# Basic elements of a C program:

## Introduction:

* C is an intermediate level language.
* C is a procedural language.
* It supports low level features like bit operations and high level features like complex data structures.
* C provides access to the implementation detail with explicit memory management and explicit error detection
* Has better performance than java because of less runtime overhead
* Java is derived from C++, which, in turn, is derived from C; so for someone who knows Java, learning the basics of C is easy.

For a quick start, remember the following:

1. C is not object-oriented. So there are no classes.
2. C supports all of the primitive data types in Java except Boolean. The boolean values of **false** and **true** are implicitly represented by a **zero** and **non-zero** integral values respectively.
3. C contains a collection of functions, which are like static methods, although the keyword static has a different meaning. Don’t worry about the word static and don’t use it initially.
4. Parameters and return types for functions pretty much follow the rules of Java method parameters and return types.
5. Execution begins in a method called main, just as in Java applications. However, don’t worry about parameters to main (for the moment). Let main return an int.
6. Statements (assignments, **if...else**, **switch**, and the three loop structures) are very Java-like, but there are a few, slight differences, which shouldn’t worry you for now. **break** and **continue** are also available.

## C data types:

### Fundamental data types

1. int: an integer (same as a long int)
2. char: a single byte to hold a single character
3. float: single precision floating point
4. double double precision floating point

### Type modifier:

1. short, long (
2. unsigned, signed

Unlike in Java, the size taken up by C variables depends on the platform. The **sizeof** operator gives the number of bytes needed for a certain variable or type. Look below for the usage of operator **sizeof.**

**usage: sizeof(<type or variable>)**

Example: (on machines with 32-bit processors)

**int num;**

**int sz;**

**sz = sizeof(int); /\* sz will normally equal to 4 \*/**

**sz = sizeof(num); /\* sz will normally equal to 4 \*/**

The **short** and **long** qualifiers are applicable to the integer and double types

Example: (on machines with 32-bit processors)

**short int short\_num; /\* short\_num will often be 2 bytes long \*/**

**long int long\_num; /\* long\_num will be no less than 4 bytes long \*/**

**long long int really)long\_num /\* could do if needed \*/**

**long double long\_double; /\* long\_double will be no less than**

**sizeof(double), implementation defined \*/**

**Typically, sizeof(char) <= sizeof(short int) <= sizeof(int) <= sizeof(long int)**

The **signed** and **unsigned** qualifiers are applicable to the char and int types

* + unsigned numbers are always positive or zero.
  + The valid range of unsigned values is 0 to 2­­­n - 1 (where n is number of bits)
  + integers are signed by default
  + whether the **char** type is signed or not is implementation dependent
  + signed and unsigned versions of a type always require the same number of bytes

## Strings:

1. Another data type: a derivative data type of char data type. Array of char terminated by a null character ‘\0’.
2. String cannot be assigned using = like other variables.
3. Various standard C library routines exist strcpy, strcat, strlength
4. Null character is at the end of the string

## Control structures:

### Sequence structure:

Is simple, the computer executes C statements one after the other in the order in which they are written

1 #include<stdio.h>

2 Int main(void)

3 {

4 Printf(“Hello”);

5 Printf(“world!”);

6 }

### Selection structure:

C provides three types of selection structures: single selection statement, double selection statement and multiple selection statement.

#### Single selection statement: the if selection statement

**If(expression)**

**Statement;**

Example:

if(grade >= 60)

Printf(“The student passed the exam!”);

#### Double selection statement: the if else selection statement

**If(expression)**

**Statement;**

**Else**

**Statement;**

Example:

if(grade >= 60)

printf(“The student has passed the exam!”);

else

Printf(“The student has failed the exam!”);

#### Multiple selection statement:

##### Else – if

**If(expression)**

**Statement;**

**Else if(expression)**

**Statement;**

**Else if(expression)**

**Statement;**

**Else if(expression)**

**Statement;**

**Else**

**Statement;**

Example:

if(grade>=90)

printf(“A”);

else if(grade>=80)

printf(“B”);

else if(grade>=70)

pintf(“C”);

else if(grade>=60)

Printf(“D”);

else

Printf(“F”);

##### Switch

**switch(expression)**

**{**

**case const-expr: statements;**

**case const-expr: statements;**

**default: statements;**

**}**

Example:

switch(grade) {

case 'A' :

printf("Excellent!\n" );

break;

case 'B' :

printf("Good\n" );

break;

case 'C' :

printf("Well done\n" );

break;

case 'D' :

printf("You passed\n" );

break;

case 'F' :

printf("Better try again\n" );

break;

default :

printf("Invalid grade\n" );

}

### Repetition structure:

#### While

**While (expression)**

**Statements;**

Example:

Sum = 0;

While(sum <= 15)

{

Sum = sum +5;

Printf(sum);

}

#### For

**For(exp1; exp2; exp3)**

**Statements;**

Example:

For(int i=0; i<10; i++)

Printf(i);

#### Do-while

**Do**

**Statements;**

**While(expression);**

Example:

int counter = 1;

do

{

Printf(“%u ”, counter);

}

While(++counter <= 10);

## C Functions (Basics):

There are several motivations for using functions such as:

* divide and conquer approach to make writing program more manageable by dividing the program into small and simple chunks
* software reusability by using existing functions as building blocks to create new programs.

C functions consist of three parts:

* Function definition: lines 12, 13, 14, 15
* Function prototype: line 3
* Function call: line 9

1 #include <stdio.h>

2

3 int square(int y); //function prototype

4

5 int main(void) //main function begins

6 {

7 int x;

8 for(x = 1; x<=10; ++x)

9 printf(“%d ”, square(x)); //function call

10 } //main function ends

11

12 int square(int y) //function definition

13 {

14 return y\*y;

15 }

### Function definition:

Contains the function code statements, and it should not be placed within the main function or any other functions.

Example:

int square(int y)

{

return y \*y;

}

### Function prototype:

Is a synopsis of the function containing the return type, function name, and the parameters used in the function. The compiler uses function prototypes to validate function calls and check the correct use of the function. Function prototype should be written at the beginning and before calling the function.

Example:

int square(int y);

### Function call:

Is used to invoke or call the function in the main function.

Example:

Printf(“%d”, square(x));

## C standard library:

The C standard library provides a rich collection of functions for performing common mathematical calculations, string manipulations, character manipulation, input /output, and many other useful operations.

* Input and output functionns <stdio.h>: includes functions for reading, writing, formatting texts
* String functions <string.h>: includes functions for string copy, concatenate, compare ….
* Mathematical functions <math.h>: includes functions for sin, cosin, power, sqrt….
* Date and time functions <time.h>: includes functions for manipulating date and time