CL-1004 Object Oriented Programming

LAB - 02

C++ data types, functions, struct revisited

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES
Spring 2023

While writing a program in any language, you need to use various variables to store various information. Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

You may like to store information of various data types like character, wide character, integer, floating point, double floating point, Boolean etc. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

Primitive Built-in Types

C++ offers the programmer a rich assortment of built-in as well as user defined data types. Following table lists down seven basic C++ data types −

Type	Keyword
Boolean	bool
Character	char
Integer	int
Floating point	float
Double floating point	double
Valueless	void
Wide character	wchar_t

Several of the basic types can be modified using one or more of these type modifiers

- signed
- unsigned
- short
- long

The following table shows the variable type, how much memory it takes to store the value in memory, and what is maximum and minimum value which can be stored in such type of variables.

Type	Typical Bit Width	Typical Range
char	1byte	-127 to 127 or 0 to 255

unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	2bytes	0 to 65,535
signed short int	2bytes	-32768 to 32767
long int	8bytes	-9223372036854775808 to 9223372036854775807
signed long int	8bytes	same as long int
unsigned long int	8bytes	0 to 18446744073709551615
long long int	8bytes	-(2^63) to (2^63)-1
unsigned long long int	8bytes	0 to 18,446,744,073,709,551,615
float	4bytes	
double	8bytes	
long double	12bytes	
wchar_t	2 or 4 bytes	1 wide character

The size of variables might be different from those shown in the above table, depending on the compiler and the computer you are using.

Following is the example, which will produce correct size of various data types on your computer.

```
#include <iostream>
using namespace std;
int main() {
  cout << "Size of char : " << sizeof(char) << endl;
  cout << "Size of int : " << sizeof(int) << endl;
```

```
  cout << "Size of short int : " << sizeof(short int) << endl;
  cout << "Size of long int : " << sizeof(long int) << endl;
  cout << "Size of float : " << sizeof(float) << endl;
  cout << "Size of double : " << sizeof(double) << endl;
  cout << "Size of wchar_t : " << sizeof(wchar_t) << endl;
  return 0;
}
```

This example uses **endl**, which inserts a new-line character after every line and << operator is being used to pass multiple values out to the screen. We are also using **sizeof()** operator to get size of various data types.

When the above code is compiled and executed, it produces the following result which can vary from machine to machine −

```
Size of char : 1
Size of int : 4
Size of short int : 2
Size of long int : 4
Size of float : 4
Size of double : 8
Size of wchar_t : 4
```

Following is another example:

```
#include <iostream>
#include <limits>
using namespace std;
int main() {
   std::cout << "Int Min " << std::numeric_limits<int>::min() << endl;
   std::cout << "Int Max " << std::numeric_limits<int>::max() << endl;
   std::cout << "Unsigned Int  Min " << std::numeric_limits<unsigned int>::min() << endl;
   std::cout << "Unsigned Int Max " << std::numeric_limits<unsigned int>::max() << endl;
   std::cout << "Long Int Min " << std::numeric_limits<long int>::min() << endl;
   std::cout << "Long Int Max " << std::numeric_limits<long int>::max() << endl;
   std::cout << "Unsigned Long Int Min " << std::numeric_limits<unsigned long int>::min() <<endl;
   std::cout << "Unsigned Long Int Max " << std::numeric_limits<unsigned long int>::max() << endl;
}
```

typedef Declarations

You can create a new name for an existing type using **typedef**. Following is the simple syntax to define a new type using typedef −

```
typedef type newname;
```

For example, the following tells the compiler that feet is another name for int −

```
typedef int feet;
```

Now, the following declaration is perfectly legal and creates an integer variable called distance −

```
feet distance;
```

Enumerated Types

An enumerated type declares an optional type of name and a set of zero or more identifiers that can be used as values of the type. Each enumerator is a constant whose type is the enumeration.

Creating an enumeration requires the use of the keyword **enum**. The general form of an enumeration type is −

```
enum enum-name { list of names } var-list;
```

Here, the enum-name is the enumeration's type name. The list of names is comma separated.

For example, the following code defines an enumeration of colors called colors and the variable c of type color. Finally, c is assigned the value "blue".

```
enum color { red, green, blue } c;
c = blue;
```

By default, the value of the first name is 0, the second name has the value 1, and the third has the value 2, and so on. But you can give a name, a specific value by adding an initializer. For example, in the following enumeration, **green** will have the value 5.

```
enum color { red, green = 5, blue };
```

Here, **blue** will have a value of 6 because each name will be one greater than the one that precedes it.

C++ Functions

A function is a block of code that performs a specific task.

Suppose we need to create a program to create a circle and color it. We can create two functions to solve this problem:

1. a function to draw the circle.
2. a function to color the circle.

Dividing a complex problem into smaller chunks makes our program easy to understand and reusable. There are two types of function:

1. Standard Library Functions: Predefined in C++
2. User-defined Function: Created by user

C++ User-defined Function

C++ allows the programmer to define their own function.
A user-defined function groups code to perform a specific task and that group of code is given a name (identifier).
When the function is invoked from any part of the program, it all executes the codes defined in the body of the function.

C++ Function Declaration

The syntax to declare a function is:

```
returnType functionName (parameter1, parameter2,...) {
    // function body
}
```

Here's an example of a function declaration.

```
// function declaration
void greet() {
    cout << "Hello World";
}
```

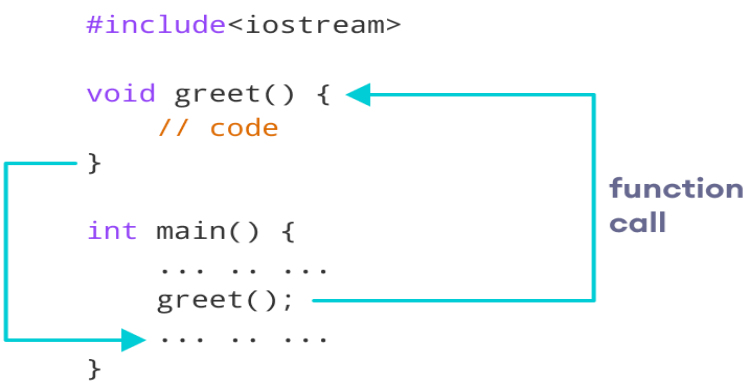
Here, the name of the function is `greet()` the return type of the function is `void` the empty parentheses mean it doesn't have any parameters the function body is written inside `{}`

Note: We will learn about `returnType` and `parameters` later in this tutorial.

Calling a Function

In the above program, we have declared a function named `greet()`. To use the `greet()` function, we need to call it.
Here's how we can call the above `greet()` function.

```
int main() {

    // calling a function
    greet();

}
```



How Function works in C++

Example 1: Display a Text

```
#include <iostream>
using namespace std;

// declaring a function
void greet() {
   cout << "Hello there!";
}

int main() {

   // calling the function
   greet();

   return 0;
}
```

Run Code

Output

```
Hello there!
```

Function Parameters

As mentioned above, a function can be declared with parameters (arguments). A parameter is a value that is passed when declaring a function.
For example, let us consider the function below:

```
void printNum(int num) {
   cout << num;
}
```

Here, the int variable num is the function parameter.
We pass a value to the function parameter while calling the function.

```
int main() {
   int n = 7;

   // calling the function
   // n is passed to the function as argument
   printNum(n);

   return 0;
}
```

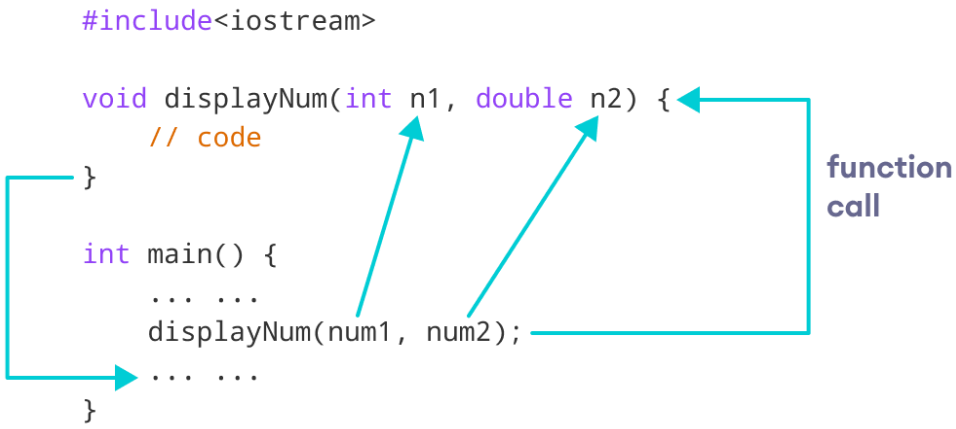
Example 2: Function with Parameters

```
// program to print a text
#include <iostream>
using namespace std;
// display a number
void displayNum(int n1, float n2) {
   cout << "The int number is " << n1;
   cout << "The double number is " << n2;
}
int main() {
    int num1 = 5;
    double num2 = 5.5;
   // calling the function
   displayNum(num1, num2);

   return 0;
}
```

Output

```
The int number is 5
The double number is 5.5
```

In the above program, we have used a function that has one int parameter and one double parameter.
We then pass num1 and num2 as arguments. These values are stored by the function parameters n1 and n2 respectively.



Note: The type of the arguments passed while calling the function must match with the corresponding parameters defined in the function declaration.

Return Statement

In the above programs, we have used void in the function declaration. For example,

```
void displayNumber() {
  // code
}
```

This means the function is not returning any value.
It's also possible to return a value from a function. For this, we need to specify the `returnType` of the function during function declaration.
Then, the `return` statement can be used to return a value from a function.
For example,

```
int add (int a, int b) {
  return (a + b);
}
```

Here, we have the data type `int` instead of `void`. This means that the function returns an `int` value.
The code `return (a + b);` returns the sum of the two parameters as the function value.
The `return` statement denotes that the function has ended. Any code after `return` inside the function is not executed.

Example 3: Add Two Numbers

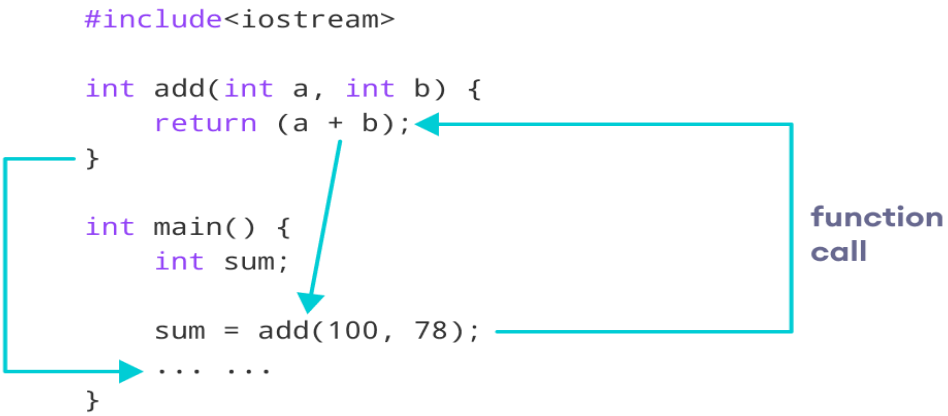
```
// program to add two numbers using a function
#include <iostream>
using namespace std;
// declaring a function
int add(int a, int b) {
```

```
    return (a + b);
}
int main() {
    int sum;
    // calling the function and storing
    // the returned value in sum
    sum = add(100, 78);
    cout << "100 + 78 = " << sum << endl;
    return 0;
}
```
Run Code

Output

```
100 + 78 = 178
```

In the above program, the `add()` function is used to find the sum of two numbers.
We pass two `int` literals `100` and `78` while calling the function.
We store the returned value of the function in the variable `sum`, and then we print it.



Notice that `sum` is a variable of `int` type. This is because the return value of `add()` is of `int` type.

Function Prototype

In C++, the code of function declaration should be before the function call. However, if we want to define a function after the function call, we need to use the function prototype. For example,

```
// function prototype
void add(int, int);

int main() {
    // calling the function before declaration.
    add(5, 3);
```

```
    return 0;
}

// function definition
void add(int a, int b) {
    cout << (a + b);
}
```

In the above code, the function prototype is:

```
void add(int, int);
```

This provides the compiler with information about the function name and its parameters. That's why we can use the code to call a function before the function has been defined.
The syntax of a function prototype is:

```
returnType functionName(dataType1, dataType2, ...);
```

Example 4: C++ Function Prototype

```
// using function definition after main() function
// function prototype is declared before main()

#include <iostream>

using namespace std;

// function prototype
int add(int, int);

int main() {
    int sum;

    // calling the function and storing
    // the returned value in sum
    sum = add(100, 78);

    cout << "100 + 78 = " << sum << endl;

    return 0;
}

// function definition
int add(int a, int b) {
    return (a + b);
}
```

Output

```
100 + 78 = 178
```

The above program is nearly identical to Example 3. The only difference is that here, the function is defined after the function call.
That's why we have used a function prototype in this example.

Benefits of Using User-Defined Functions

Functions make the code reusable. We can declare them once and use them multiple times.
Functions make the program easier as each small task is divided into a function.
Functions increase readability.

C++ Library Functions

Library functions are the built-in functions in C++ programming.
Programmers can use library functions by invoking the functions directly; they don't need to write the functions themselves.
Some common library functions in C++ are `sqrt()`, `abs()`, `isdigit()`, etc.
In order to use library functions, we usually need to include the header file in which these library functions are defined.
For instance, in order to use mathematical functions such as `sqrt()` and `abs()`, we need to include the header file `cmath`.

Example 5: C++ Program to Find the Square Root of a Number

```
#include <iostream>
#include <cmath>
using namespace std;

int main() {
   double number, squareRoot;

   number = 25.0;

   // sqrt() is a library function to calculate the square root
   squareRoot = sqrt(number);

   cout << "Square root of " << number << " = " << squareRoot;

   return 0;
}
```

Run Code

Output

```
Square root of 25 = 5
```

In this program, the `sqrt()` library function is used to calculate the square root of a number.
The function declaration of `sqrt()` is defined in the `cmath` header file. That's why we need to use the code `#include <cmath>` to use the `sqrt()` function.

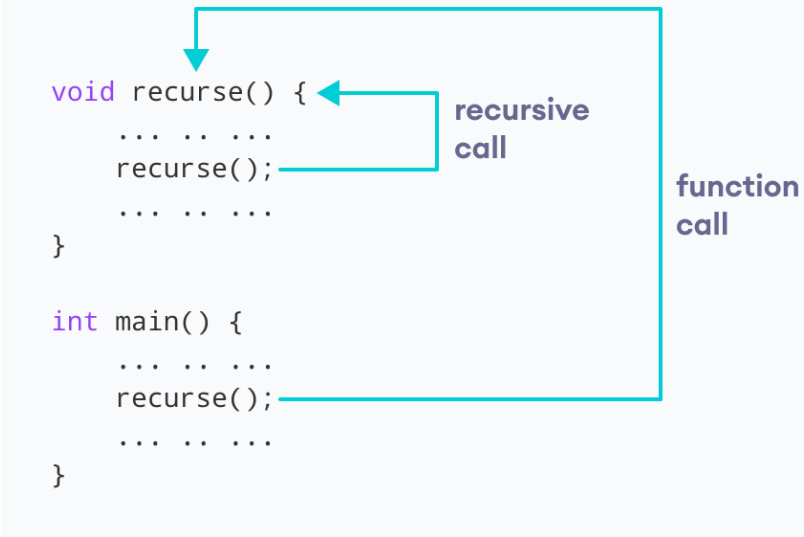
C++ Recursion

A function that calls itself is known as a recursive function. And this technique is known as recursion.

Working of Recursion in C++

```
void recurse()
{
   ... .. ...
   recurse();
   ... .. ...
}

int main()
{
   ... .. ...
   recurse();
   ... .. ...
}
```

The figure below shows how recursion works by calling itself over and over again.



The recursion continues until some condition is met.
To prevent infinite recursion, if...else statement (or similar approach) can be used where one branch makes the recursive call and the other doesn't.

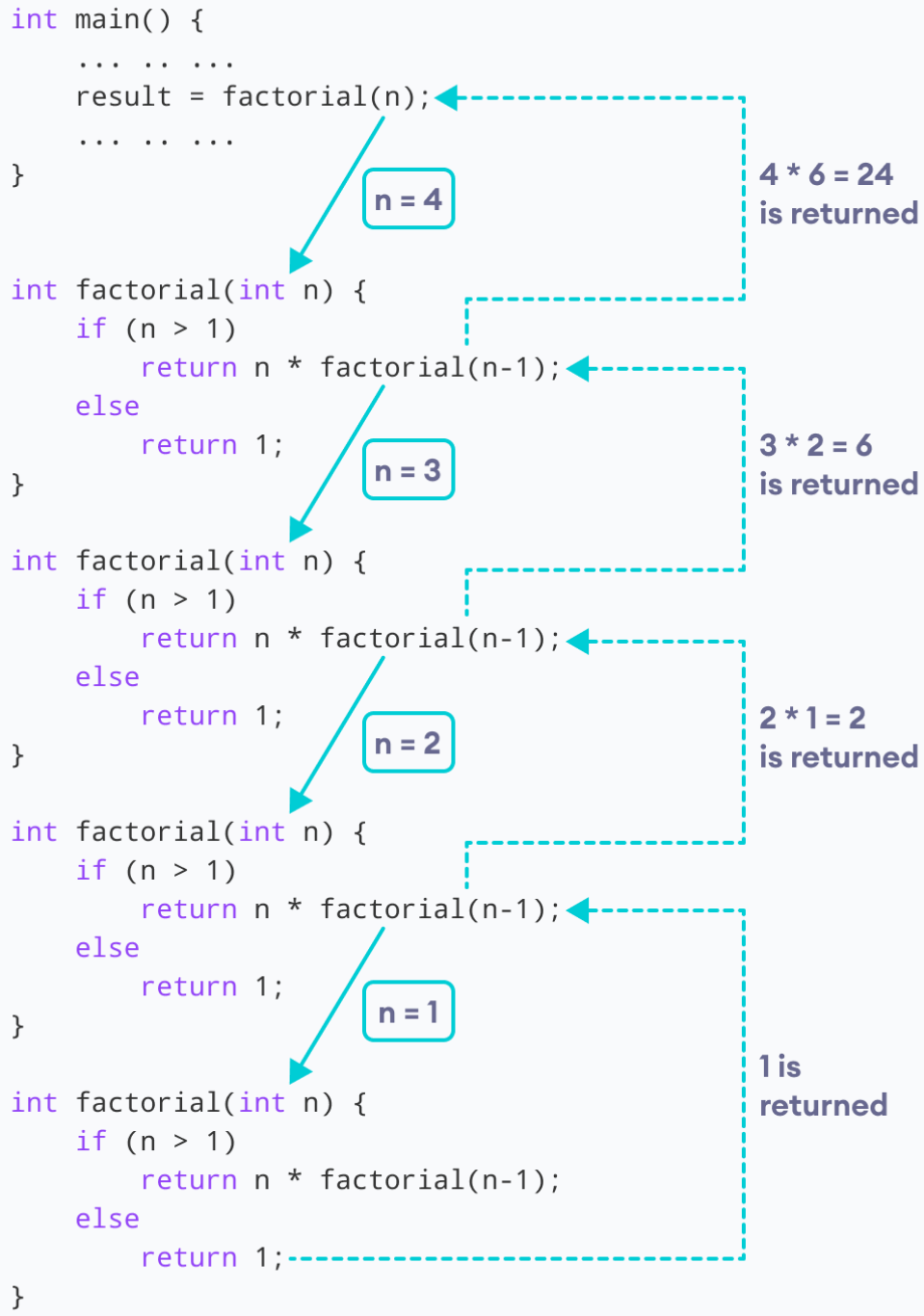
Example 1: Factorial of a Number Using Recursion

```
// Factorial of n = 1*2*3*...*n

#include <iostream>
using namespace std;

int factorial(int);

int main() {
    int n, result;

    cout << "Enter a non-negative number: ";
    cin >> n;

    result = factorial(n);
    cout << "Factorial of " << n << " = " << result;
    return 0;
}
int factorial(int n) {
    if (n > 1) {
        return n * factorial(n - 1);
    } else {
        return 1;
    }
}
```

Output

```
Enter a non-negative number: 4
Factorial of 4 = 24
```

Working of Factorial Program



As we can see, the `factorial()` function is calling itself. However, during each call, we have decreased the value of `n` by `1`. When `n` is less than `1`, the `factorial()` function ultimately returns the output.

Advantages of C++ Recursion

1. It makes our code shorter and cleaner.

2. Recursion is required in problems concerning data structures and advanced algorithms, such as Graph and Tree Traversal.

Disadvantages of C++ Recursion

1. It takes a lot of stack space compared to an iterative program.
2. It uses more processor time.
3. It can be more difficult to debug compared to an equivalent iterative program.

C++ Structures

Structure is a collection of variables of different data types under a single name. It is similar to a class in that, both holds a collection of data of different data types.

For example: You want to store some information about a person: his/her name, citizenship number and salary. You can easily create different variables name, citNo, salary to store this information separately.

However, in the future, you would want to store information about multiple people. Now, you'd need to create different variables for each information per person: name1, citNo1, salary1, name2, citNo2, salary2.

You can easily visualize how big and messy the code would look. Also, since no relation between the variables (information) would exist, it's going to be a daunting task. A better approach will be to have a collection of all related information under a single name Person, and use it for every person. Now, the code looks much cleaner, readable and efficient as well. This collection of all related information under a single name Person is a structure.

How to declare a structure in C++ programming?

The struct keyword defines a structure type followed by an identifier (name of the structure). Then inside the curly braces, you can declare one or more members (declare variables inside curly braces) of that structure. For example:

```
struct Person
{
   char name[50];
   int age;
   float salary;
};
```

Here a structure person is defined which has three members: name, age and salary.

When a structure is created, no memory is allocated. The structure definition is only the blueprint for the creating of variables. You can imagine it as a datatype. When you define an integer as below:

```
int foo;
```

The int specifies that, variable foo can hold integer element only. Similarly, structure definition only specifies that, what property a structure variable holds when it is defined.

Note: Remember to end the declaration with a semicolon (;)

How to define a structure variable?

Once you declare a structure person as above. You can define a structure variable as:

```
Person bill;
```

Here, a structure variable bill is defined which is of type structure Person. When structure variable is defined, only then the required memory is allocated by the compiler. Considering you have either 32-bit or 64-bit system, the memory of float is 4 bytes, memory of int is 4 bytes and memory of char is 1 byte. Hence, 58 bytes of memory is allocated for structure variable bill.

How to access members of a structure?

The members of structure variable is accessed using a dot (.) operator. Suppose, you want to access age of structure variable bill and assign it 50 to it. You can perform this task by using following code below:

```
bill.age = 50
```

Example: C++ Structure

```
#include <iostream>
using namespace std;

struct Person
{
   char name[50];
   int age;
   float salary;
};

int main()
{
```

```
    Person p1;

    cout << "Enter Full name: ";
    cin.get(p1.name, 50);
    cout << "Enter age: ";
    cin >> p1.age;
    cout << "Enter salary: ";
    cin >> p1.salary;

    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p1.name << endl;
    cout <<"Age: " << p1.age << endl;
    cout << "Salary: " << p1.salary;

    return 0;
}
```

Output

```
Enter Full name: Magdalena Dankova
Enter age: 27
Enter salary: 1024.4

Displaying Information.
Name: Magdalena Dankova
Age: 27
Salary: 1024.4
```

Here a structure Person is declared which has three members name, age and salary.
Inside main() function, a structure variable p1 is defined. Then, the user is asked to enter information and data entered by user is displayed.

Exercises:

1. Write a code with the help of function that asks your name, e.g., RoboAli(), first asks your name. When you Input your name, then, with your name, second, it asks you where you live. Third, when you input your location, it responded to you that it is a great place to live.

2. Write a code with help of function that tells you whether you are eligible to sit in an exam based on your attendance. E.g. you can sit in the exam if you attendance is above 80%.

3. Write a code for the MS eligibility calculator with the help of a function in this scenario; suppose a student wants to enrol in the FAST NUCES MS CYS program. Total score MCQS + Essay + Viva = 150. For eligibility, a student needs at least 75 passing marks. Write three Input functions that ask for the student's MCQS + Essay + Viva marks, and then its sum shows whether the student passes or fails the test.

4. Create a struct for airplanes. It should hold the following information:
   1. Model
   2. Role (for example, cargo, passenger, etc)
   3. Origin (country of origin)
   4. Manufacturer
   5. Status (Boolean -- In service or not)

   Create a pointer of type “airplane”, then ask the user about how many airplanes they want to store the information for. Create an array of that many
   airplanes and then store them within that array.
   Create a function that accepts the array (that you have just created) and asks
   the user to input the values for each airplane

5. Write a program to break a string into 4 substrings of equal length.
   You must do this recursively. Your function does not need to return anything,
   but it should print the substrings.
   In case the string cannot be divided into 4 equal parts, the last substring should
   carry the excess letters.
   See examples on next page

Equal division:

```
ab      cd      ef      gh
--------------------------------
Process exited after 0.1877 seconds with return value 0
Press any key to continue . . .
```

Unequal division:

```
ab      cd      ef      ghij
--------------------------------
Process exited after 0.06789 seconds with return value 0
Press any key to continue . . .
```

Hint: Use the following method from the string library:
string substr (size_t pos, size_t len)
Usage: string result = str.substr(0,4)