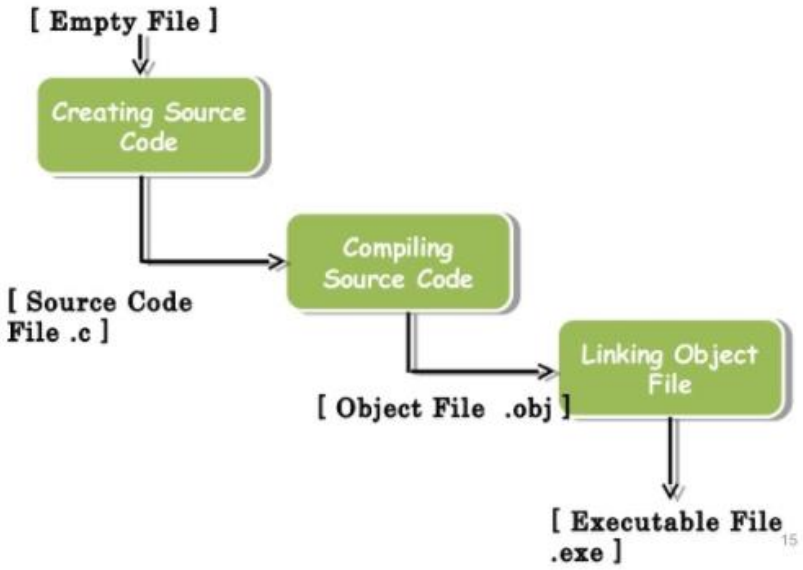
Overview of C programming language

C is a very common language, and it is compatible with different platforms such as Windows, Linux, mac OS, and many more. C is a **compiled language** - which means that in order to run it, the compiler (for example, GCC or Visual Studio) must take the code that we wrote, process it, and then create an executable file. This file can then be executed, and will do what we intended for the program to do.



Simple C program

```c
#include <stdio.h>

int main() {
  printf("Goodbye, World!");
  return 0;
}
```

Variables and Data Types

Data types

C has a few basic types:

- Integers - whole numbers either positive or negative. Defined using int, short, long, char.
- Unsigned integers - Positive whole numbers. Defined using unsigned int, unsigned short, unsigned long, unsigned char.
- Floating point numbers - real numbers (numbers with fractions). Defined using float and double.
- Structures - will be explained later, in the future practical section.
-

The different types of variables define their bounds.

A char can range only from **-128 to 127**, whereas a long can range from **-2,147,483,648 to 2,147,483,647** (long and other numeric data types may have another range on different computers, for example - from **–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 on 64-bit computer**).

Note that C **does *not* have** a Boolean type. Usually, it is defined using the following notation:

```
#include <stdio.h>

int main() {
    #define BOOL char
    #define FALSE 0
    #define TRUE 1


return 0;
}
```

Defining variables

```c
#include <stdio.h>

int main() {
    int a = 0,b = 1,c = 2,d = 3, e = 4;
    a = b - c + d * e;
    printf("%d", a); /* will print 1-2+3*4 = 11 */


return 0;
}
```

Arrays

Arrays are special variables which **can hold more than one value using the same variable**, **using an index**.

```c
int numbers[10];

/* populate the array */
numbers[0] = 10;
numbers[1] = 20;
numbers[2] = 30;
numbers[3] = 40;
numbers[4] = 50;
numbers[5] = 60;
numbers[6] = 70;

/* print the 7th number from the array, which has an index of 6 */
printf("The 7th number in the array is %d", numbers[6]);
```

Question:

Calculate the average of 3 subjects' marks such as 80, 80 and 75. And display an output in an appropriate manner.

```c
int main() {
  /* TODO: define the grades variable here */
  int average;
  int grades[3];

  grades[0] = 80;
  grades[1] = 80;
  grades[2] = 75;

  average = (grades[0] + grades[1] + grades[2]) / 3;
  printf("The average of the 3 grades is: %d", average);

  return 0;
}
```

Two dimensional arrays

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Initialization:

```c
#include <stdio.h>

int main() {
    int a[3][4] = {
        {0, 1, 2, 3} ,
        {4, 5, 6, 7} ,
        {8, 9, 10, 11}
    };


return 0;
}
```

```c
#include <stdio.h>

int main() {
    int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};


return 0;
}
```

Accessing:

```c
#include <stdio.h>

int main() {
    int val = a[2][3];


return 0;
}
```

String

```c
#include <stdio.h>

int main() {
    char name[] = "John Smith";
    /* is the same as */
    char name1[11] = "John Smith";


return 0;
}
```

The reason that we need to add one, although the string John Smith is exactly 10 characters long, is for the string termination, a special character (equal to 0) which indicates the end of the string. The end of the string is marked because the program does not know the length of the string - only the compiler knows it according to the code.

String operations

Length

```c
#include <stdio.h>

int main() {
    char * name = "Nikhil";
    printf("%d\n",strlen(name));


return 0;
}
```

String comparison

The function strncmp compares between two strings, returning the number 0 if they are equal, or a different number if they are different. The arguments are the two strings to be compared, and the maximum comparison length. There is also an unsafe version of this function called strcmp, but it is not recommended to use it. For example:

```c
char * name = "John";

if (strncmp(name, "John", 4) == 0) {
    printf("Hello, John!\n");
} else {
    printf("You are not John. Go away.\n");
}
```

String concatenations

The function 'strncat' appends **first n characters of src string string to the destination string** where n is min(n,length(src)); The arguments passed are destination string, source string, and n - maximum number of characters to be appended. For Example:

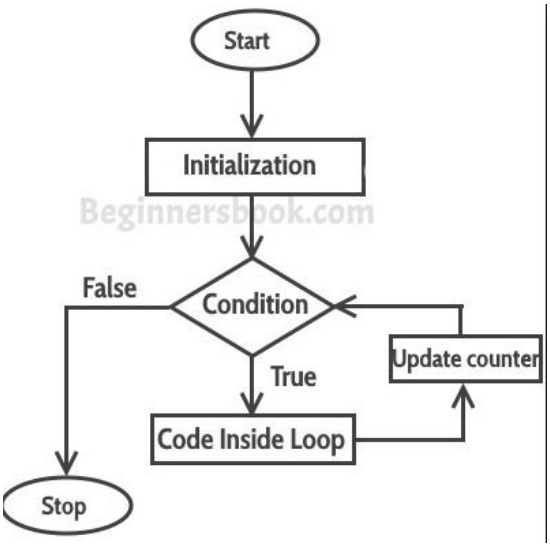
```c
#include <stdio.h>

int main() {
    char dest[20]="Hello";
    char src[20]="World";
    strncat(dest,src,3);
    printf("%s\n",dest);
    strncat(dest,src,20);
    printf("%s\n",dest);


return 0;
}
```

```
HelloWor
HelloWorWorld
```

For loop




```c
#include <stdio.h>

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


return 0;
}
```

Question

Calculate the summation of given set of integers and display the average.

```c
int main() {
    int array[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    int sum = 0;
    int i;

    for (i = 0; i < 10; i++) {
        sum += array[i];
    }

    /* sum now contains a[0] + a[1] + ... + a[9] */
    printf("Sum of the array is %d\n", sum);


return 0;
```

While loop

```c
#include <stdio.h>

int main() {
    int n = 0;
    while (n < 10) {
        n++;
    }


return 0;
}
```

Loop directives

The break directive halts a loop after ten loops, even though the while loop never finishes:

```c
#include <stdio.h>

int main() {
    int n = 0;
    while (1) {
        n++;
        printf("%d\n",n);
        if (n == 5) {
            break;
        }
    }


return 0;
```

```
1
2
3
4
5
```

The continue directive causes the printf command to be skipped.

```c
int main() {
    int n = 0;
    while (n < 10) {
        n++;

        /* check that n is odd */
        if (n % 2 == 1) {
            /* go back to the start of the while block */
            continue;
        }

        /* we reach this code only if n is even */
        printf("The number %d is even.\n", n);
    }
```

```
The number 2 is even.
The number 4 is even.
The number 6 is even.
The number 8 is even.
The number 10 is even.
```

Question

The array consists of a sequence of ten numbers. Inside the while loop, you must write two if conditions, which change the flow of the loop in the following manner (without changing the printf command):

- If the current number which is about to printed is less than 5, don't print it.
- If the current number which is about to printed is greater than 10, don't print it and stop the loop.

```c
#include <stdio.h>

int main() {
    int array[] = {1, 7, 4, 5, 9, 3, 5, 11, 6, 3, 4};
    int i = 0;

    while (i < 10) {
        if(array[i] < 5){
            i++;
            continue;
        }
```

```c
        if(array[i] > 10){
            break;
        }

        printf("%d\n", array[i]);
        i++;
    }

    return 0;
}
```

Functions

```c
/* function declaration */
int foo(int bar);

int main() {
    /* calling foo from main */
    printf("The value of foo is %d", foo(1));
}

int foo(int bar) {
    return bar + 1;
}
```

Static variables

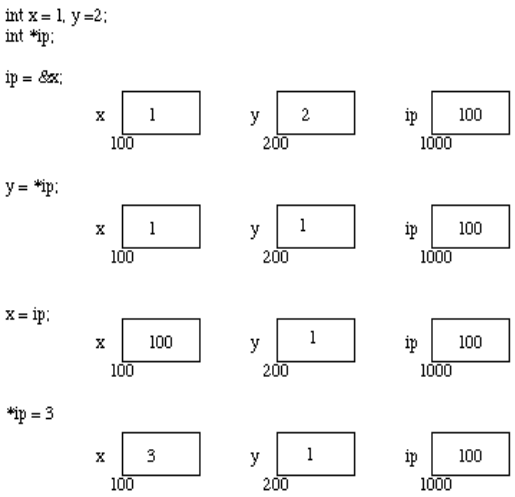
```c
#include<stdio.h>
int runner()
{
    static int count = 0;
    count++;
    return count;
}

int main()
{
    printf("%d\n", runner());
    printf("%d ", runner());
    return 0;
}
```

```
1
2
```

Pointers

A pointer is essentially a simple integer variable which holds a **memory address** that **points to a value**, instead of holding the actual value itself.



ip is declared to be a ***pointer to an integer*** and is assigned to the address of x (&x). So ip gets loaded with the value 100.

Next y gets assigned to the ***contents of*** ip. In this example ip currently ***points*** to memory location 100 -- the location of x. So y gets assigned to the values of x -- which is 1.

To assign the current value of ip to x. The value of ip at this instant is 100.

Finally we can assign a value to the contents of a pointer (*ip).

Question

Create a pointer to the local variable n called pointer_to_n, and use it to increase the value of n by one.

```c
int main() {
  int n = 10;

  int * pointer_to_n = &n;

  *pointer_to_n += 1;

  /* testing code */
  if (pointer_to_n != &n) return 1;
  if (*pointer_to_n != 11) return 1;

  printf("Done!\n");
  return 0;
}
```