CS232L Operating Systems Lab
Lab 03: Introduction to C Programming

CS Program
Habib University

Fall 2021

1 Introduction

This document assumes that you are already familiar with C++, and thereby with the basics of C, so we'll be going kind of quickish through the introduction!

In this lab you will learn how to:

1. Understand the workflow of compiling and running a C program
2. Invoke gcc and compile a program from command line
3. Do simple I/O in programs
4. Manipulate strings in programs
5. Do File I/O in programs

2 C Proramming Workflow

The basic flow of writing and executing a C program is [2] shown in Figure 1:

3 Resources

The course book comes with a tutorial for lab which introduces the basics of using **gcc** and **makefiles**. The tutorial is available at the link:

http://pages.cs.wisc.edu/~remzi/OSTEP/lab-tutorial.pdf

Start with learing how to run gcc. We'll cover makefiles next week. You can also leave the section **F.7** for later.

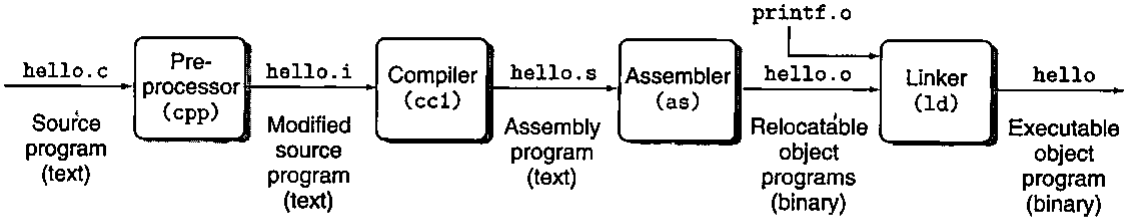

Figure 1.3 The compilation system.

Figure 1: C compilation process

4 I/O

C standard library provides with you functions to communicate with the standard input and standard output. The functions `scanf()` and `printf()` are used to take formatted input from the user and to display formatted output to the user, respectively. These functions work with formatted strings which possible contain type specifiers indicating how many and which type of variables are to be input or output. The program in listing 1 depicts a program which takes a few inputs from the user and then displays some output. Pay attention to how it uses fromatted strings and the variables for input and output.

```c
#include <stdio.h>

void print_table (int num){
  int i=0;
  for(i=1; i<=10; i++)
    printf("%d x %d = %d\n", num, i, num*i);
}

int main(){
  int i=0;
  int num=0;
  float gpa = 0.0;
  char name[20];

  printf("Enter your name:\n");
    scanf ("%s", name);

  printf("Enter your gpa:\n");
    scanf ("%f", &gpa);

  printf("Enter your favourite number:\n");
    scanf ("%d", &num);


  printf("Welcome %s (gpa=%f)! ", name, gpa);
  printf("Here's yoru table:\n");
  print_table(num);

  printf("The End.\n");
  return 0;
}
```

Listing 1: example_io.c

5 File I/O

The basic flow to do file i/o is quite simple. You:

- open a file
- manipulate the file
- close the file

Don't forget the last step as not closing a file would result in resource leaks.

In Linux opened file are represented as streams represented by `FILE *` objects.
The `fopen()` function would take a path to a file and a mode and return a pointer to a stream object which we can then pass to other functions. The mode argument tells the `fopen` whether we want open the file in read mode, write mode, or both.

```c
/*hello_fopen.c*/
#include <stdio.h>
#include <stdlib.h>

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

  FILE * stream = fopen("helloworld.txt", "w");

  fprintf(stream, "Hello World!\n");

  fclose(stream);
}
```

Listing 2: hello_fopen.c

In the above listing, the `fprintf()` function uses the stream pointer returned by the `fopen()` to write to the opened file. `fclose()` then close the file stream that had been opened by the `fopen()`.
When a process is created, the OS will open three stream by default: stdin, stdout, and stderr.

`fprintf()` and `fscanf()` can be used to do formatted output and input on opened file streams. The function `printf()` is a wrapper for `fprintf` with the first argument pointing to stdout. Similarly `fscanf()` for `scanf()` with the input stream set to stdin.

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

int main(int argc, char * argv){

  FILE * stream = fopen("students.dat", "r");

  char iD[1024],lname[1024];
  int a;
  float b;
  char c;

  while ( fscanf(stream,
                 "%s %s %d %f %c",
                 iD, lname, &a, &b, &c) != EOF){

    printf("ID: %s\n",iD);
    printf( "Name: %s\n",lname);
    printf("marks: %d\n",a);
    printf("gpa: %f\n",b);
    printf("grade: %c\n",c);
    printf("\n");
  }

  fclose(stream);
  return 0;
}
```

Listing 3: file_io.c

The EOF character is a special character that signifies the end of file. `fscanf()` would return EOF when there's no more data in the file.

5.1 Todo: file io

1. read about the different values that could be passed to the `mode` parameter of `fopen()` and experiment with them.

2. modify the above listing so that every time it runs, it should append to the file `helloworld.txt`.

3. write a program that imitates the linux `head` command i.e., display the first 10 lines of the file passed on its command line argument.

4. write a program that imitates the linux `cp` command i.e., given two filenames as arguments, copies the contents of first file to the second by overwriting the destination file. It should create the destination file if it doesn't exist.

6 String Manipulation

A string in C is a series of characters terminated by the null character ('\0'). The C standard library provides a few basic functions to manipulate strings. These functions usually starts with the prefix `str`. The program in listing 4 shows a few examples:

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

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

  const char * s1 = "hello world!\n";
  const char * s2 = "bye, bye!\n";
  char s3[20];

  // get string length
  int len1 = strlen(s1);
  printf ("The length of stirng \"hello world!\\n\" is %d\n", len1);


  // copy a string
  strcpy(s3, s1);

  // compare two strings
  if (strcmp(s1, s3) == 0)
    printf("Now s1=%s, and s3=%s\n", s1, s3);
  else
    printf("Ooops!! something went wrong copying s1 to s3!\n");


  // tokenize a string
  const char * s4 = "this is a string";
  char s5[20];
  strcpy(s5, s4);

  int i = 1;
  char * token = strtok(s5, " ");

  while (token != NULL){
    printf ("value of token is = %s\n", token);
    token = strtok(NULL, " ");
  }

  return 0;
}
```

Listing 4: cstrings.c

Read their descriptions and try playing with them.

6.1 Todo: string manipulation

1. read the descriptions of `strlen()`, `strcpy()`, and `strcmp()` and implement them as your own functions `my_strlen()`, `my_strcpy()`, and `my_strcmp()`

2. you are given a file which contains information about processes. Each line of the file contains information "process_id:name:duration:priority" where process_id, name, duration, and priority are integer, string, float and integer respectively. all these fields are separated

via colons. Write a program that reads this file and displays the process names and priority to the user.

References

[1] Raymond Eric S. *"Basics of Unix Philosophy" The Art of Unix Programming.* Addison-Wesley, Professional.

[2] BTYANT, O'HALLARON. *Computer Systems, A Programmer's Perspective.* Pearson.