C: A Tutorial Introduction

- The C programming language was designed by Dennis Ritchie and has been widespread use since the 1970s!
- Initially the standard was defined by the *The C Programming Language* book by Kernighan and Ritchie
- Later standardizations:
  - ANSI C or C'89 or ISO C'90 all refer to the same language. This is the most widely used and supported version of the language
  - C'99 was the next standardization that added several new feaures However, this is still not fully supported by all compilers... :-(
  - C'11 is the latest standardization in 2011
- Many languages have directly or indirectly borrowed from C. Examples are C#, Java, Javascript, Objective C, Perl, Python, and several others

Structure of C Programs

- Header files are usually used for declarations (files named with extension `.h`) and source files usually contain function and variable declarations (files named with extension `.c`)
- A function in C is similar to a *method* in Java. Functions have arguments and a *signature* (in C, we call them a prototype) like in Java
- In general, a C program consists of multiple header and source files. A source file will often refer to header files via the `#include` directive. For example:

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

- Comments.
  - Block comments `/* ... */` (same as in Java)
  - Line comments (C99, C++) `//` (same as in Java)

Structure of C Programs

- The `main` function does not have a fixed prototype (signature in Java). Here is the canonical C program with the recommended prototype

```
/* C-examples/intro/hello.c */
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Hello World!\n");
    return 0;
}
```

Basic types and statements

- Variable data types. Basic data types are similar to Java. E.g. `char`, `short`, `int`, `long`, `float`, `double` Note that the sizes of types are machine dependent unlike in Java!
- Defining constants. Simplest way is shown below. Other ways will be discussed later

```
#define E 2.71828182845905
```

- Operators and expressions. These are the same as in Java with some minor differences
- Control-Flow statements. The basic statements `if/else`, `while`, `do-while`, `for`, `switch` are the same as in Java. In addition, the `break/continue` statement exit from the innermost enclosing loop like in Java but cannot use a label to break to as in Java
- C also has a `goto` statement that Java does not have

C Standard Library

- The C standard library is a collection of useful functions that we can use by including appropriate header files. Some of the common header files are `<stdio.h>`, `<stdlib.h>`, `<string.h>`.
- Some commonly used functions are `printf`, `getchar`, `putchar`, string functions and memory allocation functions
- You can read the man page for any of the functions in the standard library. The standard library functions are defined in the section 3 of the man pages. For example, try the following command in the terminal:
  `man 3 printf`
  Also, try `man 3 string`
- The standard library is automatically included by the C compiler but we do have to include the appropriate header file

Character Input and Output

- Text input or output is a stream of characters. A stream is a sequence of characters divided into lines; each line consists of zero or more characters followed by a newline character
- A text file is a file consisting of lines of characters separated by the newline character.
- The C standard library provides two functions for basic character input/output (in the `<stdio.h>` header file)

```
c = getchar(); //reads character from standard input
putchar(c); //writes the character to standard output
```

- Character input and output examples:
  - File copy
  - Couting the number of characters
  - Counting the number of lines
  - Counting the number of words

File Copy example

```
/* C-examples/intro/cp1.c */
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int c; //why is this int and not char?

    c = getchar();
    while (c != EOF ) {
        putchar(c);
        c = getchar();
    }
    return 0;
}
```

Test using *file redirection* in the terminal.

```
gcc -Wall -o cp1 cp1.c
cp1 < file1 > file1.copy
```

```c
/* C-examples/intro/cp2.c */

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

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

    /* The parentheses around c = getchar() are required because
       the operator != has higher precedence than = operator */

    while ((c = getchar()) != EOF )
        putchar(c);

    return 0;
}
```

Exercise 1-7(modified). Modify above program to print the value of EOF.
How to simulate EOF in keyboard input? Use `Ctrl-d` in Linux.

Character Counting

```c
/* C-examples/intro/wc1.c */
#include <stdio.h>
#include <stdlib.h>

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

    nc = 0;
    while (getchar() != EOF ) {
        nc++;
    }
    printf("%ld\n", nc);
    return 0;
}
```

Line Counting

```c
/* C-examples/intro/wc2.c */
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int c;
    long nl;

    nl = 0;
    while ((c = getchar()) != EOF )
        if (c == '\n')
            nl++;
    printf("%ld\n", nl);
    return 0;
}
```

Word Counting

```
/* C-examples/intro/wc3.c */
#include <stdio.h>
#include <stdlib.h>
const int IN=1;  /* inside a word */
const int OUT=0; /* outside a word */
/* count number of characters, words and lines in the standard input */
int main(int argc, char *argv[])
{
    int c;
    long nc, nw, nl;
    int state;

    state = OUT;
    nl = nw = nc = 0;
    while ((c = getchar()) != EOF ) {
        nc++;
        if (c == '\n')
            nl++;
        if (c == ' ' || c == '\n' || c == '\t')
            state = OUT;
        else if (state == OUT) {
                state = IN;
                nw++;
        }
    }
    printf("%ld %ld %ld\n", nl, nw, nc);
    return 0;
}
```

Exercise 1-11. How would you test the word count program? What kinds of input are most likely to uncover bugs if there are any?

Arrays

- Write a program to count the number of occurrences of each digit, of white space characters (blank, tab, newline), and of all other characters.
- This example illustrates use of simple arrays, character manipulation and more complex if-else statements.

Arrays

C-examples/intro/count-digits.c

```
#include <stdio.h>
/* count digits, white space, others */
int main()
{
    int c, i, nwhite, nother;
    int ndigit[10];

    nwhite = nother = 0;
    for (i = 0; i < 10; ++i)
        ndigit[i] = 0;


    while ((c = getchar()) != EOF)
        if (c >= '0' && c <= '9')
            ++ndigit[c-'0'];
        else if (c == ' ' || c == '\n' || c == '\t')
            ++nwhite;
        else
            ++nother;

    printf("digits =");
    for (i = 0; i < 10; ++i)
        printf(" %d", ndigit[i]);
    printf(", white space = %d, other = %d\n", nwhite, nother);
    return 0;
}
```

Command Line Arguments

```
/*  C-examples/intro/cmdline.c */
#include <stdio.h>
#include <stdlib.h>
/*
   We are expecting 3 command line arguments: the first one a string,
   the next an integer and the last a double. The name of the executable
   is always passed in as the first command line argument, so we have a
   total of 4 command line arguments.
*/
int main(int argc, char *argv[])
{
    int i;
    if (argc != 4) {
        fprintf(stderr, "Usage: %s <string> <int> <float>\n", argv[0]);
        exit(1);
    }
    printf("argument %d = %s\n", i, argv[0]);
    printf("argument %d = %s\n", i, argv[1]);
    printf("argument %d = %d\n", i, atoi(argv[2]));
    printf("argument %d = %f\n", i, atof(argv[3]));
    return 0;
}
```

Note that `atoi` and `atof` are functions in the standard library. Read their man page to find out more

Recommended Exercises

- **Exercise 1-8**. Write a program to count blanks, tabs, and newlines.
- **Exercise 1-9**. Write a program to copy its input to its output, replacing each string of one or more blanks by a single blank.
- **Exercise 1-12**. Write a program that prints its input one word per line.
- Add a command line options to the third word count program `wc3.c`. The options are `-l` to print line count only, `-w` to word count only, `-c` to print character count only. If more than one of these options is passed, then combine the results. Also add a command line option `-help` to display an appropriate help message and exit.
- **Exercise 1-23**. Write a program to remove all comments from a C program. Don't forget to handle quoted strings and character constants properly. C comments do not nest.