Project 0: Getting to Know C (Better)

1 Background

The language C, as a system programming language, is used extensively in working with UNIX based operating systems, including Linux, Solaris, etc.. We will do most of the projects for this course in C.

Most, if not all, of you should have learned it in CS2470 System Programming in C/C++. In this lab, we will review the process of compiling and executing C programs in the UNIX environment, as well as some of the basic features of C through examples\(^1\).

When working in a programming lab on campus, you can get access to C by logging into turing via Secure Shell (SSH), under All Programs/Network. Since Unix is a multiuser system, we need a user account/password to get into the system. It is the same as what you have been using to get into my.Plyouth.edu.

We can also remotely log into the turing system with, e.g., SSH, which can be obtained via https://shareware.unc.edu/.

2 How to compile and execute a C program?

To run a program written in c, e.g., hello.c, just log into turing, and type in, assuming that \(\%\) is used as the prompt,

\[
\%cc \text{hello.c}
\]

Once the program is successfully compiled, to run the program, simply type, for any program,

\[
\%./\text{a.out}
\]

If you don’t like a.out, the default name for the executable, you can use the option -o as follows:

\[
\% cc -o \text{hello hello.c}
\]

Now you can type the following to run the executable.

\[
\%./\text{hello}
\]

\(^1\)To complete the assignments made within this project, I would suggest you to consult a book on C, e.g. [1, 2], do a Google, or you can always come to chat with me.
3 Examples

Let’s go through a few examples. For each of them, use any editor, such as Notepad++, to type in the code; save it as a text file with .c as its extension; upload the program to an appropriate folder in turing; run the executable as we did; write down the result, as well as an explanation for the questions/issues raised for each of these examples.

You can also log into turing, then use an editor, such as pico or vi to enter your program. The former is really easy. For an explanation/manual of the vi editor, please check out the tutorial that can be found in the project page.

1. The Hello program: Traditionally, this is the very first C program that everybody starts with.

```c
#include <stdio.h>

main(){
    printf("hello, world\n");
}
```

2. The temperature conversion program: This winter has been unusually warm. I just saw that the temperature on the Christmas Eve is going to be 64°F. To explain it to your Canadian friend, you can use the following program to convert it to 17.8°C.

```c
#include <stdio.h>

/* print Fahrenheit-Celsius table
   for fahr=0, 20, ..., 300 */

main(){
    int fahr, celsius;
    int lower, upper, step;

    lower=0;
    upper=300;
    step=20;

    fahr=lower;
    while(fahr<=upper){
        celsius=5*(fahr-32)/9;
        printf("%d\t%d\n", fahr, celsius);
        fahr=fahr+step;
    }
}
```
Play the program first to have some basic understanding. Notice that the loop structure in C is essentially the same as that in Java, but the input/output is quite different. Thus, you get to dig out some of the details about the `printf` line, particularly, the types of those objects to be printed. You might want to consult a C book, e.g., [1, 2] and write down all the stuff we can use after the `%` sign, together with the associated types of the objects to be printed, and a bit explanation.

**Assignment:** Find out which function, instead of sending things out, brings things in; then modify the program by turning it into an interactive one. Thus, instead of getting the Fahrenheit degrees within the program, you can actually ask the users to type it in. Please also include some prompting message.

3. **The input/output program:** The following program gets a line of digits from the keyboard, and sends back something. For this one, read through every line and write down your expectation. Then experiment with it, even modify it, to see what happens.

**Assignment:** What happens if you add `return i;` into the function `main()` as the last line? What happens if the comment “\"” is removed in the `main` function, etc.?

You also have to find out what `extern`, and other `storage class specifiers`, mean and do.

```
/home/zshen > more longest.c
#include <stdio.h>

#define MAXLINE 1000 /* maximum input line size */

int max; /* maximum length seen so far */
char line[MAXLINE];
char longest[MAXLINE];

int getline1();
void copy(void);

/* print longest input line */
main()
{
    int len;
    extern int max;
    extern char longest[];

    max=0;
    while((len=getline1())>0)
        if(len>max){
            // printf("%d\n", len);
```
```c
max=len; copy();
}
if(max>0)
    printf("%s", longest);
}

/*getline */
int getline1(){
    int c, i;
    extern char line[];

    for(i=0; i<MAXLINE-1 && (c=getchar())!=EOF && c!='\n'; ++i)
        line[i]=c;
    if(c=='\n'){
        line[i]=c; ++i;
    }
    line[i]='}0';
    return i-1;
}

/*copy */
void copy(){
    int i;
    extern char line[], longest[];

    i=0;
    while((longest[i]=line[i])!='\0')
        ++i;
}

4. **How about those arguments?** We can attach a list of string type arguments with the call made by the `main` function, and, make use of it within the program. For example, if we give two argument, `arg1`, `arg2`, we can execute the program as follows:

```c
a.out arg1 arg2
```

In the following example, we simply echo these arguments back.

```c
#include<stdio.h>

main(int argc, char *argv[]){
    int i;
    ```
5. **How about those pointers and arrays?** The following code does exactly the same thing as the previous one does. But, it follows a different approach by using *pointers*. The point type, similar to the *reference* type of Java, gives out the address of the objects, rather than the name. Such an explicit type has been widely used in operating systems.

**Assignment:** Compare this program with the previous one, and dig out all the operational details. For example, write down the data structure used by the `argv` array, together with the values of `argc` and the position of `argv` for each and every run of the loop.

```c
#include<stdio.h>

main(int argc, char *argv[]){
    while(--argc>0)
        printf("%s%s", *++argv, (argc>1) ? " ": "");
    printf("\n");
    return 0;
}
```

When you really understand the pointer related stuff, write a program that will exchange the contents of two `int` type variables, using pointers. For example, assume the program is named `switch`, then the follow segment should print out 4 and 3.

```c
int a=3, b=4;
switch(a, b);
printf("%d%d", a, b);
```

6. **How to work with files?** The following example demonstrates how to work with files.

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

void double_space(FILE *, FILE *);
```
void prn_info(char *);

main(int argc, char **argv){
    FILE *ifp, *ofp;

    if(argc!=3){
        prn_info(argv[0]);
        exit(1);
    }
    ifp = fopen(argv[1], "r"); /* open for reading */
    ofp = fopen(argv[2], "w"); /* open for writing */
    double_space(ifp, ofp);
    fclose(ifp);
    fclose(ofp);
}

void double_space(FILE *ifp, FILE *ofp){
    int c;

    while((c=getc(ifp))!=EOF){
        putc(c, ofp);
        if(c=='\n')
            putc('\n', ofp); /* Add an extra new line sign */
    }
}

void prn_info(char *pgm_name){
    printf("\n%s%s%s\n\n%s%s\n
", "Usage: ", pgm_name, " infile outfile",
    "The contents of infile will be double-spaced ",
    "and written to outfile.");
}

It actually reads in the first file, puts it into the second file, while adding in an extra space line in between every two lines in the first file, when called as follows:

a.out first second

For example,

/home/zshen > more testIn.txt
This is to test the
file copy program.
Let’s see what happens.
/home/zshen > ./a.out testIn.txt testOut.txt
/home/zshen > more testOut.txt
This is to test the

file copy program.

Let’s see what happens.

/home/zshen >

**Assignment:** Play with this program first, and then write a program of your own that will reverse whatever in the first file, and put it back into the second. For example, if the input file contains “abc”, the second should contain “cba”, at the end of the execution.

## 4 What to send in?

You should send in two kinds of files:

1. A bunch of .c file, either numerically labeled or named after the question, e.g., `tempInt.c` for the interactive version of the temperature program, together with a sample session with the input and output.

2. You should also send in a .txt file that collects clearly labeled answers to the assignments made for all the problems.

Below is a general grading guideline:

1 (+/−): A serious effort has been made to solve some of the problems and appropriate files sent in.

2 (+/−): All the six problems are correctly solved, and appropriate files are sent in....

3: ... and all the assignments are also corrected addressed.

**References**
