Analysis & Design of Algorithms

Semster III 2018-19

Lab - 1

Topics: C Programming - Command Line Arguments, File I/O, Array, Pointers.

---

1 Command Line Arguments

1.1 Using Linux

A simple program in C is a text file with a “.c" suffix as in “cprogram.c". This can be compiled (using gcc) to get an executable file:

gcc -o cprogram cprogram.c

The above command generates an executable file named test that can be invoked as follows:

./cprogram

Question: Why do we need to use “./cprogram" instead of just "cprogram"?

Question: Which part of the program gets executed when invoked as above?

We may write a program that requires one or more of its inputs to be fed at the command line at the time of invocation. This may also be required when we want the program to be configured/customized based on inputs given at the time of invocation. This can be achieved by using command line arguments in C. When the “main” procedure in a C program gets invoked it can be passed some arguments - referred to as “command line arguments”. For this purpose C allows the main procedure to be defined with two parameters (whose types are fixed):

```c
#include <stdio.h>
int main(int argc, char *argv[])
{
// body of the procedure goes here.
}
```

Note that the first parameter (argc) is an integer value that keeps a count of the number of command line arguments and the second parameter (argv) is an array of strings (i.e. values of type char *) that holds all the arguments passed via the command line to this program (that caused the invocation of this main procedure). In particular, argv[0] holds the name with which the program was invoked and for i > 0, each argv[i] holds a string corresponding to the ith word (i.e. a string separated by space) passed as the command line argument.

For instance, given the following main program:

```
/* File: test.c */
#include <stdio.h>
int main(int argc, char *argv[])
{
  printf("\n%s", argv[0]);
}
```

executing these commands gcc -o cprogram cprogram.c
./cprogram will output

1.2 passing arguments using IDE codeblocks (Windows)

1. With code::blocks you can set your command line arguments like this in the menu:
2. Project > Set programs' arguments...
3. This opens a window where you can insert your parameters.

Exercise

1.a If you invoke the program test as follows, what will be the value of argc? ./cprogram hello world

1.b Modify the program test so that it prints "First C Program hello world" for the above invocation. Generalize the program so that it prints the name of the program and all the command line arguments in sequence.

1.c Modify the program so that if a command line argument is an integer the parameter string gets converted to an integer. Refer to help pages for conversion function atoi.

2 FILES AND INPUT/OUTPUT

A file is an abstraction of the way data gets stored in external or secondary memory (i.e. hard disk, flash memory device, optical disk etc.). A file can be treated as a sequential access FIFO list by a program i.e. if the contents of a file were (say, for instance): Q W E R T Y the first accessible element is Q, then W, then E, and so on i.e. to access E one has to access Q, then W, and then access E; and when say a value U is added to the above file, the contents would be

Q W E R T Y U
i.e. if and when a new element is added it is added to the end (of the list i.e. file).

Linux supports text and binary files. We will deal with text files for now. Text files can be accessed character by character or word by word (i.e strings separated by space). C provides an I/O library stdio that contains procedures for I/O access. The header file stdio.h contains the headers (i.e. dummy definitions) of these procedures.

Exercise

2.a: Locate the file stdio.h in your computer, read and understand at least one pair of I/O procedures. Note that the command find may be used to locate files by name. Refer to the help pages for information on how find works.

2.b: C libraries support two procedures fscanf and fprintf for reading and writing to a file. Refer to the man pages for information on how to use these procedures. They are similar to scanf and printf but take an additional (first) argument that is a file pointer.

2.c: C libraries provide procedures fopen and fclose for initialization and finalization of a file. Refer to the help pages for information on how to use these procedures.

Question: How do you obtain a file pointer?
Since files are abstractions of physical persistent storage, typically initialization and finalization are required i.e. initialization must be done before any read/write operations, and finalization must be done after all read/write operations (particularly before close of program execution).

The typical structure of a program fragment that reads from and/or writes to a file is as follows:

```
FILE *f = fopen("testfile.txt"); // fopen returns a fle pointer

/* read or write operations using the file pointer f */

fclose(f);
```

We will now see two sample program to illustrate the file pointers.

```
// Sample 1: Write to a text file using fprintf()
/* Sample1.c */
#include <stdio.h> int main() {
int num;
FILE *fptr;
fptr = fopen("program.txt","w");
if(fptr == NULL)
  {
    printf("Error!");
    exit(1);
  }
printf("Enter num: ");
scanf("%d",&num);
fprintf(fptr,"%d",num);
fclose(fptr);
return 0; }
```

You can compile this as we did for the previous exercise and run the executable. After you compile and run this program, you can see a text file program.txt created in the same directory. When you open the file, you can see the integer you entered.

```
// Example 2: Read from a text file using fscanf()
/* Sample2.c */
#include <stdio.h> int main() {
int num;
FILE *fptr;
Fptr = fopen ("program.txt", "r");
if (fptr == NULL){
  printf("Error! opening file");
  // Program exits if the file pointer returns NULL.
  exit(1);
  }
fscanf(fptr,"%d",&num);
printf("Value of n=%d",num);
fclose(fptr);
return 0;
}
```

This program reads the integer present in the program.txt file and prints it onto the screen. If you successfully created the file from Sample 1, running this program will get you the integer you entered.

Other functions like fgetchar(), fputc() etc. can be used in similar way. You can also call fscanf and fprintf functions in a loop to read multiple lines from a file. We will see how to do that in the linked list example.

Exercise

2.d: Write a C program that copies the contents of a file into a different file (given two filenames as command line arguments). Error-proof your argument: i.e. detect and print messages when errors occur - for instance, file is not present, unable to read/write, etc. Also ensure that your program terminates gracefully when errors occur.

2.e: Modify your program (for 2d) so that if only one file name is passed as command line argument then the contents are copied to display. Note that stdout is a file pointer that typically refers to the output file abstraction corresponding to the display device.

The table below illustrates the modes to be used.

Table 1: File System Mode

File Mode	Meaning of Mode	During Inexistence of file
R	Open for reading.	If the file does not exist, fopen() returns NULL.
Rb	Open for reading in binary mode.	If the file does not exist, fopen() returns NULL.
W	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
A	Open for append. i.e, Data is added to end of file.	If the file does not exists, it will be created.
Ab	Open for append in binary mode. i.e, Data is added to end of file.	If the file does not exists, it will be created.
r+	Open for both reading and writing.	If the file does not exist, fopen() returns NULL.
rb+	Open for both reading and writing in binary mode.	If the file does not exist, fopen() returns NULL.
w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb+	Open for both reading and writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exists, it will be created.
ab+	Open for both reading and appending in binary mode.	If the file does not exists, it will be created.

3 Pointers in C

Let us revise the basics on pointers in this exercise. A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is

type *var-name;

Here, type is the pointer's base type; it must be a valid C data type and var-name is the name of the pointer variable. The asterisk '* 'used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Take a look at some of the valid pointer declarations

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to. How to Use Pointers? There are a few important operations, which we will do with the help of pointers very frequently.

a. We define a pointer variable.

b. assign the address of a variable to a pointer

c. finally access the value at the address available in the pointer variable. This is done by using unary operator * that returns the value of the variable located at the address specified by its operand.

The following example makes use of these operations

```
/* Pointers_Sample.c */
  #include <stdio.h> int main () {
  int   var = 20;   /* actual variable declaration */
  int   *ip;        /* pointer variable declaration */
  ip = &var;  /* store address of var in pointer variable*/
  printf("Address of var variable: %x\n", &var  );

  /* address stored in pointer variable */
  printf("Address stored in ip variable: %x\n", ip );
```

```c

  /* access the value using the pointer */
  printf("Value of *ip variable: %d\n", *ip );
  return 0;
  }
```

When the above code is compiled and executed, it produces the following result
Address of var variable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20

NULL Pointers: It is always a good practice to assign a NULL value to a pointer variable in case you do not have an exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a null pointer. The NULL pointer is a constant with a value of zero defined in several standard libraries. Consider the following program

```c
/* NULL_Pointer_Sample.c */
 #include <stdio.h>
int main () {
    int  *ptr = NULL;
  printf("The value of ptr is : %x\n", ptr  );
  return 0;
}
```

When the above code is compiled and executed, it produces the following result
The value of ptr is 0
To check for a null pointer, you can use an 'if' statement as follows:

```c
if(ptr)     /* succeeds if p is not null */  \\
if(!ptr)    /* succeeds if p is null */  \\
```

We shall be seeing more of pointers in the next exercise.

Exercise

3.a: Write a C program that reads the contents of a file (containing a sequence of characters or a string) into a array. Perform insertion and deletion operation. Find out whether the character string stored in array is palindrome or not. To accomplish the task, copy the content of array into another array and check for palindrome by comparing content of each element in both the array. Pass the filenames as command line arguments.

3.a: Write a C program that reads the content of two different files in two separate array. Both the files contain sequence of integer numbers separated by space in ascending order. Write a function which will traverse these two array and merge them without disturbing the order. Merged list should be then copied to an output file. Pass file names as command line arguments.