FEEG 6002

Advanced Computational Methods I

Author: Yusaf Sultan
Lecturers: Ranga Dinesh Kahanda Koralage, Richard Boardman, Ondrej Hovorka
Word Count: 2438

Contents

List of Weeks

List of Figures

List of Tables

1. Lecture 1

START OF WEEK 1

1.1. Hello World in C using Quincy

The code required to create an executable file which prints hello world to the console is shown in **Figure 1**.

```c
#include <stdio.h>

int main()
{
  printf("Hello World \n");
  return 0;
}
```

Figure 1: C code which prints hello world to terminal

- **Line 1**: Imports the header file called the **standard input output library** which contains the **`printf`** function that is utilized later in the code. This line is ran before the code is compiled and pastes in all of the functions within the header file.
- **Line 3**: Defines the **`main`** function which is always where execution starts for a C program. The function is defined as an integer using **`int`** and returns 0 to the OS upon successful execution.
- **Line 5** Utilizes the **`printf`** function to print "hello world" to the terminal. This function requires **`\n`** at the end to start a new line and each line of a C program requires a semicolon to signify the end of a line of code.

If this code is created as a C source file in Quincy it can be **saved** using **`ctrl + s`**, **compiled** using **`F5`** and than **executed** using **`F9`**. This will print the hello world text in the Quincy terminal as shown in **Figure 2**.

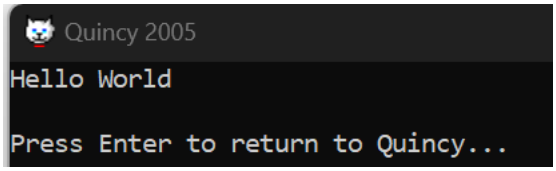

Figure 2: **Figure 1** executable file output

When pressing **`F5`** Quincy sends a command to the C compiler which contains the following key flags:

- **`-ansi`**: Ensures compatibility with C90 standard.
- **`-Wall`**: Enables all warnings.
- **`-pedantic`**: Issues all warnings ISO C requires (C90 when using -ansi).

1.2. Using the Command Line

The command line is a **non-graphical way of working with a computer, via screen and keyboard.** There exist many different terminals, some common window terminal commands are shown below:

- `dir` : Display all files in the current folder.
- `cd \` : Navigate to root folder.
- `cd \<dirname>` : Navigate to the specified folder.
- `cd ..` : Move up one folder.
- `mkdir <newdir>` : Make a new folder.
- `rmdir <rmdir>` : Remove a folder.
- `rename <old> <new>` : Rename a file or folder.
- `move <file1> <file2>` : Move file1 to file2.
- `del <file>` : Delete file.
- `cls` : Clear screen.
- `help` : Shows all commands.
- `copy <src> <dest>`: "Copy a file from src to dest.",
- `type <file>`: "Print the contents of a file.",
- `echo <text>`: "Print text to the console.",
- `exit`: "Close the command prompt.",
- `path`: "Show system PATH variable.",
- `ipconfig`: "Display network configuration.",
- `ping <host>`: "Send ICMP packets to a host.",
- `tasklist`: "List running processes.",
- `taskkill /IM <process> /F`: "Kill a running process.",
- `systeminfo`: "Display detailed system information."

2. Lecture 2

2.1. Command Line

It was previously states that a command line is a non-graphical way of interacting with the computer using the screen and a keyboard. The command line like many other programming languages is a **REPL** type of program, which stands for:

- **Read**: read user input.
- **Evaluate**: run the command.
- **Print**: show the output.
- **Loop**: wait for the next command.

The command line can be used to manually perform many OS tasks (moving, renaming, deleting files and folders etc.,). Alternatively a bash file (`.bat`) can be used to chain together commands and automate scripts.

GUIs are not easy to program, often add unnecessary complexity to the program and are difficult to automate. The command line is therefore an attractive alternative for user interaction. The shell allows for interaction with the command line and also allows for many different programs to interact with one another. To allow for interactivity, Shell's use the **Unix philosophy** which states:

- Write programs that do one thing and do it well.
- Write programs to work together.
- Write programs to handle text streams (universal interface).

2.2. Input Output (I/O) Streams

The input and output of programs is done via a stream. There exists three standard streams for OSs which are shown in **Table 1**.

Standard Stream	Abbreviation	File Descriptor	Connected to
Input	`stdin`	0	Keyboard
Output	`stdout`	1	Screen
Error	`stderr`	2	Screen

Table 1: Standard streams

2.3. Program I/O in C

The built in C library `stdio` defines these streams. As well as some functions which make use of these streams, as shown below:

- `printf()` → Utilizes the `stdout` to print to the terminal.
- `fprintf()` → Takes an argument (either `stdout` or `stderr`) to either print out to standard out or standard error.
- `getchar()` → Get one character from `stdin` (note that this saves as the ASCII value of the character).
- `putchar()` → Send one character to `stdout` (note that this takes the ASCII number and outputs the corresponding character).

One common tool to stop an infinitely looping code is to use the **EOF** (end of file) character, which can be inputted in a windows shell by doing **ctrl + Z** and then pressing **Enter**.

Note that for **fprintf()** if **stdout** is passed then it behaves the same as **printf**. If **stderr** is instead passed than an **unbuffered, undirectable** output is printed to the shell.

Applying these functions in one program, a parrot code can be created as shown in **Figure 3** which returns the inputted text from the terminal.

```c
#include <stdio.h>
int main()
{
  int c;
  fprintf(stderr, "Enter some characters (Ctrl-D quits)\n");
  while ((c = getchar()) != EOF) {
    putchar(c);
  }
  fprintf(stderr, "Goodbye\n");
  return 0;
}
```

Figure 3: C code which parrots the input and prints it back out to the terminal.

2.4. Program I/O in Terminal

The **echo** command can be used to print things within the terminal (**stdout**), the standard output can be redirected and written to a file using **>** this is shown in **Figure 4**

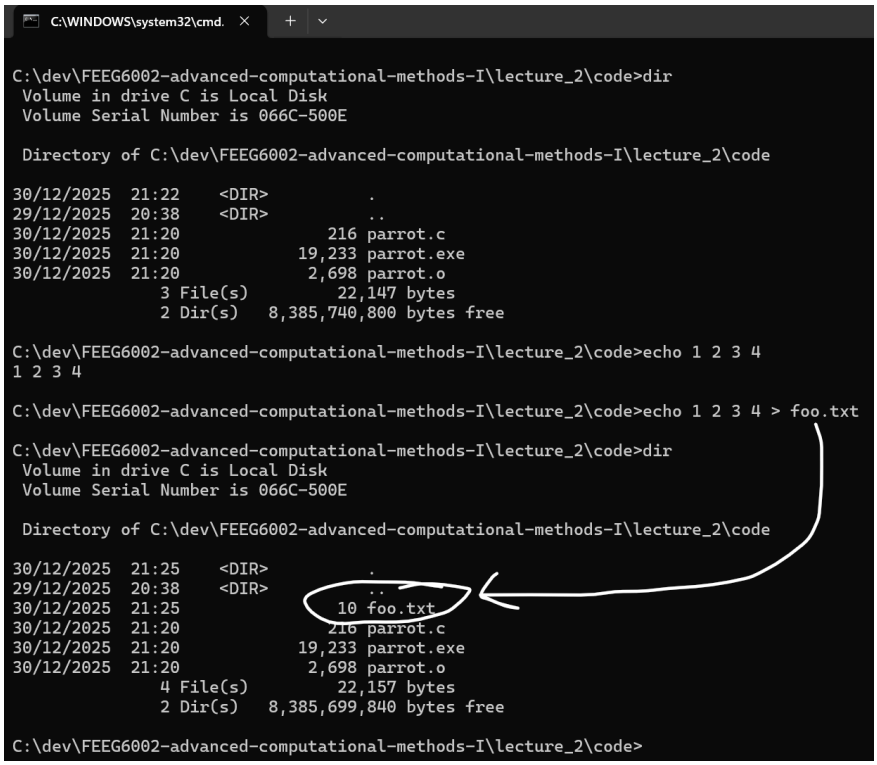

Figure 4: Redirecting terminal **stdout**.

To append to a text file **>>** can be used instead. The **stdin** can also be redirected using the **<** operator, this is shown in **Figure 5**.

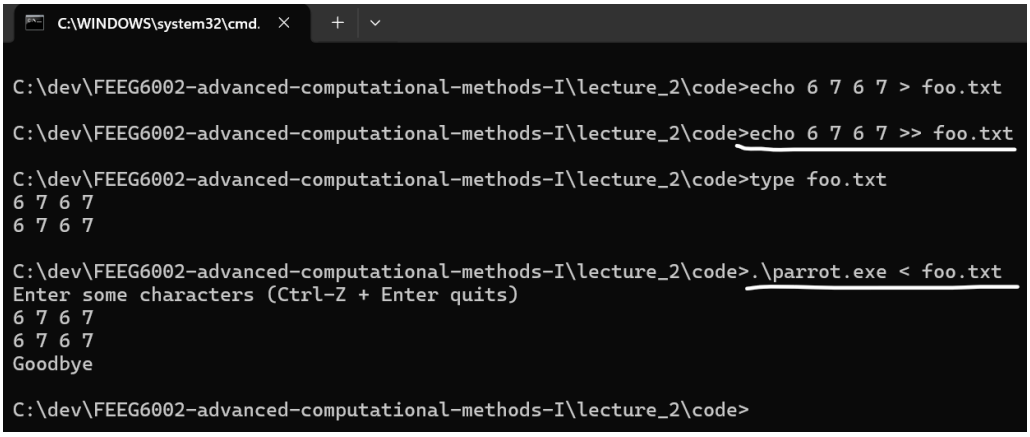

Figure 5: Redirecting terminal **stdin** into a executable file.

The output of the **parrot.exe** can itself be redirected into another text file by utilizing both the **>** command, as shown in **Figure 6**.

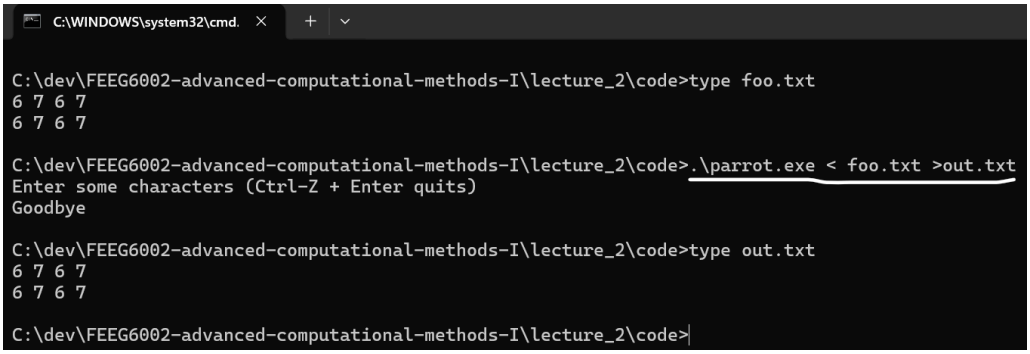

Figure 6: Redirecting program **stdout** into a text file.

Note that the **stderr** from the program is still outputted to the shell, this can also be redirected using the **2** file descriptor, this is shown in **Figure 7**.

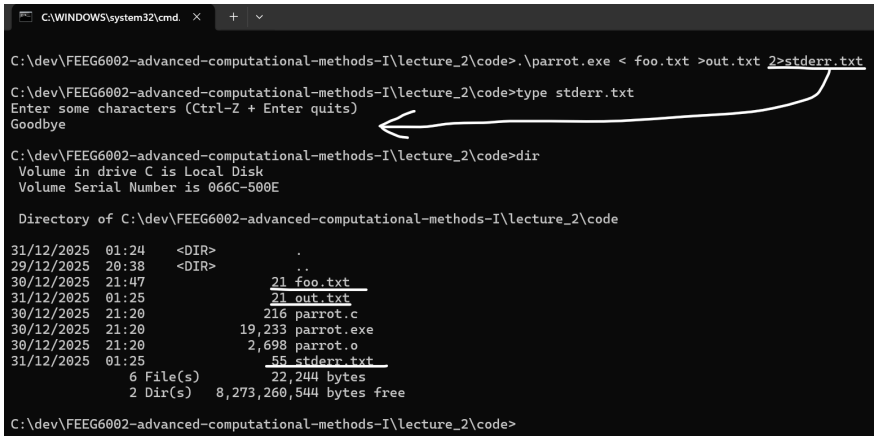

Figure 7: Redirecting program **stdout** and **stderr** into a text file.

Note that if anything is ever re-routed to **`dev\null`** then this output is quietly destroyed. Pipes can be used to chain together commands, effectively chaining the **`stdout`** of one command to the **`stdin`** of the second command. An example of using pipes to chain together commands is shown in **Figure 8**.

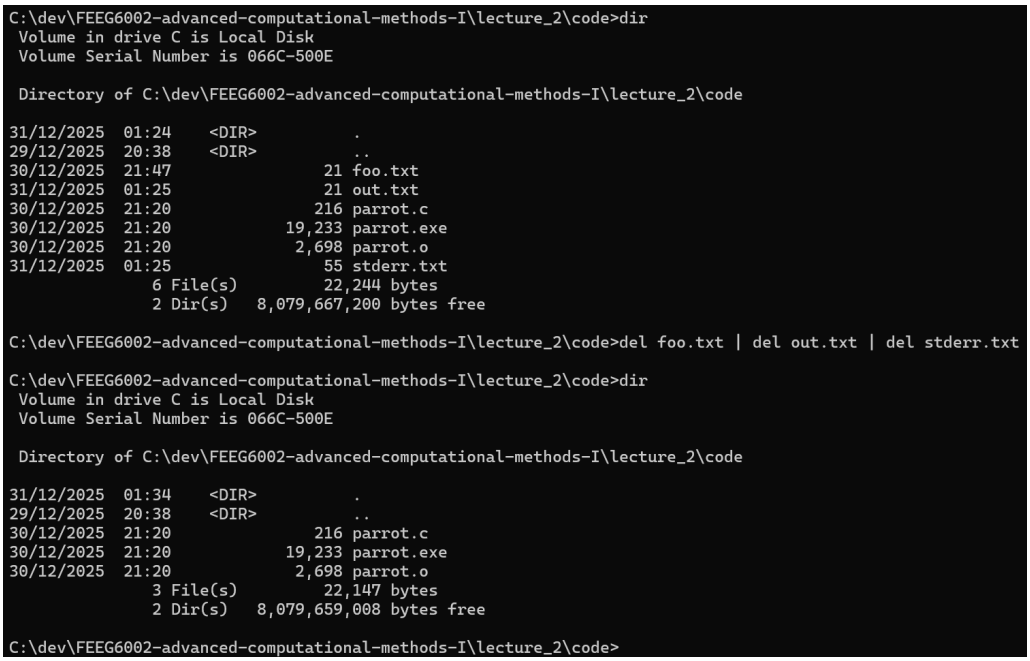

Figure 8: Utilizing pipes in the terminal to chain commands.

3. Lecture 3

START OF WEEK 2

4. Lecture 4

4.1. Basic Data Types

There exists 4 key datatypes within C, these are:

- **`int`** → Integer or a whole number. Uses 4 bytes (32 bits) of memory with 1 sign bit and 31 magnitude bits. Can store numbers in the range of $-2,147,483,648$ to $+2,147,483,647$
- **`char`** → Stores a single character using 1 byte (4 bits). Stores the character as an ASCII code in memory.
- **`float`** → Stores a decimal number in 4 bytes (32 bits). Called single precision, this allows for 6 - 7 decimal digits and has a range from 1.175×10^{-38} to 3.402×10^{38}.
- **`double`** → Stores a decimal number in 8 bytes (64 bits). Called double precision, this allows for 15 - 16 decimal digits and has a range from 2.225×10^{-308} to 1.798×10^{308}.
- **`bool`** → Can be True or False (1 or 0).

4.2. Signed, Unsigned, Short and Long Datatypes

`int` and **`char`** datatypes can be signed or unsigned, by default they are both signed. Declaring an integer as unsigned means it can **only store non-negative values**. This means instead of the range being centered at zero, all bits are used to store the number. For a 32 bit integer, this means the range for this variable is 0 to 4,294,967,295.

An integer can also be defined as **`short`**, **`long`** and **`long long`**, which specify how many bits are used to store the number. The number of bits for each type as well as their maximum ranges are show in **Table 2**.

Keyword	Windows Size	Linux/macOS Size	Signed	Unsigned Range
short	16-bit	16-bit	±32k	0 - 65k
int	32-bit	32-bit	±2.1B	0 - 4.2B
long	32-bit	64-bit	varies	varies
long long	64-bit	64-bit	±9e18	0 - 1.8e19

Table 2: Integer type sizes and ranges

4.3. Using `printf` to Print Variables

The **`printf`** command can be used to print out all of the variable types perviously mentioned. On top of this, formatting can also be used to print specific outputs. An example code which shows this is shown in **Figure 9**.

```c
# include <stdio.h>

int main(void) {
  int var1 = -42;
  unsigned int var2 = 142;
  float var3 = 3.14;
  double var4 = 3.1428;
  char var5[] = "Hello World";

  printf("Printing int var1=%d\n", var1);
  printf("Printing unsigned int var2=%u\n", var2);
  printf("Printing float var3=%f\n", var3);
  printf("Printing double var4=%f\n", var4);
  printf("Printing double in scientific notation var4=%e\n", var4);
  printf("Printing char[] var5=%s\n", var5);
  return 0;
}
```

```shell
C:\>. \printing_data_types.e
Printing int var1=-42
Printing unsigned int var2=142
Printing float var3=3.140000
Printing double var4=3.142800
Printing double in scientific notation var4=3.142800e+000
Printing char[] var5=Hello World
```

Figure 9: C code which assigns each variable type and prints it.

Further formatting can be achieved by using specific formatters, this is shown in **Figure 10**.

```c
#include <stdio.h>

int main(void) {

  double pi = 3.1415926535897931;

  printf("As a standard float representation: pi=%f\n", pi);
  printf("In exponential notation: pi=%e\n", pi);
  printf("Whichever of the above two is shorter: pi=%g\n", pi);
  printf("Request 10 digits overall: pi=%10f\n", pi);
  printf("With 3 postdecimal digits: pi=%10.3f\n", pi);
  printf("12 postdecimal digits: pi=%.12f\n", pi);

return 0;
}
```

```shell
C:\>. \printing_data_types.e
As a standard float representation: pi=3.141593
In exponential notation: pi=3.141593e+000
Whichever of the above two is shorter: pi=3.14159
Request 10 digits overall: pi= 3.141593
With 3 postdecimal digits: pi=     3.142
12 postdecimal digits: pi=3.141592653590
```

Figure 10: C code which customizes the output of **`printf`** using formatting flags.

A summary of which formatters are used with specific datatype is shown below:

- **%d** & **%i** → Signed **int**, **short**, **chars**.
- **%ld** & **%li** → Signed **long** integers.
- **%f** → Floating point representation of **double** or **float**.
- **%e** → Exponential notation of **double** or **float**.
- **%g** → **f** or **e** depending on which is shorter.
- **%s** → Arrays of characters (**char**, **char[]**).
- **%u** → Unsigned integers (unsigned **int**, **short**, **char**).
- **%<n1>.<n2>f** → **n1** overall digits and **n2** decimal digits (This can result in whitespace).

4.4. Using scanf

The **scanf** command is useful for allowing the user to input a value which can then be saved as a variable, an example program which takes a character and integer input is shown in **Figure 11**.

```c
#include <stdio.h>

int main(void)
{
    int inputted_numb;
    char inputted_char;

    printf("Enter an integer.\n");
    scanf("%d", &inputted_numb);
    printf("Inputted number = %d\n", inputted_numb);

    printf("Enter a character.\n");
    scanf(" %c", &inputted_char);
    printf("Inputted character = %c\n",
    inputted_char);

    return 0;
}
```

```shell
C:\>.
\scanf_example.exe
Enter an integer.
67
Inputted number = 67
Enter a character.
B
Inputted character =
B
```

Figure 11: C code which uses **scanf** to take in a user input.

Note that for this code to run a space is required in line 13 before **%c** as the **\n** from the previous line effects it. Furthermore, the **&** refers to the memory address of the variable, not its value.

4.5. Conditional Statements in C

if-else and **if-else if-else** statements can be used to run a specific block of code when a condition is met. There are two types of conditions that can be specified within a conditional, these are **relational operators** and **logical operators**. These can be combined together to form composite statements, all of these operators are shown below:

- **Relational operators**:
- **a==b** → a equal to b
- **a!=b** → a not equal to b
- **a>b** → a greater than b
- **a<b** → a less than b
- **a>=b** → a greater than or equal to b
- **a<=b** → a less than or equal to b

- **Logical operators**:
- **&&** → logical AND
- **||** → logical OR
- **!** → logical NOT

An example code that uses composite statements using multiple operators is shown in **Figure 12**.

C

```c
#include <stdio.h>

int main(void)
{
    int user_input;

    printf("Enter an integer.\n");
    scanf(" %d", &user_input);

    if (user_input > -10 && user_input < 0) {
        printf("Input is between -10 and 0\n");
    } else if (user_input == 5) {
        printf("Input is 5\n");
    } else {
        printf("The input is: %d\n", user_input);
    }

    return 0;
}
```

Shell

```
C:\>.\conditional_statements.exe
Enter an integer.
5
Input is 5
C:\>.\conditional_statements.exe
Enter an integer.
-7
Input is between -10 and 0
C:\>.\conditional_statements.exe
Enter an integer.
52
The input is: 52
```

Figure 12: C code which uses conditional statements.

5. Lecture 5

5.1. For Loop

A for loop in C requires three input conditions, these are:

- **Initialization** → An expression that is executed once at the start of the loop, often used to initialize a counter variable.
- **Condition** → An expression that is checked before every iteration. If the condition is true, the loop executes, if false, then the loop stops.
- **Update** → An expression applied after each iteration of the loop. It typically increments or updates the loop variable.

Some basic code which utilizes a for loop is shown in **Figure 13**. Note that **`i++`** is equivalent to **`i = i + 1`**.

C

```c
#include <stdio.h>

int main(void){

  int i;
  int N = 5;

  for (i=0; i<N; i++){
    printf("This is iteration %d of %d.\n",
    i, N-1);
  }
  return 0;
}
```

Shell

```
C:\>.\basic_for.exe
This is iteration 0 of 4.
This is iteration 1 of 4.
This is iteration 2 of 4.
This is iteration 3 of 4.
This is iteration 4 of 4.
```

Figure 13: C code which uses a for loop.

Note that using **`++i`** ≠ **`i++`**. Consider the code shown in **Figure 14**.

C

```c
#include <stdio.h>

int main(void) {
    int a = 5;
    int b = 5;

    printf("++a (pre-increment): value = %d\n", ++a);
    printf("b++ (post-increment): value = %d\n", b++);
    printf("b after post-increment = %d\n", b);
    return 0;
}
```

Shell

```
C:\>.\pre_post_increment
++a (pre-increment): value = 6
b++ (post-increment): value = 5
b after post-increment = 6
```

Figure 14: C code which uses a for loop.

5.2. While Loop

A while loop will iterate as long as a defined condition is not met. Note that there is a risk of infinite looping if the condition is never met. An example of a while loop in C is shown in **Figure 15**.

```c
#include <stdio.h>

int main(void) {
  int i = 1;

  while (i < 10) {
    printf("i = %d \n", i);
    i = i * 2;
  }
  return 0;
}
```

```shell
C:\>.\while_loop.exe
i = 1
i = 2
i = 4
i = 8
```

Figure 15: C code which uses a while loop.

Typically a **for loop** is used when the **number of iterations is known**. On the other hand, a **while loop** when the number of iterations is unknown but there is some sort of **criteria or tolerance** that can be built into a condition.

5.3. Buffering

The standard output **`stdout`** is often buffered, meaning it stores the output data in a buffer and writes it out all at once. The standard error **`stderr`** is typically unbuffered, which means error messages appear immediately.