CSCB09 Software Tools and Systems Programming

C – I/O

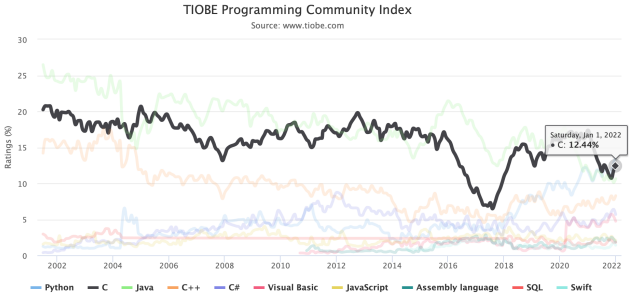
Marcelo Ponce

Winter 2023

Department of Computer and Mathematical Sciences - UTSC

C Programming Language

- Widely used and important.
- Impressive for a language that was invented in 1972!
- We will use the standard `c99`.
- Used in a huge range of disciplines, from OS, controlers, upto applications in scientific disciplines.



`https://www.tiobe.com/tiobe-index/`

Why C?

- What do you think about C so far ...?
- Disappointed about the lack of support for strings, arrays, etc.... ?
- Turns out you also have to do much of memory management manually ....

Why C?

- What do you think about C so far ...?
- Disappointed about the lack of support for strings, arrays, etc.... ?
- Turns out you also have to do much of memory management manually ....

Your ride with Python

- Cruise control
- Seat heating
- Cup holders

Your ride with C

- Performance and speed
- But... no amenities... (?!)
- and it only comes with a stickshift...
- ...

I/O

I/O between User and Program

- From A48, output to terminal (`STDOUT`): `printf()`
- STDOUT, STDERR, STDIN
- What about input from keyboard?
  - Command line input
  - STDIN input

Exercise 1

- Implement a simplified version of the `wc` (word count) utility with only one functionality:
- Your program will use `scanf` to count the number of words (strings) the user inputs and print the count to the screen.
- The user signals the end of input (`EOF` = End of File) by hitting Ctrl-D (Ctrl-Z on Windows) on a new line
- (For simplicity you may assume no input word is longer than 20 characters)

```c
#include <stdio.h>

int main() {

  // YOUR CODE HERE !!!

}
```

Exercise 1 – solution

```c
#include <stdio.h>

int main(){
  char input_string[20];
  int counter = 0;

  while (scanf("%s", input_string) != EOF) {
    counter = counter + 1;
  }

  printf("%d\n", counter);
}
```

CLA: Command Line Arguments

Command Line Arguments

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

- What is: `char **arg` ?

Quick review of string and array facts

- What is a string?
  - Array of chars
  - Null character ('\0') denotes the string end
  - C library functions to operate on strings, `<string.h>`

Quick review of string and array facts

- What is a string?
  - Array of chars
  - Null character ('\0') denotes the string end
  - C library functions to operate on strings, <string.h>

```
char name[20];
strncpy(name, "CSCB09", 20);
// NOT: name = "CSCB09";
```

  - What is the type of name?

Quick review of string and array facts

- What is a string?
  - Array of chars
  - Null character ('\0') denotes the string end
  - C library functions to operate on strings, `<string.h>`

```
char name[20];
strncpy(name, "CSCB09", 20);
// NOT: name = "CSCB09";
```

  - What is the type of `name`? $\longrightarrow$ `char*`

Quick review of string and array facts

- What is a string?
  - Array of chars
  - Null character ('\0') denotes the string end
  - C library functions to operate on strings, `<string.h>`

```
char name[20];
strncpy(name, "CSCB09", 20);
// NOT: name = "CSCB09";
```

  - What is the type of `name`? $\longrightarrow$ `char*`
- How would you declare an array of 30 strings?

Quick review of string and array facts

- What is a string?
  - Array of chars
  - Null character ('\0') denotes the string end
  - C library functions to operate on strings, <string.h>

```
char name[20];
strncpy(name, "CSCB09", 20);
// NOT: name = "CSCB09";
```

  - What is the type of name? → char*
- How would you declare an array of 30 strings?

```
char* name_list[30];
name_list[0] = name;
```

Quick review of string and array facts

- What is a string?
  - Array of chars
  - Null character ('\0') denotes the string end
  - C library functions to operate on strings, `<string.h>`

```
char name[20];
strncpy(name, "CSCB09", 20);
// NOT: name = "CSCB09";
```

  - What is the type of `name`? $\longrightarrow$ `char*`
- How would you declare an array of 30 strings?

```
char* name_list[30];
name_list[0] = name;
```

- What is the type of `name_list`?

Quick review of string and array facts

- What is a string?
  - Array of chars
  - Null character ('\0') denotes the string end
  - C library functions to operate on strings, <string.h>

```
char name[20];
strncpy(name, "CSCB09", 20);
// NOT: name = "CSCB09";
```

  - What is the type of name? → char*
- How would you declare an array of 30 strings?

```
char* name_list[30];
name_list[0] = name;
```

- What is the type of name_list? → char**

Command Line Arguments

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

- What is: `char **arg` ?

Command Line Arguments

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

- What is: `char **arg` ?
- An array of strings
- Need `argc` to know how many strings to expect in array.
- `argv[1]` is the first command line argument
- and so, what is `argv[0]` then?

Exercise 2

Adding CLAs to wc...

- Extend your word count program to take exactly one command line argument.
- If the commandline argument is "-w" your program will print the number of words in user input (as before)
- If the commandline argument is "-c" your program will print the number of characters in user input
- As before, the user signals the end of input (EOF = End of File) by hitting Ctrl-D (Ctrl-Z on Windows) on a new line
- (For simplicity you may assume that the user does provide exactly one command line argument).

Exercise 2 – solution

```c
#include <stdio.h>  // basic IO, eg. printf
#include <string.h> // for strings manipulation

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

  char input_string[20];
  char input_c;
  int counter = 0;

  // CLA processing: -w _ words
  if (strcmp(argv[1], "-w") == 0) {
    while (scanf("%s", input_string) != EOF) {
      counter = counter + 1;  // increment counter
    }
    printf("%d␣\n", counter);
    return 0;
  }
```

```c
#include <stdio.h>  // basic IO, eg. printf
#include <string.h> // for strings manipulation

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

  char input_string[20];
  char input_c;
  int counter = 0;

  // CLA processing: -w _ words
  if (strcmp(argv[1], "-w") == 0) {
    while (scanf("%s", input_string) != EOF) {
      counter = counter + 1;  // increment counter
    }
    printf("%d \n", counter);
    return 0;
  }

  // CLA processing: -c _ characters
  if (strcmp(argv[1], "-c") == 0) {
    while (scanf("%c", &input_c) != EOF) {
      counter = counter + 1;  // increment counter
    }
    printf("%d \n", counter);
    return 0;
        }

  printf("Invalid argument!");
}
```

1. In real program: check that `argc==2`
2. In real program: read at most 20 chars `scanf("%20s", ...)`
3. Improvements: ...

I/O between user and program

- From A48, output to terminal (`STDOUT`): `printf()`

I/O between user and program

- From A48, output to terminal (`STDOUT`): `printf()`
- What about input from terminal?

I/O between user and program

- From A48, output to terminal (`STDOUT`): `printf()`
- What about input from terminal?
  - Command line input
  - `STDIN` input

I/O between user and program

- From A48, output to terminal (`STDOUT`): `printf()`
- What about input from terminal?
  - Command line input
  - `STDIN` input
- What about **file I/O**?

I/O between user and program

- From A48, output to terminal (`STDOUT`): `printf()`
- What about input from terminal?
  - Command line input
  - `STDIN` input
- What about **file I/O**?
  - Turns out you were doing file I/O already with `printf` & `scanf`

```
int printf(const char *format, ...)  // writes to STDOUT
int scanf(const char *format, ...)  // Reads from STDIN
```

I/O between user and program

- From A48, output to terminal (`STDOUT`): `printf()`
- What about input from terminal?
  - Command line input
  - `STDIN` input
- What about **file I/O**?
  - Turns out you were doing file I/O already with `printf` & `scanf`

```
int printf(const char *format, ...)  // writes to STDOUT
int scanf(const char *format, ...)  // Reads from STDIN
```

  - For general file I/O need to specify where to read to / write from:

```
int fprintf(FILE *stream, const char *format, ...)
int fscanf(FILE *stream, const char *format, ...)
```

File interfaces in C/Unix

- Two main mechanisms for managing open files:

File interfaces in C/Unix

- Two main mechanisms for managing open files:
- File *descriptors* (low-level, managed by OS):
  - Each open file is identified by a small integer
  - STDIN is 0, STDOUT is 1
  - Use for pipes, sockets (will see later what those are ...)
  - (Remember how I said in first lecture that everything is a file ..?)

File interfaces in C/Unix

- Two main mechanisms for managing open files:
- File *descriptors* (low-level, managed by OS):
  - Each open file is identified by a small integer
  - STDIN is 0, STDOUT is 1
  - Use for pipes, sockets (will see later what those are ...)
  - (Remember how I said in first lecture that everything is a file ..?)
- File *pointers* (aka streams, file handles) for regular files:
  - A C language construct for easier working with file descriptors
  - You use a pointer to a file structure (FILE *) as handle to a file.
  - The file struct contains a file descriptor and a buffer.
  - Use for regular files

Operating on regular files

- Let's say we want to read from/write to a file `my_file.txt`
- For such, we need a file *handle*: `FILE *`

Operating on regular files

- Let's say we want to read from/write to a file `my_file.txt`
- For such, we need a file *handle*: `FILE *`
- A file first needs to be opened to obtain a `FILE *`

```
FILE *fopen(const char *filename, const char *mode)
```

  - `filename` identifies the file to open
  - `mode` tells how to open the file:
    `"r"` for reading, `"w"` for writing, `"a"` for appending
  - returns a pointer to a FILE struct which is the handle to the file.
  - To close a file: `void fclose(FILE *stream);`

```
FILE *fp = fopen("my_file.txt", "w");
fprintf(fp, "Hello!\n");
fclose(fp);
```

Exercise 3

- Extend our Exercise 1 word count program to take exactly one command line argument, which is a file name.
- This program will open the file, count the number of words in it and print the count.

```c
#include <stdio.h>

int main(){
  char input_string[20];
  int counter = 0;

  while (scanf("%s", input_string) != EOF) {
    counter = counter + 1;
  }

  printf("%d\n", counter);
}
```

Exercise 3

- Extend our Exercise 1 word count program to take exactly one command line argument, which is a file name.
- This program will open the file, count the number of words in it and print the count.

```c
#include <stdio.h>

int main(){
  char input_string[20];
  int counter = 0;

  while (scanf("%s", input_string) != EOF) {
    counter = counter + 1;
  }

  printf("%d\n", counter);
}
```

```c
FILE *fopen(const char *path, const char *mode);
int fscanf(FILE *stream, const char *format,...);
int fclose(FILE *stream);
```

Exercise 3 – solution

```c
#include <stdio.h>

int main(int argc, char **argv){
  char input_string[20];
  int wc = 0;
  FILE *input_file;

  input_file = fopen(argv[1], "r");
  // In real programs, error check after opening file:
  // if (input_file == NULL) then there was an error
  while( fscanf(input_file, "%s", input_string) != EOF) {
    wc = wc + 1;
  }

  fclose(input_file);
  printf("%d\n", wc);
}
```

Basic file I/O functions (stdio.h)

```
FILE *fopen(const char *path, const char *mode);

int fscanf(FILE *stream, const char *format,...);

char *fgets(char *s, int size, FILE *stream);

char fgetc(FILE *stream);

int fprintf(FILE *stream, const char *format,...)

int fputs(const char *str, FILE *stream);

int fseek(FILE *stream, long int offset, int whence);

void rewind(FILE *stream);

int fclose(FILE *stream);
```