

# **CS211: Programing For Operating Systems**

Robert Davidson

# Contents

<b>1</b>	<b>Intro to C</b>	<b>3</b>
1.1	Hello World . . . . .	3
1.2	Variables . . . . .	3
1.3	An Example . . . . .	3
1.4	Operators . . . . .	4
1.5	Control Structure . . . . .	4
1.5.1	For loop . . . . .	4
1.5.2	While loop . . . . .	5
1.5.3	Do While loop . . . . .	5
1.6	Output . . . . .	5
1.7	Input . . . . .	6
1.8	Functions . . . . .	6
1.9	Call-by-Value and Pointers . . . . .	6
1.10	Characters . . . . .	7
1.11	Strings . . . . .	7
1.11.1	String Functions . . . . .	7

# 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 is either a 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

```
1 int a = 10;
2 if(a > 10){
3     printf("a is greater than 10\n");
4 }else if(a == 10){
5     printf("a is equal to 10\n");
6 }else{
7     printf("a is less than 10\n");
8 }
9
```

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

Listing 5: Complex If-Else

```
1 if(a > 10 && a < 20){
2     printf("a is between 10 and 20\n");
3 }
4
```

### 1.5.1 For loop

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

Listing 6: Print numbers from 0 to 9

```
1 for(int i = 0; i < 10; i++){
2     printf("i is %d\n", i);
3 }
```

### 1.5.2 While loop

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

Listing 7: Print numbers from 0 to 9

```
1 int i = 0;
2 while(i < 10){
3     printf("i is %d\n", i);
4     i++;
5 }
```

### 1.5.3 Do While loop

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

Listing 8: Print numbers from 0 to 9

```
1 int i = 0;
2 do{
3     printf("i is %d\n", i);
4     i++;
5 }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
<code>\a</code>	Produces a beep or flash
<code>\b</code>	Moves cursor to last column of previous line
<code>\f</code>	Moves cursor to start of next page
<code>\n</code>	New line
<code>\r</code>	Carriage return
<code>\t</code>	Tab
<code>\v</code>	Vertical tab
<code>\\</code>	Prints a backslash
<code>\'</code>	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
<code>%c</code>	Single character (char)
<code>%d</code> or <code>%i</code>	Decimal integer (int)
<code>%e</code> or <code>%E</code>	Floating-point (scientific notation)
<code>%f</code>	Floating-point value (float)
<code>%g</code> or <code>%G</code>	Same as <code>%e/%E</code> or <code>%f</code> , whichever is shorter
<code>%s</code>	String (char array)
<code>%u</code>	Unsigned int
<code>%x</code>	Hexadecimal integer
<code>%p</code>	Pointer (memory address)
<code>%%</code>	Prints the <code>%</code> character

## 1.7 Input

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

Listing 9: Reading an integer

```
1  int number;
2  char letter;
3  printf("Enter a number and a char: ");
4  scanf("%d %c", &number, &letter);
5
6  printf("You entered: %d and %c\n", number, letter);
```

- The scan `scanf()` returns an integer equal to the number of successful 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 reading the standard input stream: `getchar()` and `gets()`.

Listing 10: Check for no input

```
1  int number;
2  printf("Enter a number between 1 and 30: ");
3  scanf("%d", &number);
4
5  while ((number<1) || (number>30))
6  {
7      printf("Invalid number. Please enter a number between 1 and 30: ");
8      scanf("%d", &number);
9  }
```

## 1.8 Functions

### Def. Prototype and Definition

**Prototype :** A function prototype is a declaration of a function that tells the compiler what the function looks like. It includes the function name, return type, and parameter types. The prototype must be declared before the function is called.

**Definition :** A function definition is the actual implementation of the function. It includes the function name, return type, parameter types, and the body of the function. The definition must be declared after the function is called.

## 1.9 Call-by-Value and Pointers

In C it is important to distinguish between a variable and the value stored in it. A variable has a location in memory. The value of the variable is stored in that location. For example:

```
1  int i = 10;
```

tells the system to allocate a location in memory to store the value 10. The variable `i` is a pointer to that location in memory. One of the distinguishing features of C is that we can manipulate the memory address of the variable almost as easily as we can manipulate the value stored in it.

### Def. Pointers

- if `i` is a variable, then `&i` is a pointer to the location in memory where the value of `i` is stored.
- The declaration `int *p;` creates a variable `p` that can store the memory address of an integer. The `*` indicates that `p` is a pointer to an integer.
- If a memory address is stored in the variable `p`, then `*p` is the value stored at that address.

## 1.10 Characters

### *Concept: Character representation*

In C, a character is just an unsigned integer. Each character is represented by an integer between 0 and 127.

#### Printing characters:

- `printf("%c", c)`
- `putchar(c)`

#### Reading characters:

- `scanf("%c", c)`
- `c = getchar()`

## 1.11 Strings

### *Concept: String representation*

C does not have a string data type. Instead it uses arrays of type `char` to represent strings. For example if we make a declaration:

```
char greeting[20]="Hello. How are you?";
```

the system stores each character as an element of the array `greeting`.

### 1.11.1 String Functions

Useful functions defined in `string.h`:

#### **strncpy()**

```
char *strncpy(char *dest, char *src, int n)
```

Copies at most `n` characters from `src` to `dest`. The advantage of this is that we don't copy more characters to `dest` than it can hold (prevents buffer overflow). `strncpy()` does not append a null character to the end of the string, so we need to do that manually.

#### **strcat()**

```
char *strcat(char *dest, char *src)
```

Concatenates the string `src` to the end of the string `dest`. The destination string must be large enough to hold the concatenated result. `strcat()` appends a null character to the end of the string.

#### **strcmp()**

```
int strcmp(char *str1, char *str2)
```

Compares two strings and returns an integer:

- 0 if the strings are equal
- A negative integer if `str1` comes first alphabetically
- A positive integer if `str2` comes first alphabetically

#### **strstr()**

```
char *strstr(char *haystack, char *needle)
```

Searches for the first occurrence of the string `needle` in the string `haystack`. If found, it returns a pointer to the first occurrence of `needle` in `haystack`. If not found, it returns `NULL`.

#### **strlen()**

```
int strlen(char *str)
```

Returns the length of the string `str` (not including the null character).