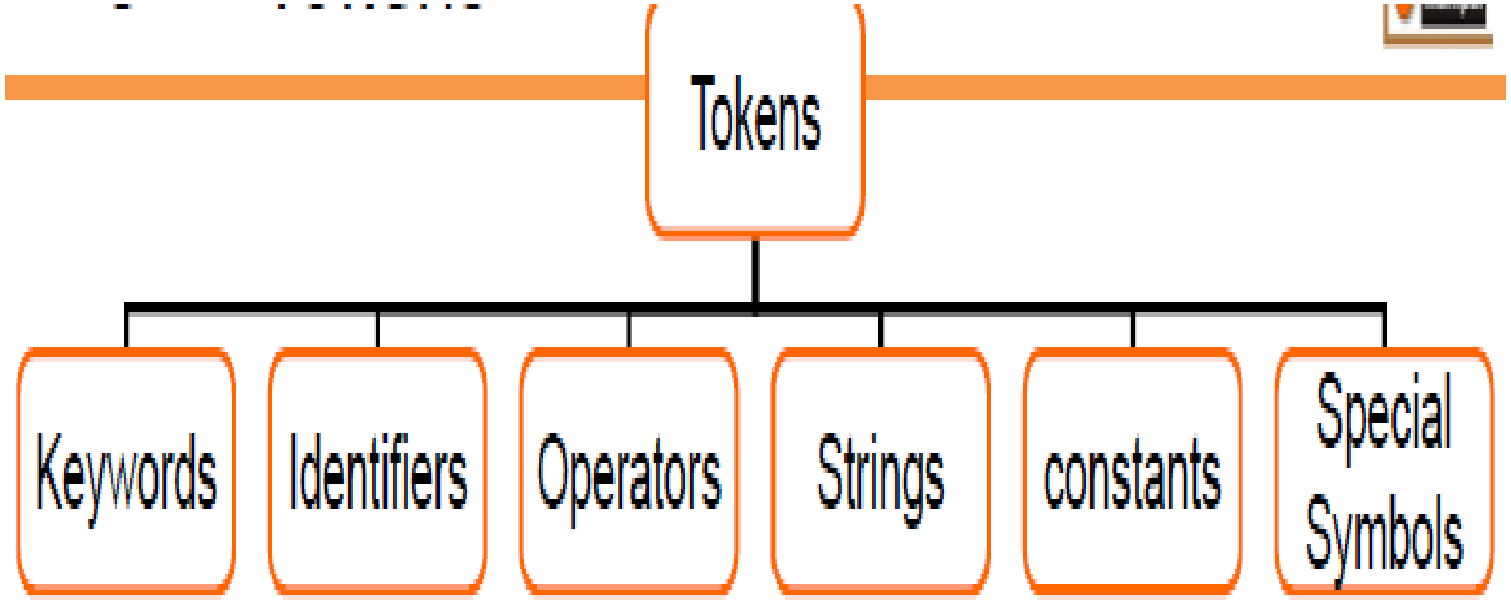
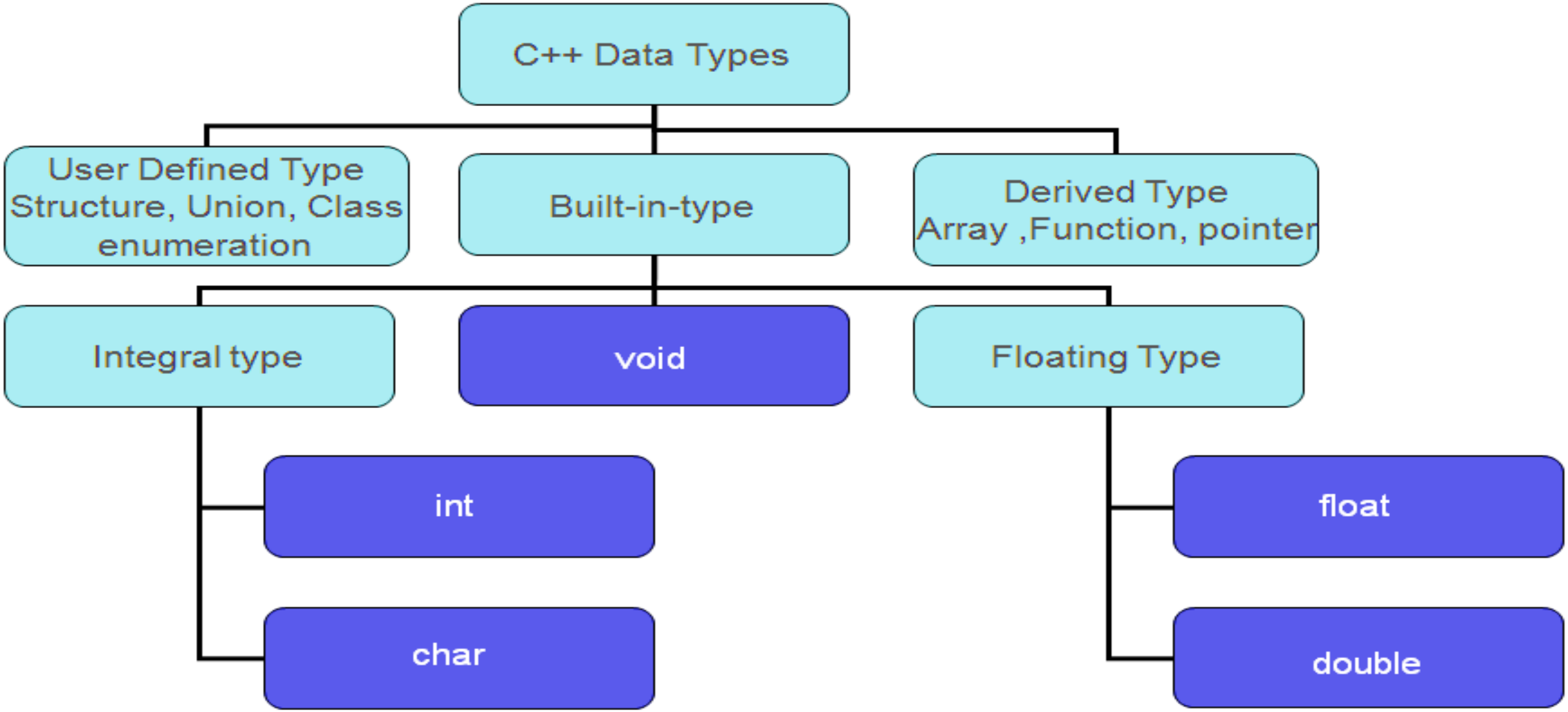
Introduction to Basics of Programming

C++ Tokens


Tokens
Keywords
Identifiers
Operators
Strings
constants
Special Symbols

C++ Tokens

- (i) Keywords words that are basically sequence of characters defined by a computer language that have one or more fixed meanings.
  - They are also called as *reserved words*.
  - Keywords cannot be changed. ex. Int, float, do-while, if, else…
- (ii) Identifiers words which have to be identified by keywords.
  - user defined names. ex. int amount, float avg,…
- (iii) Operators +, -, *, %, /, …
- (iv) Strings “Manipal”
- (v) Constants -15, 10
- (vi) Special Symbols { } (,…


C++ Data Types
User Defined Type
Structure, Union, Class
enumeration
Built-in-type
Derived Type
Array ,Function, pointer
Integral type
void
Floating Type
int
char
float
double

//program in C++

Comments

- These are single line comments.
- All lines beginning with two slash signs (//) are considered comments
- They do not have any effect on the behavior of the program
- The programmer can use them to include short explanations or observations within the source code itself.

#include <iostream>

- Lines beginning with a sign (#) are directives for the preprocessor.
- They are not regular code lines.
- directive #include<iostream> tells the preprocessor to include the iostream standard header file.
- This specified file (iostream) includes the declarations of the basic standard input-output library in C++, and it is included because its functionality is going to be used later.

main()

- The main function is the point where all C++ programs start their execution, independently of its location within the source code.
- it is essential that all C++ programs have a main function.
- The word main is followed in the code by a pair of parentheses (). That is because it is a function declaration.
- Optionally, these parentheses may enclose a list of parameters within them.
- Right after these parentheses we can find the body of the main function enclosed in braces{ }.

Simple C++ Program

```
// Display “This is my first C++ program”
// Single line comment
#include <iostream>
using namespace std;                    // preprocessor directive
main( )                                 // Entry point for program execution
{Begin
                // block of statements:
 cout << “This is my first C++ program”;
               // block of statements:
 End}
```

Simple C++ Program

```
#include<iostream>
using namespace std;
int main()
{
        cout<<"Enter Roll Number and marks of three subjects";
        int RollNo,marks1,marks2,marks3;
        float minimum = 35.0;
        cin>>marks1>>marks2>>marks3;
        avg = (marks1+marks2+marks3)/3;
        if (avg < minimum )
        cout<<RollNo<<"fail";
        else
        cout<<RollNo<<"pass";
        return 0;
        }
```

Program to read and display a number

```
#include<iostream>
using namespace std;
int main() {                        //program body begins
        int number;                  //variable declaration
        cout<<"enter number";    // user friendly info display
        cin>>number;                 // reading or input value to the variable
        cout<<"\nthe no. is\n"<<number;   //writing or output variable value
        return 0;
 } // end of program
```

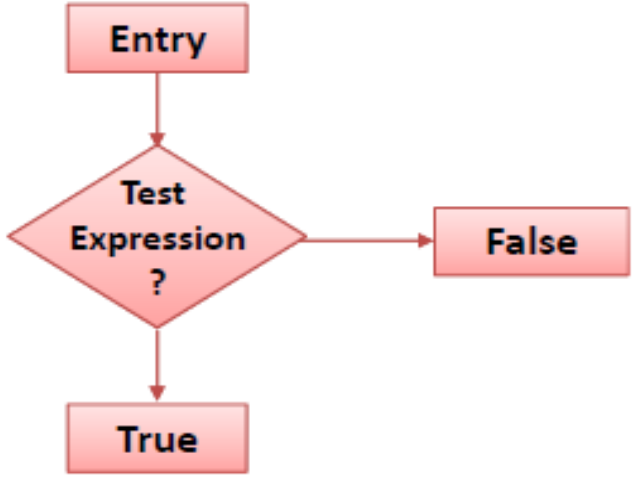
C++ **decision making** and **branching** statements

1. if Statement

2. switch statement

if statement

- Used to control the ***flow of execution*** of statements.
- It's a ***Two-way decision statement***, used in conjunction with an expression.
  - It takes the form: **if(test expression)**
- It allows to ***evaluate the*** (en, depending on the value; true or false, it trans rticular statement.
- i.e. **Two-way branching**



Different forms of **if** statement

1. Simple if statement.
2. if…else statement.
3. Nested if…else statement.
4. else if ladder.

Simple **if** Statement

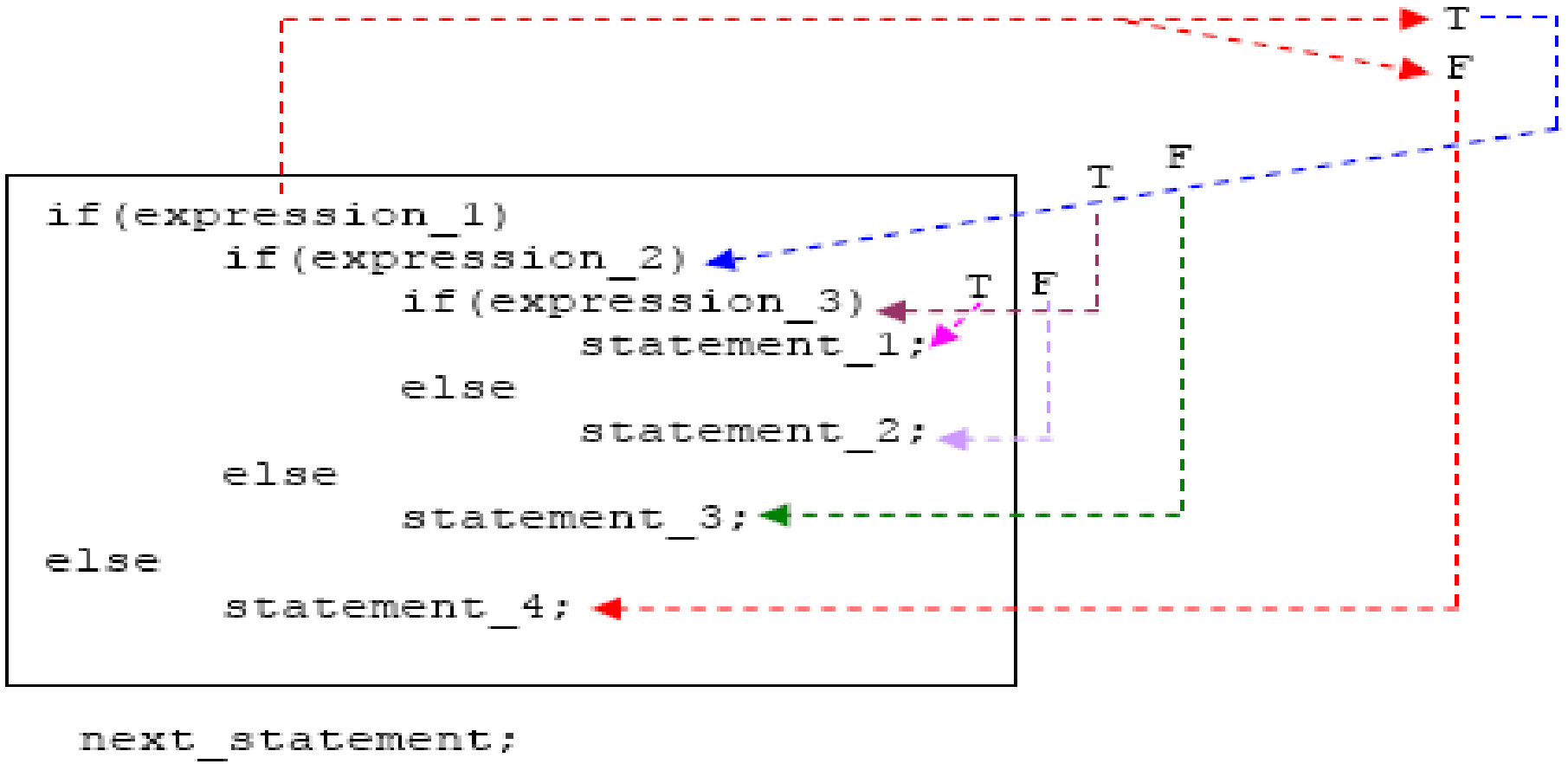
General form of the simplest if statement:

```
if (test Expression)
        {
                statement-block;
        }
statement_x;
```

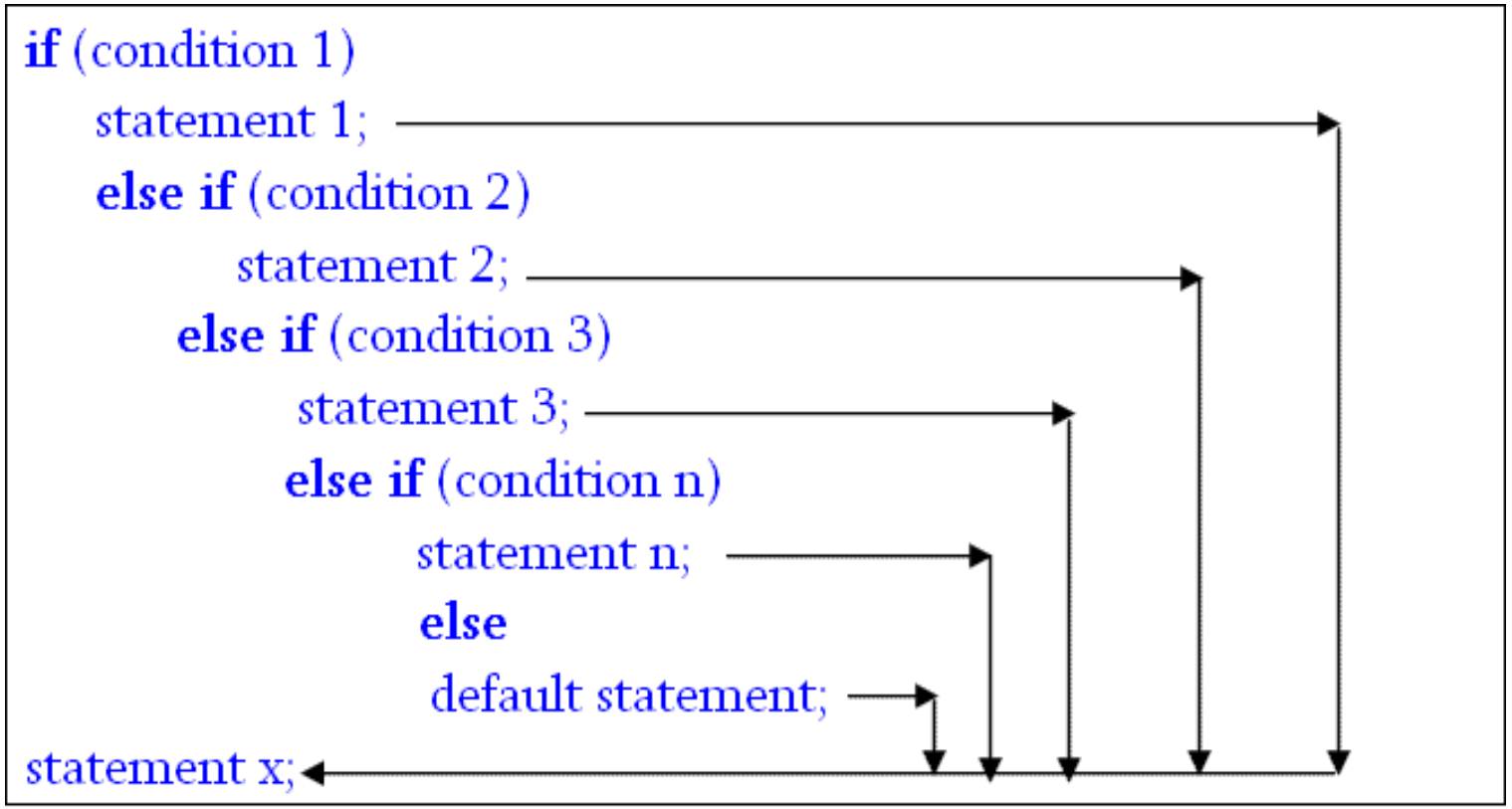
If else statement

```
if(expression)
        statement_1;
else
        statement_2;
next_statement;
```

Nesting of if-else Statements



else if Ladder



switch Statement

- Switch is multiple–branching statement - based on a condition, the control is transferred to one of the many possible points.
- The most flexible control statement in selection structure of program control.
- Enables the program to execute different statements based on an expression that can have more than two values. Also called multiple choice statements.

General form:

```
switch(expression)
{
 case value_1 : statement(s);
                break;
 case value_2 : statement(s);
                break; ...
 case value_n : statement(s);
                break;
 default : statement(s);
}
 next_statement;
```

switch- example

```
index=mark/10;
switch (index)
{
case 10:
case  9:
case  8:  grade='A';
           break;
case  7:
case  6:
          grade='B';
          break;
case  5:
          grade='C'
          break;
case 4:
          grade='D'
          break;
default:  grade='F';
           break;
   }  cout<<grade;
```

Decision Making and Looping Control Structures

- Iterative (repetitive) control structures are used to repeat certain statements for a specified number of times.
- The statements are executed if the condition is true
- These kind of control structures are also called as loop control structures
- Three kinds of loop control structures:
  - while
  - do while
  - for

While statement

Basic format:

```
while (test condition)
{
        body of the loop
}
```

- **Entry controlled** loop statement
- Test condition is evaluated & if it is true, then body of the loop is executed.
- After execution, the test condition is again evaluated & if it is true, the body is executed again.
- This is repeated until the test condition becomes false, & control transferred out of the loop.
- **Body of loop may not be executed if the condition is false at the very first attempt**.

Do - While statement

General form:

```
do
        {
                body of the loop
        }
while (test condition);
```

for statement

The general form:

```
for (initialization; test condition; increment)
        {
                Body of the loop
        }
Next statement;
```

Nesting of **for** loop

One for statement within another for statement.

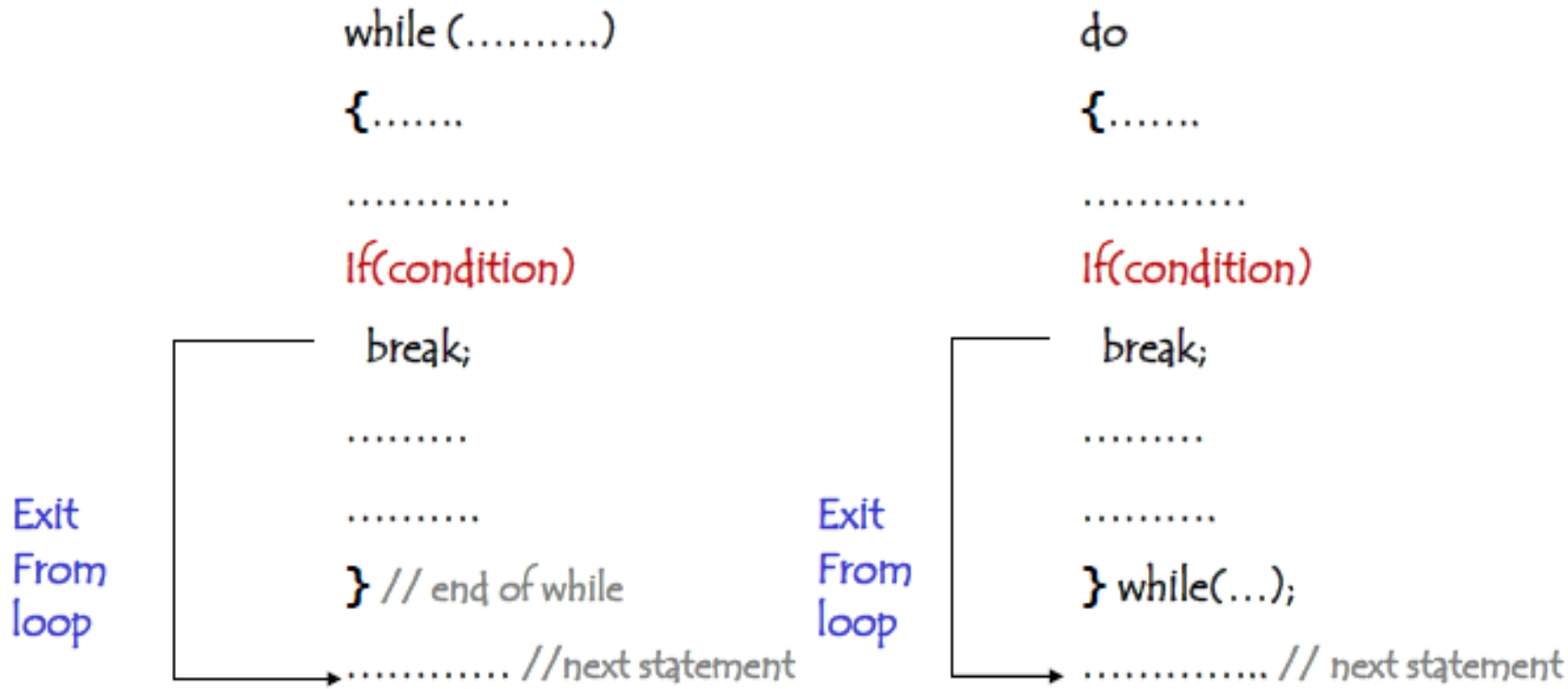
```
for (i=0; i< m; ++i)
        {.....
                ....
                for (j=0; j < n;++j)
                {.....
                        .....
                } // end of inner 'for' statement
        }// end of outer 'for' statement
```

Jumping out of a loop

- An early exit from a loop can be accomplished by using the break statement.
- When the break statement is encountered inside a loop, the loop is immediately exited & the program continues with the statement immediately following the loop.
- When the loops are nested , the break would only exit from the loop containing it.

i.e., the break will exit only a single loop.

Exiting a loop with **break** statement

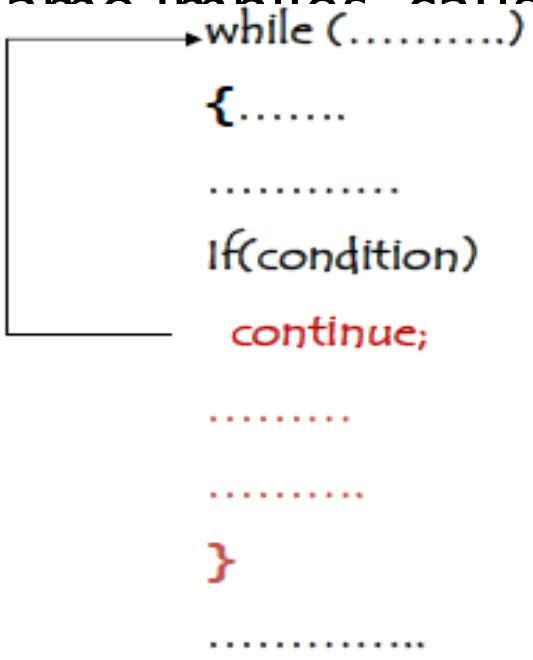


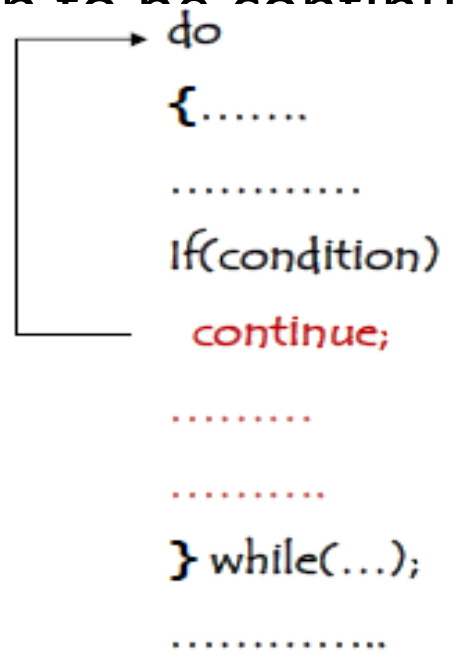
Skipping a part of loop

- Skip a part of the body of the loop under certain conditions

  Using continue statement.

- As the name implies, causes the loop to be continued with next iterati





Tutorial

- Check if a given number is prime or not
- Factorial of given 10 numbers(do not use arrays)
- Print all odd numbers between m and n
- Menu driven program to sum all elements entered up to -1
- Find sin(x) using series.
- Find cos(x) using series
- Find e^x using series
- Print triangle in the following form using loops until n.

Ex. If n=6

1

2 3

4 5 6

C++ OOP

- C++ is an object-oriented programming language which gives a clear structure to programs and allows code to be reused, lowering development costs.
- OOP stands for Object-Oriented Programming.
  - object-oriented programming is about creating objects that contain both data and functions.
- C++ is portable and can be used to develop applications that can be adapted to multiple platforms.
- C++ is fun and easy to learn.

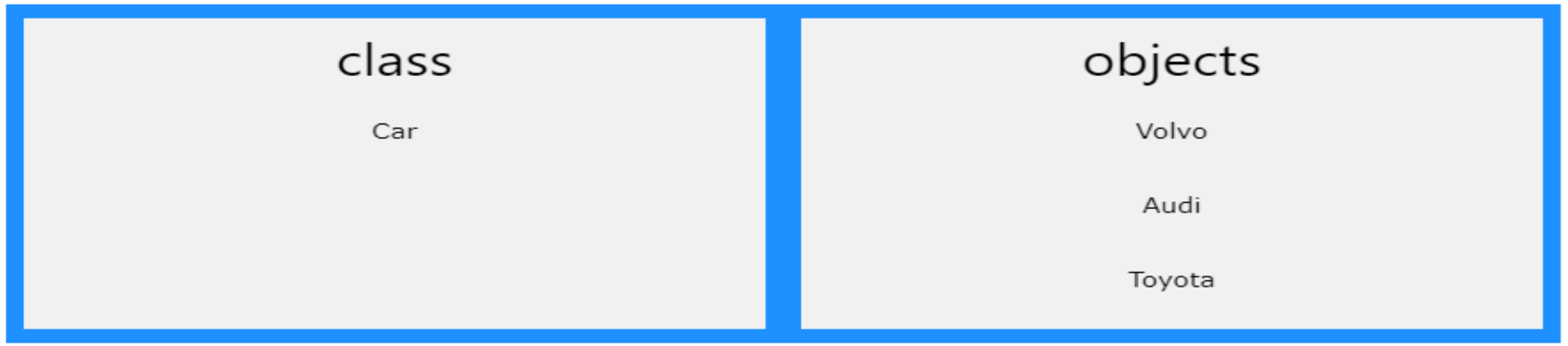
- As C++ is close to C# and Java, it makes it easy for programmers to switch to C++ or vice versa

What are Classes and Objects?

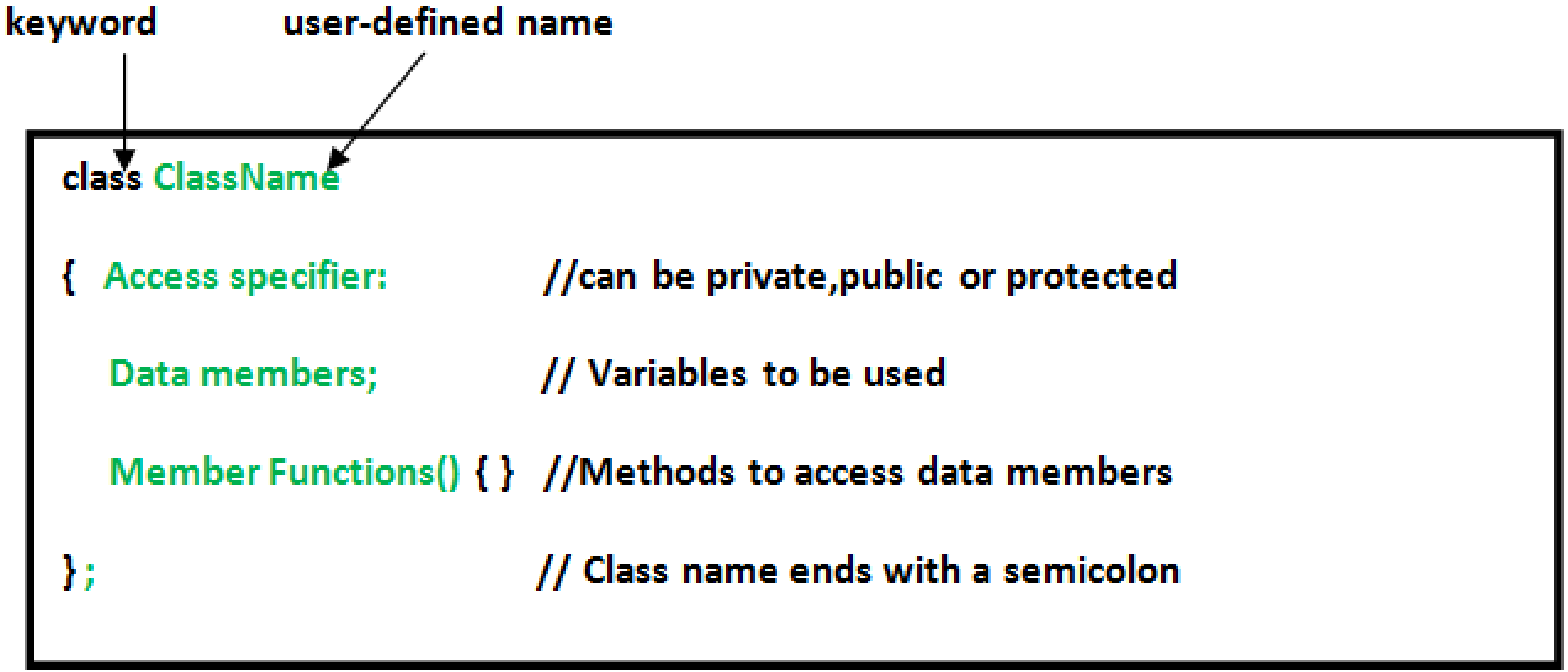
- Classe
  progr

Look at the following illustration to see the difference between class and objects:

Another example:



Syntax:


keyword
user-defined name
class ClassName
{ Access specifier: //can be private,public or protected
Data members; // Variables to be used
Member Functions() { } //Methods to access data members
}; // Class name ends with a semicolon

Create a Class

To create a class, use the `class` keyword:

Create a class called "MyClass":

```
class MyClass {        // The class
  public:              // Access specifier
    int myNum;         // Attribute (int variable)
    string myString;   // Attribute (string variable)
};
```

- The class keyword is used to create a class called MyClass.
- The public keyword is an **access specifier**, which specifies that members (attributes and methods) of the class are accessible from outside the class. (will learn more about access specifiers later.)
- Inside the class, there is an integer variable myNum and a string variable myString. When variables are declared within a class, they are called **attributes**.
- At last, end the class definition with a semicolon ;.

Create an Object

- In C++, an object is created from a class.
  - We have already created the class named MyClass, so now we can use this to create objects.

- To create an object ,
  - specify the class name, followed by the object name.
  - Class_name object_name;
- To access the class attributes (attribute_1),
  - use the dot syntax (.) on the object:
  - Object_name.attribute_1

Create an object called "myObj" and access the attributes(myNum and myString):

```
class MyClass {        // The class
  public:              // Access specifier
    int myNum;         // Attribute (int variable)
    string myString;   // Attribute (string variable)
};

int main() {
  MyClass myObj;  // Create an object of MyClass

  // Access attributes and set values
  myObj.myNum = 15;
  myObj.myString = "Some text";

  // Print attribute values
  cout << myObj.myNum << "\n";
  cout << myObj.myString;
  return 0;
}
```

Multiple Objects

```
// Create a Car class with some attributes
class Car {
  public:
    string brand;
    string model;
    int year;
};

int main() {
  // Create an object of Car
  Car carObj1;
  carObj1.brand = "BMW";
  carObj1.model = "X5";
  carObj1.year = 1999;

  // Create another object of Car
  Car carObj2;
  carObj2.brand = "Ford";
  carObj2.model = "Mustang";
  carObj2.year = 1969;

  // Print attribute values
  cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year << "\n";
  cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year << "\n";
  return 0;
}
```

C++ Access Specifiers

- The public keyword is an access specifier.
- Access specifiers define how the members (attributes and methods) of a class can be accessed.
- In the Last example ,
  - the members are public - which means that they can be accessed and modified from outside the code.
- In C++, there are three access specifiers:
  - public - members are accessible from outside the class
  - private - members cannot be accessed (or viewed) from outside the class
  - protected - members cannot be accessed from outside the class, however, they can be accessed in inherited classes.

```
class MyClass {
  public:    // Public access specifier
    int x;   // Public attribute
  private:   // Private access specifier
    int y;   // Private attribute
};

int main() {
  MyClass myObj;
  myObj.x = 25;  // Allowed (public)
  myObj.y = 50;  // Not allowed (private)
  return 0;
}
```

```
error: y is private
```

Class Methods

- Methods are **functions** that belongs to the class.
- There are two ways to define functions that belongs to a class:
  - Inside class definition
  - Outside class definition

Note: You access methods just like you access attributes; by creating an object of the class and using the dot syntax (.):

Inside Example

- ```
  class MyClass {        // The class
    public:              // Access specifier
      void myMethod() {  // Method/function defined inside the class
        cout << "Hello World!";
      }
  };

  int main() {
    MyClass myObj;     // Create an object of MyClass
    myObj.myMethod();  // Call the method
    return 0;
  }
  ```
  ```

```
class MyClass {          // The class
  public:                // Access specifier
    void myMethod() {  // Method/function defined inside the class
      cout << "Hello World!";
    }
};

int main() {
  MyClass myObj;     // Create an object of MyClass
  myObj.myMethod();  // Call the method
  return 0;
}
```

Outside Example

- To define a function outside the class definition, you have to declare it inside the class and then define it outside of the class.
- This is do[illegible] llowed the scope resolutior[illegible] e function:

```
class MyClass {        // The class
  public:              // Access specifier
    void myMethod();   // Method/function declaration
};

// Method/function definition outside the class
void MyClass::myMethod() {
  cout << "Hello World!";
}

int main() {
  MyClass myObj;     // Create an object of MyClass
  myObj.myMethod();  // Call the method
  return 0;
}
```

Asymptotic notations

- Asymptotic notations are the **mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.**
  - used to describe the running time of an algorithm - how much time an algorithm takes with a given input, n.

- There are mainly three asymptotic notations:
  - Big-O notation: describes the worst-case running time of a program.
  - Omega notation: describes the best running time of a program
  - Theta notation: counting the number of iterations the algorithm *always* takes with an input of n.

- **Algorithmic Common Runtimes**
- The common algorithmic runtimes from fastest to slowest are:
  - constant: Θ(1)
  - logarithmic: Θ(log N)
  - linear: Θ(N)
  - polynomial: Θ(N^2)
  - exponential: Θ(2^N)
  - factorial: Θ(N!)

Practice Questions:

- Program to find sum and difference of 2 numbers defined in a class.(Use sum and difference as class methods. Define add method inside the class and difference outside the class.

- Reference

https://www.w3schools.com/cpp