CS211: Programing For Operating Systems

Robert Davidson

Contents

1 Intro to C

It is a very small language and relies heavily on libraries The compiler must be told in advance how these functions should be used. So before the compilation process, the **preprocessor** is run to include the function prototypes The compiler then compiles the code into an object file.

1.1 Hello World

Listing 1: Hello World in C

```
1  #include <stdio.h>
2  int main(){
3      printf("Hello World\n");
4      return 0;
5  }
6
```

- **Line 1** : `#include <stdio.h>` is a preprocessor directive that tells the compiler to include the standard input/output library. This library contains the `printf` function.
- In C almost every line it either preprocessor directive, variable declaration, or a function call.
- C uses curly braces to delimit blocks of code and semicolons to terminate statements.
- **Line 4**: In our case, we assume main is called by the Operating System, so return 0 is used to indicate that the program has run successfully.

1.2 Variables

In C all variables must be declared before they are used. The declaration should have a type; telling the compiler what sort of data the variable will hold. The types of variables are:

- **int** : Integer (1, 2, 3, 4, 5, ...)
- **float** : Floating-point number (7 decimal digits)
- **double** : Double-precision floating-point number (15 decimal digits)
- **char** : Character (a, b, c, ...)
- **void** : No type (used for functions that do not return a value)

We can Also declare arrays as follows:

Listing 2: Declaring Arrays

```
1  int arr[5]; // Array of 5 integers
2  char name[10]; // Array of 10 characters
```

To access the first element of arr we can do `arr[0]`

1.3 An Example

Listing 3: Example of Variables

```
1  int d=-101;
2  float f=1.23456;
3  char c='a';
4  printf("Values of d, f, c are: %d, %f, %c\n", d, f ,c );
```

Explanation: In this case, `%d` is a placeholder for an integer, `%f` is a placeholder for a float, and `%c` is a placeholder for a character.

1.4 Operators

Operator	Description	Example
+	Addition	`a + b`
-	Subtraction	`a - b`
*	Multiplication	`a * b`
/	Division	`a / b`
%	Modulus	`a % b`

Table 1: Arithmetic Operators

Operator	Description	Example
=	Assignment	`a = b`
+=	Add and assign	`a += b`
-=	Subtract and assign	`a -= b`
*=	Multiply and assign	`a *= b`
/=	Divide and assign	`a /= b`
%=	Modulus and assign	`a %= b`
++	Increment	`a++`
–	Decrement	`a-`

Table 2: Assignment and Arithmetic Assignment Operators

Operator	Description	Example
==	Equal	`a == b`
!=	Not Equal	`a != b`
>	Greater	`a > b`
<	Less	`a < b`
>=	Greater or Equal	`a >= b`
<=	Less or Equal	`a <= b`

Table 3: Relational Operators

Operator	Description	Example
&&	Logical AND	`a && b`
||	Logical OR	`a || b`
!	Logical NOT	`!a`

Table 4: Logical Operators

1.5 Control Structure

Listing 4: If-Else

```
    int a = 10;
    if(a > 10){
            printf("a is greater than 10\n");
        }else if(a == 10){
            printf("a is equal to 10\n");
        }else{
    printf("a is less than 10\n");
    }

```

Logical opeators, `&&` and `||` can be used to make more complex conditions.

Listing 5: Complex If-Else

```
    if(a > 10 && a < 20){
        printf("a is between 10 and 20\n");
    }

```

1.5.1 For loop

`for(initial val; continuation condition; increment/decrement){...}`

Listing 6: Print numbers from 0 to 9

```
for(int i = 0; i < 10; i++){
    printf("i is %d\n", i);
}
```

1.5.2 While loop

`while(expression){...}`

Listing 7: Print numbers from 0 to 9

```
int i = 0;
while(i < 10){
    printf("i is %d\n", i);
    i++;
}
```

1.5.3 Do While loop

`do{...}while(expression);`

Listing 8: Print numbers from 0 to 9

```
int i = 0;
do{
    printf("i is %d\n", i);
    i++;
}while(i < 10);
```

1.6 Output

`printf()` is used to print formatted output to the screen. It is a variadic function, meaning it can take any number of arguments. The first argument is a format string, followed by the values to be printed.

The format string may contain a number of escape characters, represented by a backslash. Some of the most common escape characters are:

Sequence	Description
`\a`	Produces a beep or flash
`\b`	Moves cursor to last column of previous line
`\f`	Moves cursor to start of next page
`\n`	New line
`\r`	Carriage return
`\t`	Tab
`\v`	Vertical tab
`\\`	Prints a backslash
`\'`	Prints a single quote

A conversion character is a letter that follows a `%` and tells `printf()` to display the value stored in the corresponding variable. Some of the most common conversion characters are:

Specifier	Description
`%c`	Single character (char)
`%d` or `%i`	Decimal integer (int)
`%e` or `%E`	Floating-point (scientific notation)
`%f`	Floating-point value (float)
`%g` or `%G`	Same as %e/%E or %f, whichever is shorter
`%s`	String (char array)
`%u`	Unsigned int
`%x`	Hexadecimal integer
`%p`	Pointer (memory address)
`%%`	Prints the % character

1.7 Input

`scanf()` reads input from standat input, format it, as directed by a conversion character and store the address of a specified variable.

Listing 9: Reading an integer

```
    int number;
```

```
    char letter;
    printf("Enter a number and a char: ");
    scanf("%d %c", &number, &letter);

    printf("You entered: %d and %c\n", number, letter);
```

- The scan `scanf()` returns an integer equal to the number of successfull conversions made.
- There is related function `fscanf()` that reads from a file. `scanf()` is really just a wrapper for `fscanf()` that treats the keyboard as a file.
- There are other useful functions for readint the standard input stream: `getchar()` and `gets()`.

Listing 10: Check for no input

```
    int number;
    printf("Enter a number between 1 and 30: ");
    scanf("%d", &number);

    while ((number<1) || (number>30))
    {
        printf("Invalid number. Please enter a number between 1 and 30: ");
        scanf("%d", &number);
    }
```