt_execute(stmt))
        {
                printf("mysql_stmt_execute(), 1 failed\r\n");
                printf("%s\r\n", mysql_stmt_error(stmt));
                exit(0);
        }
        printf("done.\r\n");
        printf("closing connection…\r\n");

        /* Close the statement */
        if (mysql_stmt_close(stmt))
        {
                printf("failed while closing the statement\r\n");
                printf("%s\r\n", mysql_stmt_error(stmt));
                exit(0);
        }
        mysql_close(con);
        printf("closed.\r\n");

        exit(0);
}
```

Save this code.

Now you can compile and run this file.

```
$ gcc -c -I/usr/include/mysql demo-delete-data.c
$ gcc -o demo-delete-data demo-delete-data.o -L/usr/lib/mysql -lmysqlcli
$ ./demo-delete-data
```

Program output:


agusk@ubuntu15: ~/books/cprog
agusk@ubuntu15:~/books/cprog$ gcc -c -I/usr/include/mysql demo-delete-data.c
agusk@ubuntu15:~/books/cprog$ gcc -o demo-delete-data demo-delete-data.o -L/usr/
lib/mysql -lmysqlclient
agusk@ubuntu15:~/books/cprog$ ./demo-delete-data
connecting to mysql server...
connected.
deleting data...
done.
closing connection...
closed.
agusk@ubuntu15:~/books/cprog$ ./demo-read-data
connecting to mysql server...
connected.
show data product
1 product-1 0.23 2013-12-28 16:57:52
2 product-2 0.46 2013-12-28 16:57:52
4 product-4 0.92 2013-12-28 16:57:53
5 product-updated 10.33 2013-12-28 16:57:53
6 product-6 1.38 2013-12-28 16:57:53
8 product-8 1.84 2013-12-28 16:57:53
9 product-9 2.07 2013-12-28 16:57:53
10 product-10 2.3 2013-12-28 16:57:53
11 product-1 0.23 2015-03-28 05:38:58
12 product-2 0.46 2015-03-28 05:38:58
13 product-3 0.69 2015-03-28 05:38:58
14 product-4 0.92 2015-03-28 05:38:58
15 product-5 1.15 2015-03-28 05:38:58
16 product-6 1.38 2015-03-28 05:38:58
17 product-7 1.61 2015-03-28 05:38:58
18 product-8 1.84 2015-03-28 05:38:58
19 product-9 2.07 2015-03-28 05:38:58
20 product-10 2.3 2015-03-28 05:38:58
closing connection...
closed.
agusk@ubuntu15:~/books/cprog$

10. Socket Programming

This chapter explains how to get started with Socket programming using C.

10.1 Getting Local Hostname

We usually know our hostname using command hostname. For instance, you can type the following command.

```
$ hostname
```

We also can get our hostname by program. We can use gethostname() function. It is defined as below.

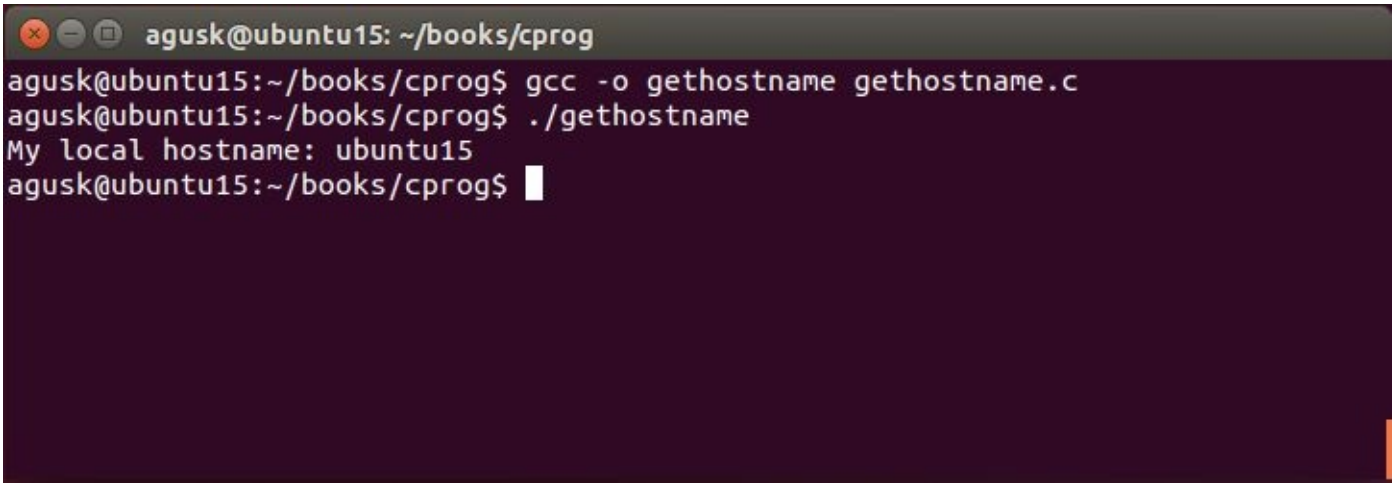
```
#include <unistd.h>

int gethostname(char *name, size_t len);
```

For illustration, we create a simple app to read local hostname. Create a file, called **gethostname.c**, and write this code.

```
#include <stdio.h>
#include <sys/unistd.h>


int main(void)
{
    char hostname[128];

    gethostname(hostname, sizeof(hostname));
    printf("My local hostname: %s\n", hostname);


   return 0;
}
```

Save this code and try to compile and run.

The following is a sample output.



10.2 Creating and Connecting

In this section we explore socket programming. For illustration we build client-server app. In this scenario, we just connect and then close.

To create communication, we can use socket() function. It can be defined as below.

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

domain is defined as below.

```
Name                Purpose
------------------------------------------------
AF_UNIX, AF_LOCAL   Local communication
AF_INET             IPv4 Internet protocols
AF_INET6            IPv6 Internet protocols
AF_IPX              IPX - Novell protocols
AF_NETLINK          Kernel user interface device
AF_X25              ITU-T X.25 / ISO-8208 protocol
AF_AX25             Amateur radio AX.25 protocol
AF_ATMPVC           Access to raw ATM PVCs
AF_APPLETALK        Appletalk
AF_PACKET           Low level packet interface
```

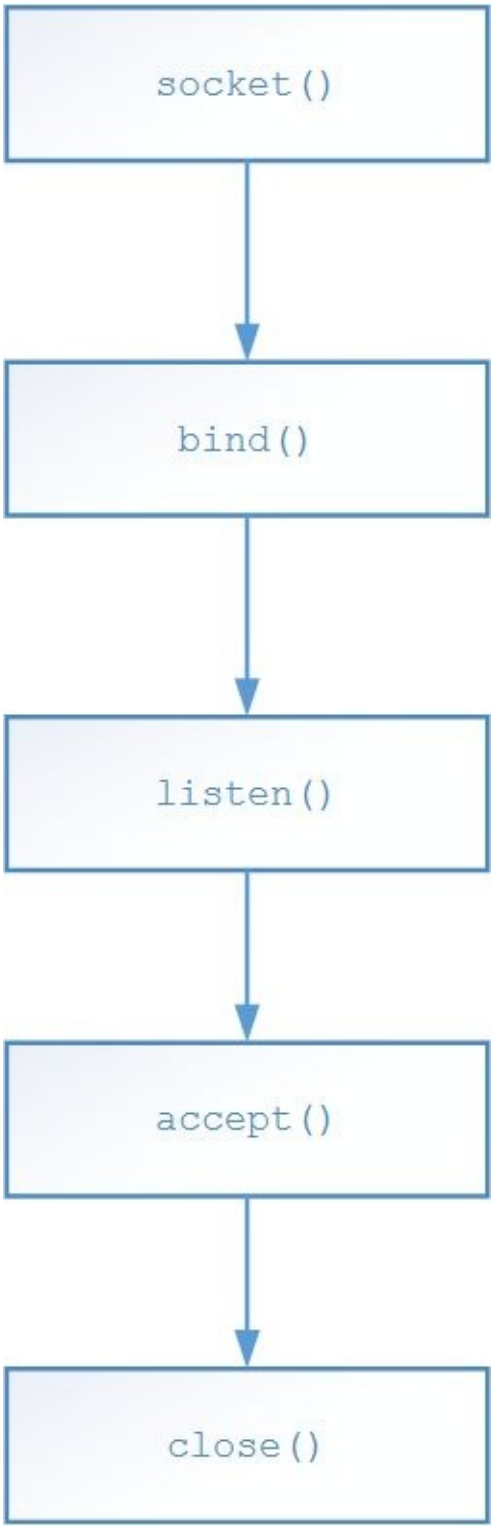
type is defined as below.

```
SOCK_STREAM     Provides sequenced, reliable, two-way, connection-
                based byte streams.  An out-of-band data transmission
                mechanism may be supported.

SOCK_DGRAM      Supports datagrams (connectionless, unreliable
                messages of a fixed maximum length).

SOCK_SEQPACKET  Provides a sequenced, reliable, two-way connection-
                based data transmission path for datagrams of fixed
                maximum length; a consumer is required to read an
                entire packet with each input system call.

SOCK_RAW        Provides raw network protocol access.

SOCK_RDM        Provides a reliable datagram layer that does not
                guarantee ordering.

SOCK_PACKET     Obsolete and should not be used in new programs;
```

We will focus on socket type: SOCK_STREAM and SOCK_DGRAM.

10.2.1 Server

To build server, you can follow steps, shown in Figure below.



bind(), listen(), accept() and close() function can be defines as follows.

```
#include <sys/types.h>
#include <sys/socket.h>
```

```c
#include <unistd.h>

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
int listen(int sockfd, int backlog);
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
int close(int fd);
```

We usually struct sockaddr_in to pass parameter addr. It is defined as follows.

```c
struct sockaddr_in {
   sa_family_t    sin_family; /* address family: AF_INET */
   in_port_t      sin_port;   /* port in network byte order */
   struct in_addr sin_addr;   /* internet address */
};

struct in_addr {
   uint32_t       s_addr;     /* address in network byte order */
};
```

For implementation, we build server app by creating a file, called **simple_server.c**, and write this code.

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <errno.h>


int main(void)
{

    int sockfd, newsockfd, portno, clilen;
    struct sockaddr_in serv_addr, cli_addr;

    portno = 8056; // server port
    errno = 0;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
    {
        printf("\n socket() failed with error [%s]\n",strerror(errno)
        return -1;
    }

    bzero((char *) &serv_addr, sizeof(serv_addr));

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);
```

```
30
31     if (bind(sockfd, (struct sockaddr *) &serv_addr,
32           sizeof(serv_addr)) < 0)
33     {
34         printf("\n bind() failed with error [%s]\n",strerror(errno));
35         return -1;
36     }
37
38     printf("socket has created and binded\n");
39     printf("listening incoming socket client….\n");
40
41     listen(sockfd,5);
42     clilen = sizeof(cli_addr);
43     newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen
44     if (newsockfd < 0)
45     {
46         printf("\n accept() failed with error [%s]\n",strerror(errno)
47         return -1;
48     }
49
50     printf("a client was connected\n");
51
52     // close socket
53     close(newsockfd);
54     close(sockfd);
55
56    return 0;
57 }
```

You can compile this program.

```
$ gcc -o simple_server simple_server.c
```

10.2.2 Client

To connect the server, we can use connect() function. This function can be defined as below.

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int sockfd, const struct sockaddr *addr,
            socklen_t addrlen);
```

Server information is passed to addr parameter.

For client implementation, we create a file, called **simple_client.c**, and write this code.

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <errno.h>


int main(void)
{
    int sockfd, portno;

    struct sockaddr_in serv_addr;
    struct hostent *server;

    // server hostname. You can change it
    char server_hostname[] = "ubuntu15";
    portno = 8056; // server port
    errno = 0;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
    {
        printf("\n socket() failed with error [%s]\n",strerror(errno)
        return -1;
    }

    printf("socket has created\n");

    server = gethostbyname(server_hostname);
    if (server == NULL)
    {
        printf("\n gethostbyname() failed with error [%s]\n",strerror
        return -1;
    }

    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,
         (char *)&serv_addr.sin_addr.s_addr,
         server->h_length);
    serv_addr.sin_port = htons(portno);

    printf("connecting to server….\n");
    if (connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr
    {
        printf("\n connect() failed with error [%s]\n",strerror(errno
        return -1;
    }

    printf("connected to server\n");

```

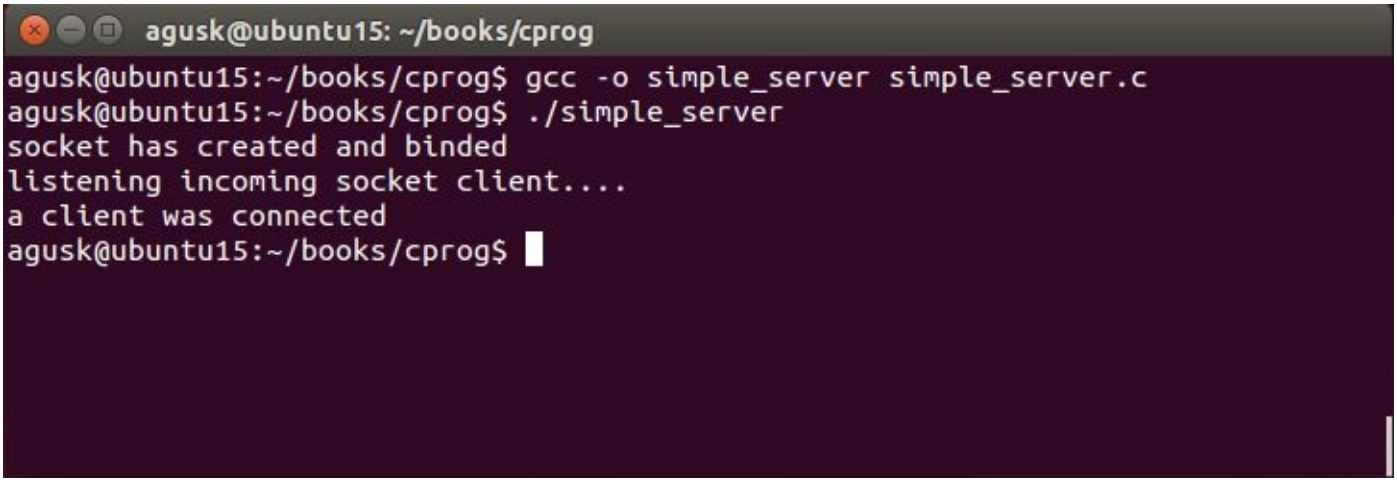
```
54      // close socket
55      close(sockfd);
56
57      return 0;
58  }
```

Save all code. Try to compile.

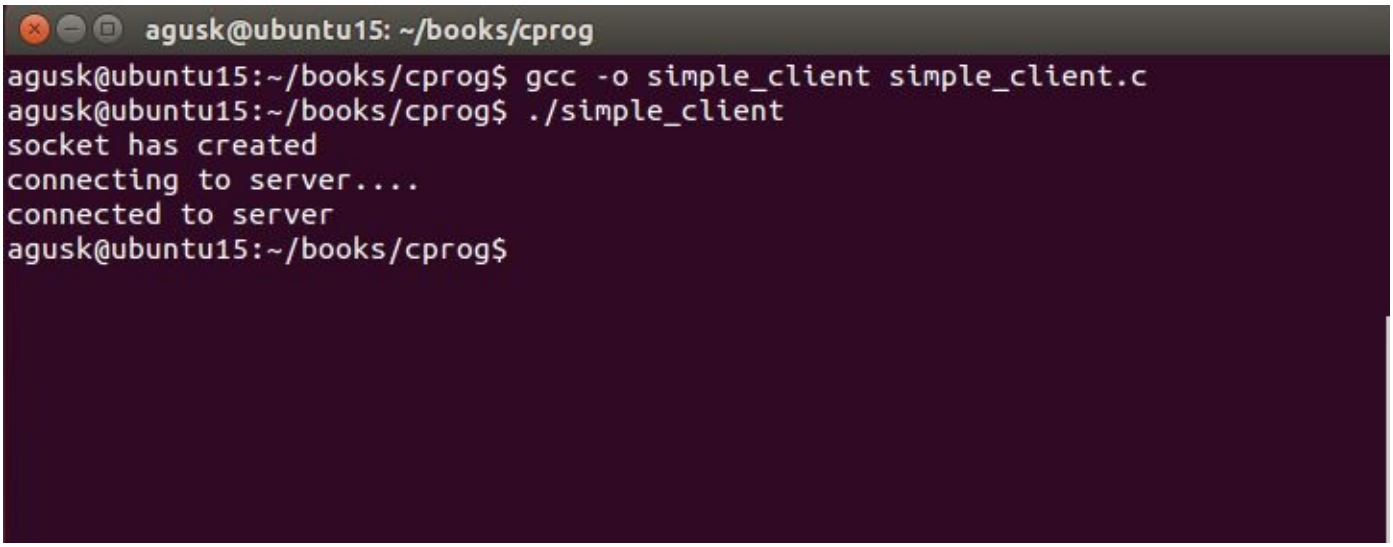
```
$ gcc -o simple_client simple_client.c
```

10.2.3 Testing

Now we can test our server and client app. Firstly we run server app. The following is a sample output.



Then you run client app.



After client connected to server, server and client close connection.

10.3 Data Transfer

We can send and receive data via socket using read() and write functions. These functions are defined as below.

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
```

These functions are general I/O functions. We can use specific I/O for socket operations using send() and recv() functions. They can implemented as follows.

```
#include <sys/types.h>
#include <sys/socket.h>

ssize_t send(int sockfd, const void *buf, size_t len, int flags);
ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

In this section we build client-server like section 10.2 but we try to transfer data using write() and read() functions.

10.3.1 Server

After client connected, we read data from client using read() function. Then we send message to client using write() function. After that, we close connection.

For implementation, we create a file, called **data_server.c**, and write this code.

```
 1 #include <stdio.h>
 2 #include <sys/types.h>
 3 #include <sys/socket.h>
 4 #include <netinet/in.h>
 5 #include <string.h>
 6 #include <errno.h>
 7
 8
 9 int main(void)
10 {
11
12     int sockfd, newsockfd, portno, clilen, sz;
13     char buffer[256];
14     struct sockaddr_in serv_addr, cli_addr;
15
16     portno = 8059; // server port
17     errno = 0;
```

```
18
19      sockfd = socket(AF_INET, SOCK_STREAM, 0);
20      if (sockfd < 0)
21      {
22          printf("\n socket() failed with error [%s]\n",strerror(errno)
23          return -1;
24      }
25
26      bzero((char *) &serv_addr, sizeof(serv_addr));
27
28      serv_addr.sin_family = AF_INET;
29      serv_addr.sin_addr.s_addr = INADDR_ANY;
30      serv_addr.sin_port = htons(portno);
31
32      if (bind(sockfd, (struct sockaddr *) &serv_addr,
33            sizeof(serv_addr)) < 0)
34      {
35          printf("\n bind() failed with error [%s]\n",strerror(errno));
36          return -1;
37      }
38
39      printf("socket has created and binded\n");
40      printf("listening incoming socket client….\n");
41
42      listen(sockfd,5);
43      clilen = sizeof(cli_addr);
44      newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen
45      if (newsockfd < 0)
46      {
47          printf("\n accept() failed with error [%s]\n",strerror(errno)
48          return -1;
49      }
50
51      printf("a client was connected\n");
52
53      // read data
54      sz = read(newsockfd,buffer,255);
55      if (sz < 0)
56      {
57          printf("\n read() failed with error [%s]\n",strerror(errno));
58          return -1;
59      }
60      printf("Received message: %s\n",buffer);
61
62      // write data
63      sz = write(newsockfd,"this is message from server",30);
64      if (sz < 0)
65      {
66          printf("\n write() failed with error [%s]\n",strerror(errno))
67          return -1;
68      }
69
70      // close socket
```

```
71     close(newsockfd);
72     close(sockfd);
73 
74    return 0;
75 }
```

This code is similar to code on section 10.2.1 but we continue to read and send data.

Save this code and compile it.

```
$ gcc -o data_server data_server.c
```

10.3.2 Client

On client side, we send message after connected. Then we read message from server.

Let's start to create a file, called **data_client.c**, and write this code.

```
 1 #include <stdio.h>
 2 #include <sys/types.h>
 3 #include <sys/socket.h>
 4 #include <netinet/in.h>
 5 #include <netdb.h>
 6 #include <string.h>
 7 #include <errno.h>
 8 
 9 
10 int main(void)
11 {
12     int sockfd, portno, sz;
13 
14     struct sockaddr_in serv_addr;
15     struct hostent *server;
16     char buffer[256];
17 
18     // server hostname. You can change it
19     char server_hostname[] = "ubuntu15";
20     portno = 8059; // server port
21     errno = 0;
22 
23     sockfd = socket(AF_INET, SOCK_STREAM, 0);
24     if (sockfd < 0)
25     {
26         printf("\n socket() failed with error [%s]\n",strerror(errno)
27         return -1;
28     }
29 
30     printf("socket has created\n");
31 
```

```
32      server = gethostbyname(server_hostname);
33      if (server == NULL)
34      {
35          printf("\n gethostbyname() failed with error [%s]\n",strerror
36          return -1;
37      }
38
39      bzero((char *) &serv_addr, sizeof(serv_addr));
40      serv_addr.sin_family = AF_INET;
41      bcopy((char *)server->h_addr,
42           (char *)&serv_addr.sin_addr.s_addr,
43           server->h_length);
44      serv_addr.sin_port = htons(portno);
45
46      printf("connecting to server….\n");
47      if (connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr
48      {
49          printf("\n connect() failed with error [%s]\n",strerror(errno
50          return -1;
51      }
52
53      printf("connected to server\n");
54
55      // write data
56      strcpy(buffer,"this is message from client");
57      sz = write(sockfd,buffer,strlen(buffer));
58      if (sz < 0)
59      {
60          printf("\n write() failed with error [%s]\n",strerror(errno))
61          return -1;
62      }
63
64      // read data
65      bzero(buffer,256);
66      sz = read(sockfd,buffer,255);
67      if (sz < 0)
68      {
69          printf("\n read() failed with error [%s]\n",strerror(errno));
70          return -1;
71      }
72      printf("Received message: %s\n",buffer);
73
74      // close socket
75      close(sockfd);
76
77      return 0;
78  }
```

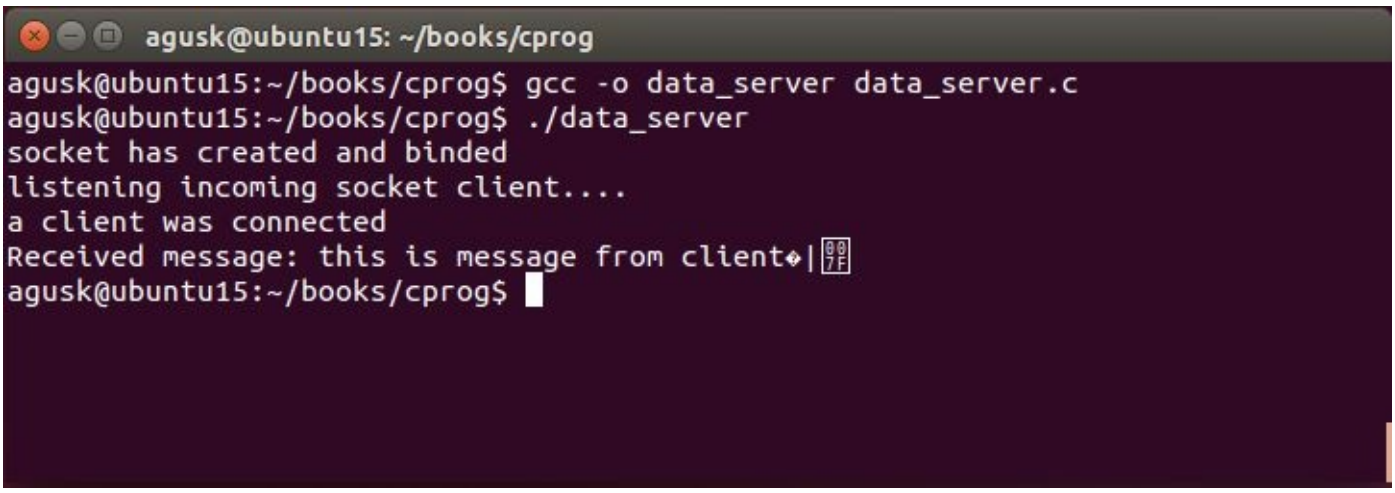
Save this code and compile it.

```
$ gcc -o data_client data_client.c
```

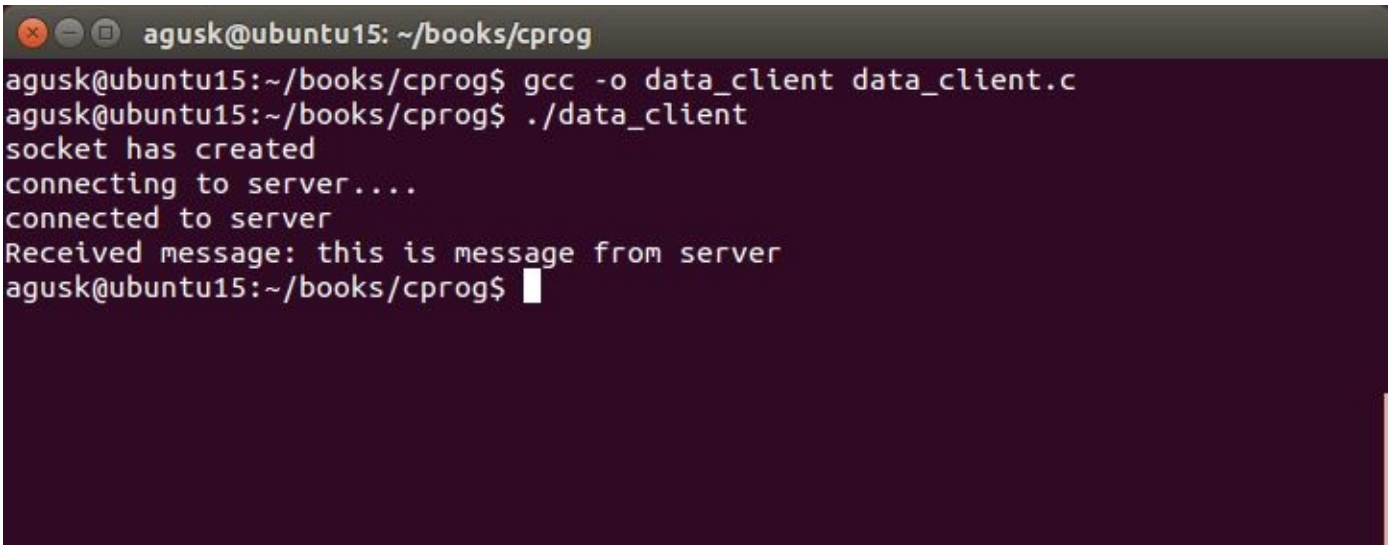
10.3.3 Testing

We are ready to test our client-server app. Firstly we run server app.

The following is a sample output for server app.



Then we run client app. You can see the output, shown in Figure below.



Contact and Source Code

If you have question related to this book, please contact me at aguskur@hotmail.com . My blog: http://blog.aguskurniawan.net

You can download source code for this book
on http://www.aguskurniawan.net/book/cprog03282015.zip

Contact

If you have question related to this book, please contact me at aguskur@hotmail.com . My blog: http://blog.aguskurniawan.net.