Introduction to C++

Fifth.School.on.LHC.Physics

Taimoor Khurshid
Exp High Energy Physics Group
National Center for Physics

Introduction

- C++ is a statically typed, compiled, general purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming.

- C++ is regarded as a middle-level language, as it comprises a combination of both high-level and low-level language features.

- C++ was developed by Bjarne Stroustrup starting in 1979 at Bell Labs in Murray Hill, New Jersey as an enhancement to the C language and originally named C with Classes but later it was renamed C++ in 1983.

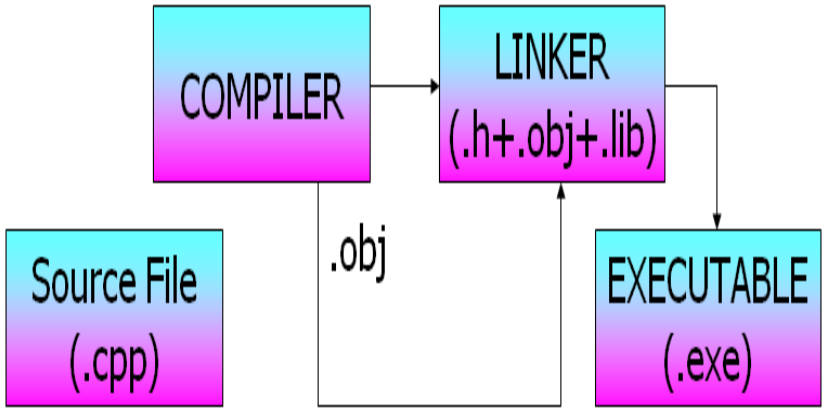
- C++ is a superset of C, and that virtually any legal C program is a legal C++ program.

C++ Program Structure

Let us look at a simple code that would print the words Hello World.

```
#include <iostream>
using namespace std;
// main() is where program execution begins.
int main()
{
   cout << "Hello World"; // prints Hello
World
return 0;
}
```

```
// This is a comment
/* C++ comments can  also
   span multiple lines
*/
```



C++ built-in data Types

- Every programming language consists of certain data types.
  C++ data types are :

1- bool (1B)
2- char (1B)
3- short int (2B)
4- int (4B)
5- long int (8B)
6- float (4B)
7- double (8B)
8- long double(16B)

Modifier Types

- *Signed*
- *unsigned*
- *long*
- *short*

- We can check these figures by the built-in sizeof() operator.

Note:

- The modifiers signed, unsigned, long, and short can be applied to integer base types.
- In addition, signed and unsigned can be applied to char.
- long can be applied to double as well.
- None of these modifiers are applicable to bool type

C++ Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.

Arithmetic Operators
(+, -, \, *, ++, --)

Relational Operators
(==, !=, >, <, >=, <=)

Logical Operators
(&&, ||, !)

Bitwise Operators
(&, |, ^, ~, <<, >>)

Assignment Operators
(=, +=, -=, *=, /=, %=, <<=, >>=, &=, ^=, |=)

Misc Operators
(sizeof, cast, conditional(?:) etc.)

C++ Loop Types

C++ programming language provides following types of loop to handle looping requirements

while loop:

- Repeats a statement or group of statements until a given condition is true.
- It tests the condition before executing the loop body.

for loop:

- Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.

do...while loop:

- Like a while statement
- Except that it tests the condition at the end of the loop body

nested loops:

- You can use one or more loop inside any another like:
  - While loop
  - for loop
  - do..while loop

C++ Decision Making

C++ provides following types of decision making statements.

If statement:

- An if statement consists of a boolean expression followed by one or more statements.

If…else statement :

- An if statement can be followed by an optional else statement, which executes when the boolean expression is false.

Switch statement:

- A switch statement allows a variable to be tested for equality against a list of values.

nested if statement:

- You can use one if or else if statement inside another if or else if statement(s).

nested switch statement:

- You can use one swicth statement inside another switch statement(s).

C++ functions

Syntax:

```
return_type function_name ( parameter list )
 {
   body of the function
 }
```

Functions have three main parts in the program:

1-Function Declaration
2-Function Definition(function declarator + body)
3-Function Call

Few Examples:

- **void function1 (int a , int b);**
- **int function2 (void)**
- **void function3 (void)**
- **int function4 (int a, int b);**

Some functions may not have a return value.
void can be used to indicate that "*the function does not return a value*".

Some functions may not have formal parameters.
void can be used to indicate that "*the function does not take any formal parameters*".

Function Example:

C++ arrays

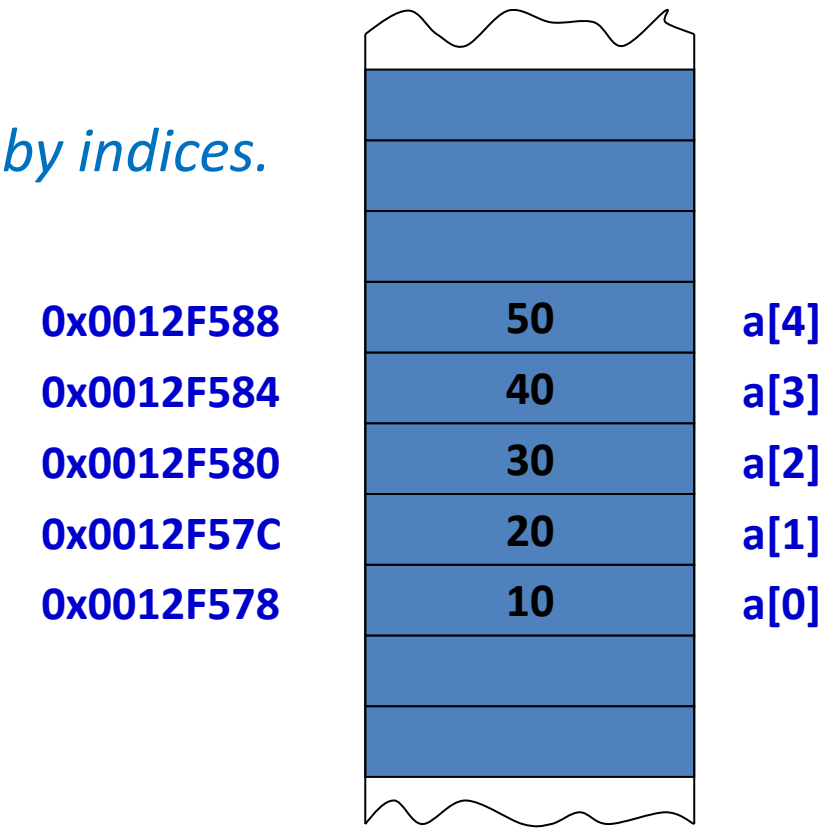
Syntax: **Type array_name[size]**

Array Example:

- *Allocates memory for size variable*
- *Index of first element is 0*
- *Index of last element is size-1*
- *Size must be a constant*
- *Individual elements of array accessed by indices.*
- *Easier to use in loops.*

int a[5] = {10, 20, 30, 40, 50};

a[0] is 10
a[1] is 20
a[2] is 30
a[3] is 40
a[4] is 50



C++ Strings

Strings Example:

- Strings are the arrays of type character.
- Last character in any string must be the null character ('\0').
- So we have to specify a string size equal to the desired data size+1.
- A character array/string can be initialized by a string literal such as "TSLP" or individual character constants as 'F', 'S', 'L', 'P'.

Please note that 'a' and "a" are different

Example:

char word[6] = "fruit"; //word has size 6

OR

char list[6] = {'f','r','u','i','t', '\0'};

Null String "" "" **contains null terminator.**

C++ pointers

- Most of C++'s power is derived from pointers.
- They allow different sections of code to share information easily.
- Pointers are variables that contain as their values, "***addresses of other variables***".

OR

- **A pointer is a variable that holds a memory address.**
- Pointers are declared using an asterisk '*'
- An integer pointer is declared as

```
int *ptr;
```

meaning that ptr is a pointer, pointing to an integer value.

Explanation: Pointers Example:

C++ structures

- In Arrays one must use homogeneous data types. An array can only hold multiple items of the same type.
- Structure is a group of different predefined data types that must be declared globally

Syntax:

```
struct structure name/tag
    {
      // structure members reside here

    } structure_variable;
```

Structures Example:

```
struct STUDENT_TYPE
{
      char name[20], street[20], city[20], state[30];
      int zipcode;
      int age;
      double IDnum;
      double grade;
};
```

Note semicolon!!!

STUDENT_TYPE is called the **structure tag**
It is your brand new data type, like int, double or char.

name, street, city, state, zipcode, age, IDnum, and **grade** are **structure members.**

C++ Classes

- A class is an entity(sub-system) that contains some information in the form of data and functions.
- It has three types of access specifiers or modifiers for maintaining access privileges to it's members.
  - public (accessible to everyone)
  - private (only accessible to class itself)
  - protected (only accessible to children classes using inheritance)
- By default the members of every class are private unless specified explicitly, whereas the members of any structure are public.
- Class is an abstract construct; we can't directly interact with it unless:
  - we create its object(s)

C++ Classes

Constructor:

- A class constructor is a function of a class which is the very first function to be executed when it's object is instantiated (created).
- It is a special type of function that gets called automatically when any object of that class is created.
- Constructor always has the same name as that of it's class.
- A class data is not loaded into the memory unless it's object is made.
- All the normal data of a class is copied for it's newly created object(except for it's functions).This leads for every object to maintain it's specific state.

Destructor:

A destructor is a function of a class which is the very last function to execute just before an object is going out of the memory.

C++ Classes

Member Functions:

The functions residing in the class are known as member functions of that class.

- private data member of a class can't be directly accessed by the object,
  - Use member functions which are public in their access level

One should always make two types of functions.

1. Accessors (get(),for read-only purpose)
2. Mutator (set(),for write mode)

Member functions are defined outside the class body provided that:

1. Declared inside the class.
2. While defining, it should be like following in declarator
   - return_type class_name scope_resolution_operator function_name function_paramters

e,g. void MyClass :: func1(int a) {….}

C++ Classes

Example Program:

```
class Test
{  private:
       int somedata;
   public:
       Test() { somedata = 0; cout <<"Object Creation via Constructor\n";}
       ~Test() { cout << "Object Destruction via Destructor\n";}
       void setData(int d)
       { somedata = d; }
       void getData()
       { cout <<"Data is "<<somedata << endl; }
 };
main()
{  Test obj1,obj2;
   obj1.setData(1066);
   obj2.setData(1776);
   obj1.getData();
   obj2.getData(); }
```

C++ Classes

Static Class Data and Static Functions:

- Static data is that data for which the state of all the objects remains same at a particular instant of time.

- This data is meant for the whole class equally rather than for individual objects.

- For all static data, no copy is generated for each newly created object rather it is used from a common (shared) location in memory for all the objects.

- To show the difference, it is always initialized outside the class body and with the reference of the class name (plus the scope resolution operator ::)

C++ Classes

Static Class Data and Static Functions:

- If value of a static data is forcefully changed by any object, it also gets changed for every other object of that class at that very moment.

- Static function can only operate on the static data, but static data can be used in non-static functions.

- Static members of a class can be called either through the class name or through the object (unlike the non-static members).

C++ Classes

Static Class Data and Static Functions:

```
class gamma
{
   private: static int total;int id;
   public : gamma() { total++;id = total;}
            ~gamma() { total--;  cout << "\nDestroyoing ID # "<<id;}
  static void showTotal() {cout << "\nTotal is "<<total;}
  void showId() {cout << "ID Number is"<< id;}
};

int gamma::total=0;  // static member initialization

main()
{
     gamma :: showTotal();
     gamma g1; gamma :: showTotal();
     gamma g2, g3; gamma :: showTotal();
     g1.showId(); g2.showId(); g3.showId();
     cout << "\nEnd Of Program";
}
```

C++ Classes

Array of Objects,Pointer to Objects:

- Arrays can also be of user-defined data types.
  e.g. Test t[10];

- We can get pointer to user-defined data types too.
  E.g.Test* t = new Test();

By doing this we get a nameless object whose reference is maintained by 't'.

- For user-defined pointers constructor is not called unless we allocate memory for the object dynamically.

- Pointer to a class can be placed in the same class as it's member.This technique is very useful in the complex scenarios.

- We have to use '->' operator instead of '.' to access class members or structure members through a pointer.

Good Luck

Back up slides

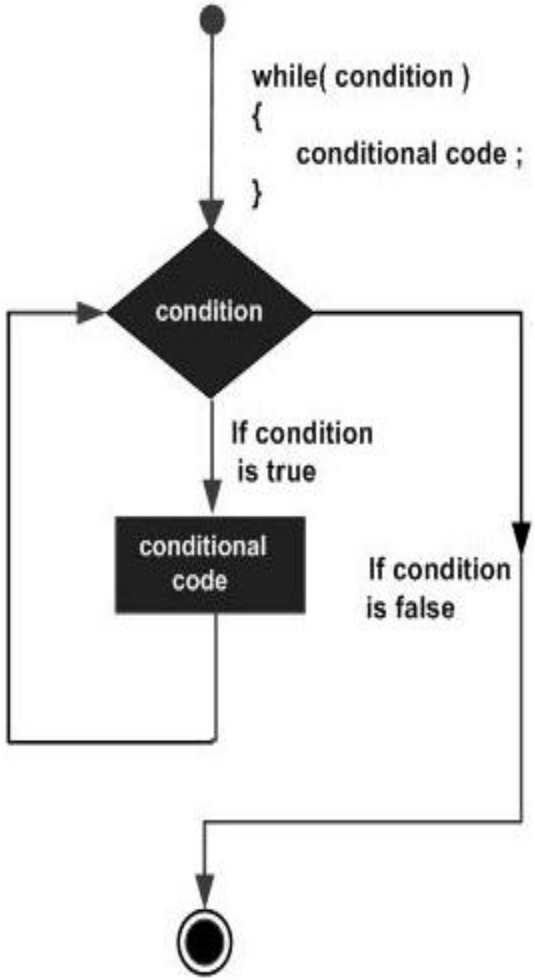
Back

While loop

Syntax:

```
while(condition)
{
  statement(s);
}
```

Flow Diagram



Example program:

```
#include <iostream>
using namespace std;
int main ()
{
// Local variable declaration:
int a = 10;
// while loop execution
while( a < 20 )
    { cout << "value of a: " << a << endl;
     a++;
    }
return 0;
}
```

Out put:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

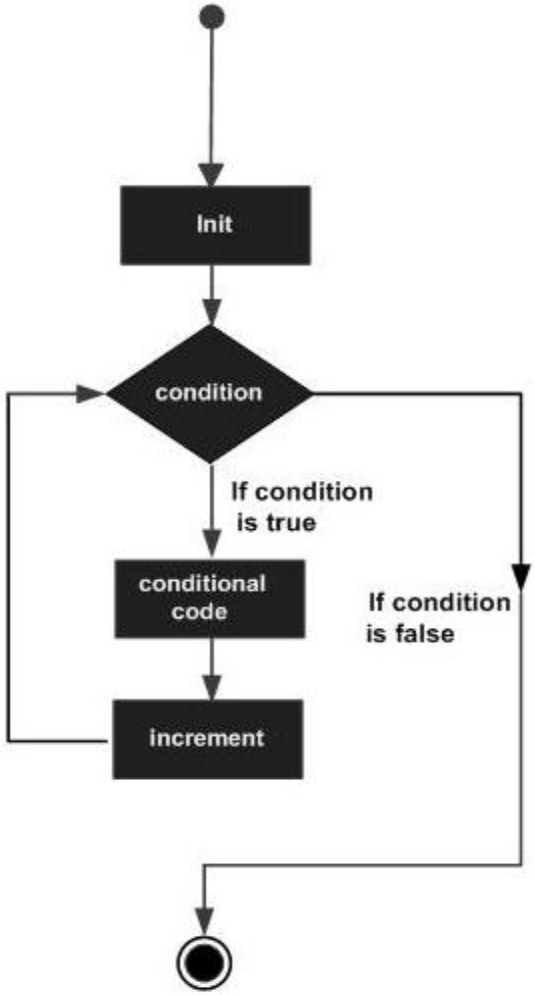
Back

For loop

Syntax:

```
for ( init; condition; increment )
  {
    statement(s);
  }
```

Flow Diagram



Example program:

```
#include <iostream>
using namespace std;
int main ()
{
 // for loop execution
for( int a = 10; a < 20; a = a + 1 )
  {
    cout << "value of a: " << a << endl;
  }
 return 0;
}
```

Out put:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

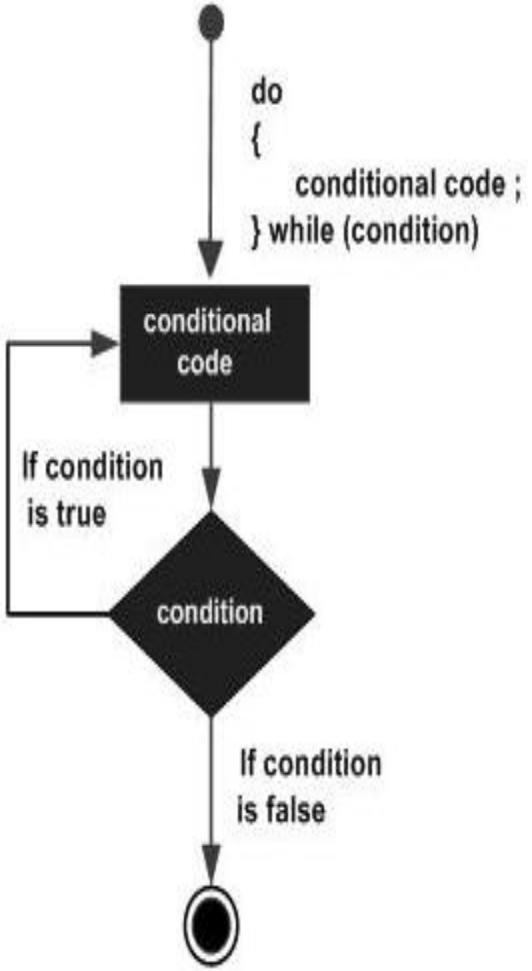
Back

Do…while loop

Syntax:

```
do
  {
    statement(s);
  } (condition);
```

Flow Diagram



Example program:

```
#include <iostream>
using namespace std;
int main ()
 {
// Local variable declaration:
int a = 10;
// do loop execution
do
{
 cout << "value of a: " << a << endl;
 a = a + 1;
}while( a < 20 );
return 0;
}
```

Out put:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Back

Nested loops

Syntax nested for:

```
for ( init; condition; increment )
 {
   for ( init; condition; increment )
    {
      statement(s);
    }
   statement(s);
 // you can put more statetments.
 }
```

Syntax nested while:

```
while(condition)
 {
   while(condition)
    {
      statement(s);
    }
   statement(s);
 // you can put more statetments.
 }
```

Syntax nested do while:

```
do
 {
   statement(s);
   // you can put more
    statetments.
   do
    {
      statement(s);
    }while( condition );
 }while( condition );
```

Example program for nested loop:

```
#include <iostream>
using namespace std;
int main ()
{ int i, j;
 for(i=2; i<100; i++)
{
for(j=2; j <= (i/j); j++)
 if(!(i%j)) break;// if factor found, not prime
if(j > (i/j)) cout << i << " is prime\n";
 }
return 0;
}
```

Out put:

```
2 is prime
 3 is prime
 5 is prime
7 is prime
11 is prime
13 is prime
.
.
97 is prime
```

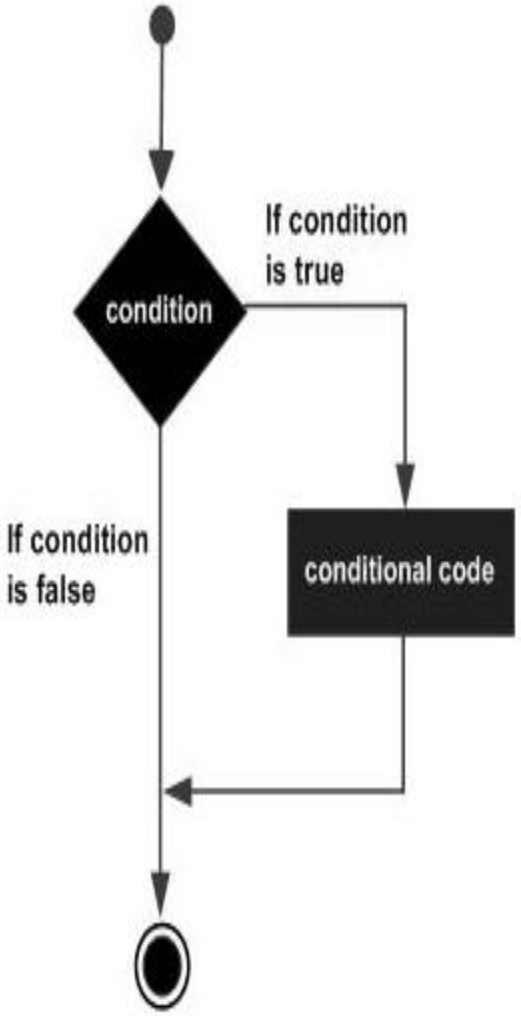
Back

If statement

Syntax:

```
If ( boolean_expression )
  {
    statement(s); //will execute if boolean is true
  }
```

Flow Diagram



Example program:

```
#include <iostream>
using namespace std;
int main ()
{
// local variable declaration:
int a = 10;
// check the boolean condition
if( a < 20 )
  {
    // if condition is true then print the following
    cout << "a is less than 20;" << endl;
  }
cout << "value of a is : " << a << endl;
return 0;
}
```

Out put:

```
a is less than 20;
value of a is : 10
```

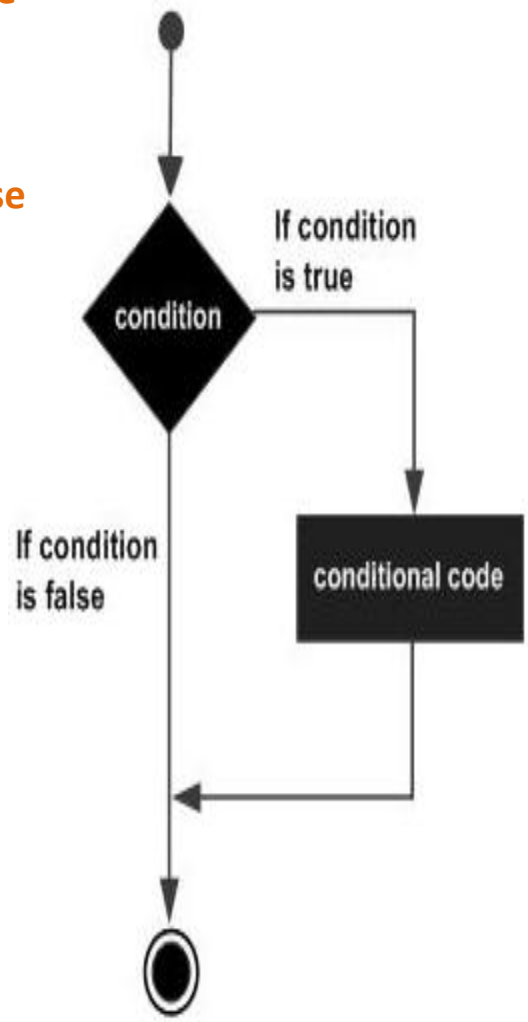
Back

If…else statement

Syntax:

```
If ( boolean_expression )
  {
    statement(s); //will execute if boolean is true
  }
else
  {
    statement(s); //will execute if boolean is false
  }
```

Flow Diagram



Example program:

```
#include <iostream>
using namespace std;
int main ()
{
int a = 100;
// check the boolean condition
if( a < 20 )
  {
   // if condition is true then print the following
   cout << "a is less than 20;" << endl;
  }
else
  {
   // if condition is false then print the following
   cout << "a is not less than 20;" << endl;
   }
cout << "value of a is : " << a << endl;
return 0;
}
```

Out put:

```
a is not less than 20;
value of a is : 100
```

Back

switch statement

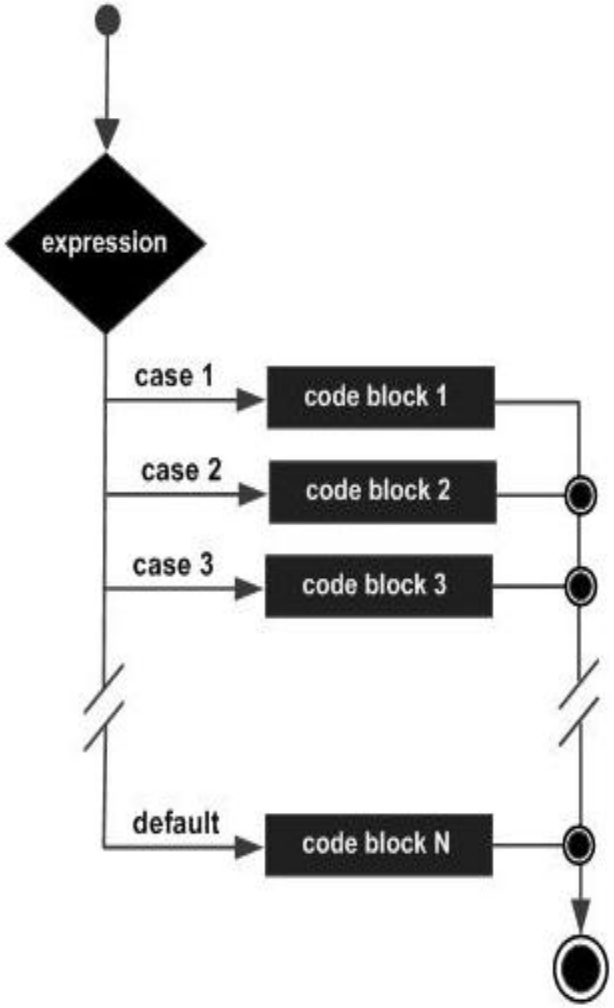
Example program:

```
#include <iostream>
using namespace std;
int main ()
{
// local variable declaration:
char grade = 'D';
switch(grade) {
case 'A' :
    cout << "Excellent!" << endl;
    break;
case 'B' :
case 'C' :
    cout << "Well done" << endl;
    break;
case 'D' :
    cout << "You passed" << endl;
    break;
case 'F' :
    cout << "Better try again" << endl;
    break;
default :
    cout << "Invalid grade" << endl;
}
cout << "Your grade is " << grade << endl;
return 0;
}
```

Syntax:

```
switch(expression){
case constant-expression :
        statement(s);
        break; //optional
case constant-expression :
        statement(s);
        break; //optional
default : //optional
        statement(s);
}
```

Flow Diagram



Out put:

```
You passed
Your grade is D
```

Back

Nested if statement

Syntax:

```
if( boolean_expression 1)
 {
  // Executes when the boolean expression 1 is true
  if(boolean_expression 2)
    {
      // Executes when the boolean expression 2 is true
    }
 }
```

Example program:

```
#include <iostream> using namespace std;
int main ()
{
 // local variable declaration:
 int a = 100;
 int b = 200;
 // check the boolean condition
 if( a == 100 )
   {
    // if condition is true then check the following
    if( b == 200 )
      {
       // if condition is true then print the following
       cout << "Value of a is 100 and b is 200" << endl;
      }
   }
 cout << "Exact value of a is : " << a << endl;
 cout << "Exact value of b is : " << b << endl;
 return 0;
}
```

Out put:

```
Value of a is 100 and b is 200
Exact value of a is : 100
Exact value of b is : 200
```

Back

function example

$$n! = 1 \times 2 \times 3 \ldots (n-2) \times (n-1) \times n$$

```
#include <iostream>
using namespace std;
int Factorial(int M);
int main()
{
   int number=0, result=0;
   while(number<1 || number>10)
    {
      cout << "Integer number = ";
      cin >> number;
    }
   result = Factorial(number);
   cout << "Factorial = " << result << endl;
   return 0;
}
```

```
int Factorial(int M)
  {
     int factorial=1;
     for(int i=1; i<=M; i++)
      {
         factorial = factorial*i;
      }
      return factorial;
  }
```

Back

function example

Static Variable Example

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

void func();
int main()
{
  func();  // prints 0  0
  func();  // prints 0  1
  func();  // prints 0  2
}
void func()
{
   int w=0;
   static int x = 0;
   cout << “Value of ‘w’ is : ” << w << “\t”;
   cout << “Value of ‘x’ is : ”  << x << endl;
   x++; w++;
}
```

Back

Array example

An example which will show array declaration, assignment and accessing arrays in C++:

```
#include <iostream>
using namespace std;
#include <iomanip>
using std::setw;
int main ()
{
   int n[ 10 ]; // n is an array of 10 integers
   // initialize elements of array n to 0
   for ( int i = 0; i < 10; i++ )
      {
        n[ i ] = i + 100; // set element at location i to i + 100
      }
   cout << "Element" << setw( 13 ) << "Value" << endl;
   // output each array element's value
   for ( int j = 0; j < 10; j++ )
     {
       cout << setw( 7 )<< j << setw( 13 ) << n[ j ] << endl;
     }
  return 0;
}
```

Out put:

Element	Value
0	100
1	101
2	102
3	103
4	104
5	105
6	106
7	107
8	108
9	109

Back

strings examples

```
# include <iostream.h>
void main(void)
{
  char data[4] = "C++";
  cout << data;
}
```

```
# include <iostream.h>
void main(void)
{
  char data[4] = {'C', '+', '+'};
  for (int i=0;i<3;i++)
     cout << data[i];
}
```

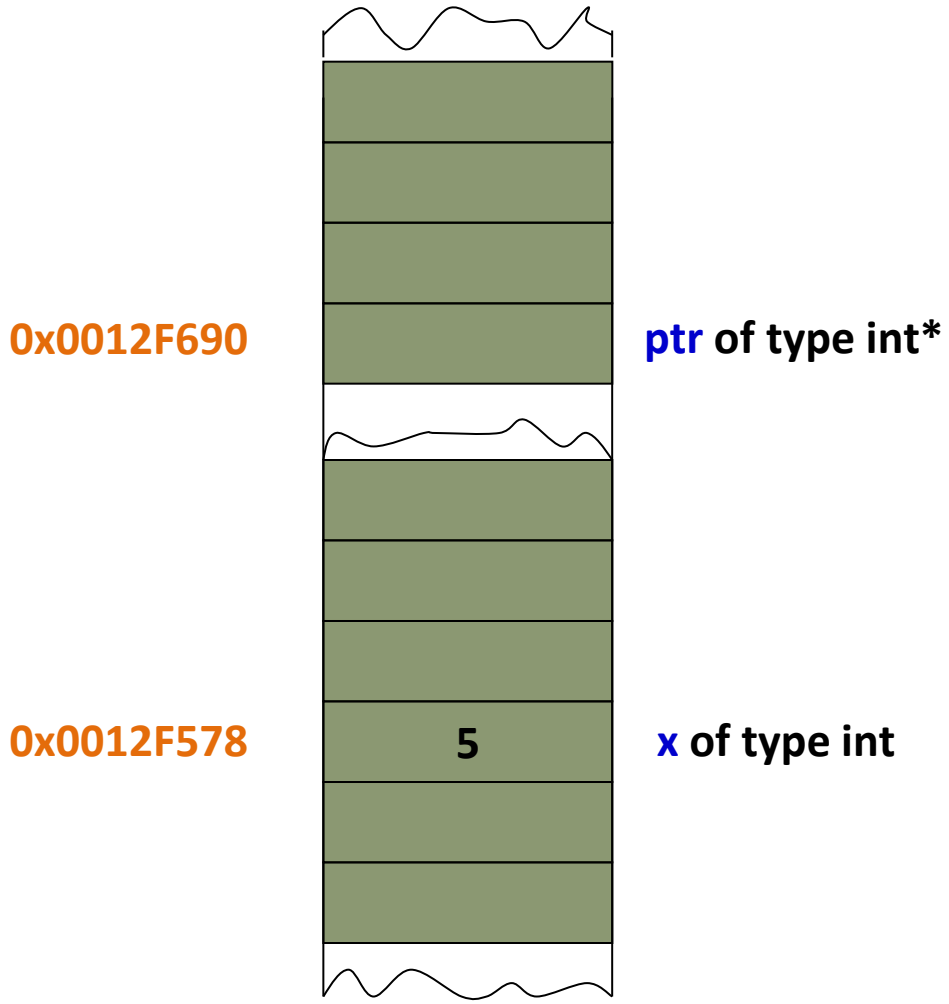
```
char str[]="This is C++";
int i=0;
while(str[i])
      {
        cout<<str[i]<<endl;
        i++;
      }
```

```
int main()
{
  char str[80];
  cout << "Enter a string: ";
  cin >> str; // read string from
   keyboard
  cout << "Here is your string: ";
  cout << str<<"\n";
  return 0;
}
```

```
#include <iostream>
int main ()
 {
 char name[256], title[256];
 cout << "Enter your name: ";
 cin.getline (name,256);
 cout << "Enter your favourite movie: ";
 cin.getline (title,256);
 cout << name << "'s favourite movie is "
<< title;
 return 0;
}
```

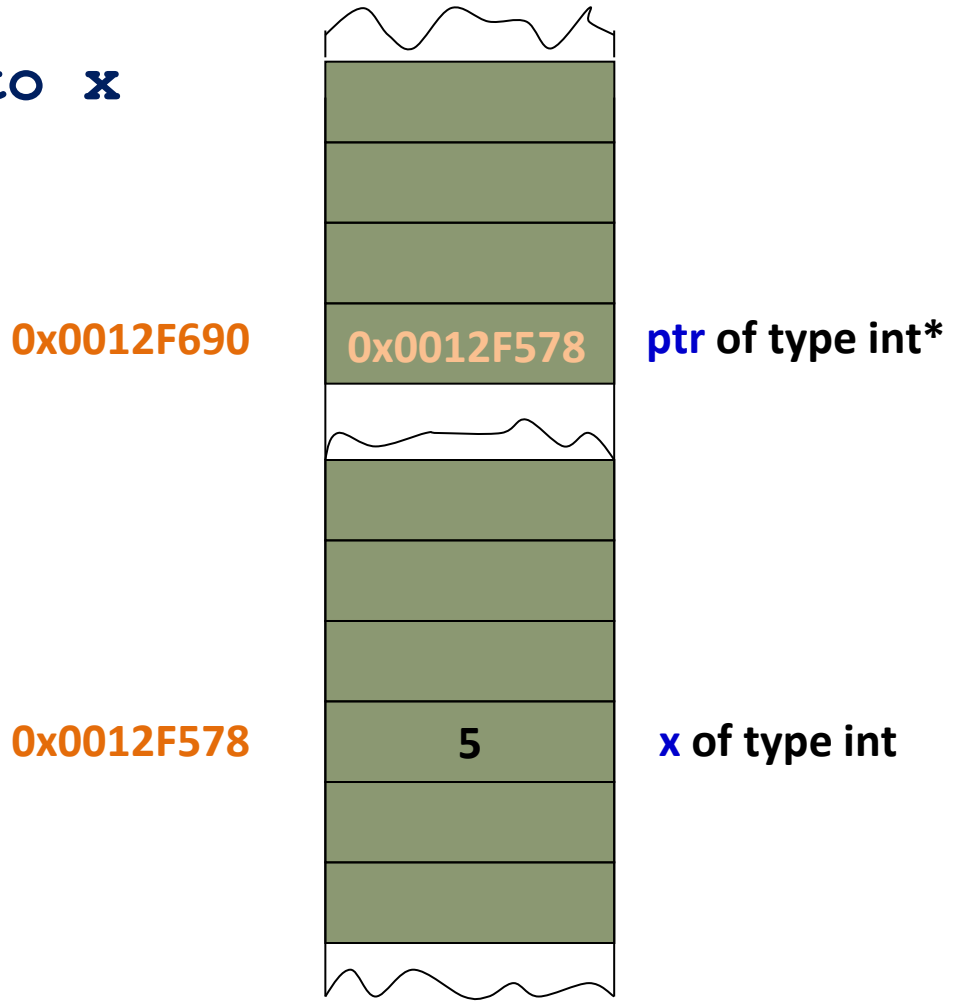
C++ pointers

```
int x = 5;
int* ptr;
```



C++ pointers

```
int x = 5;
int* ptr;
ptr = &x;    //points to x
```

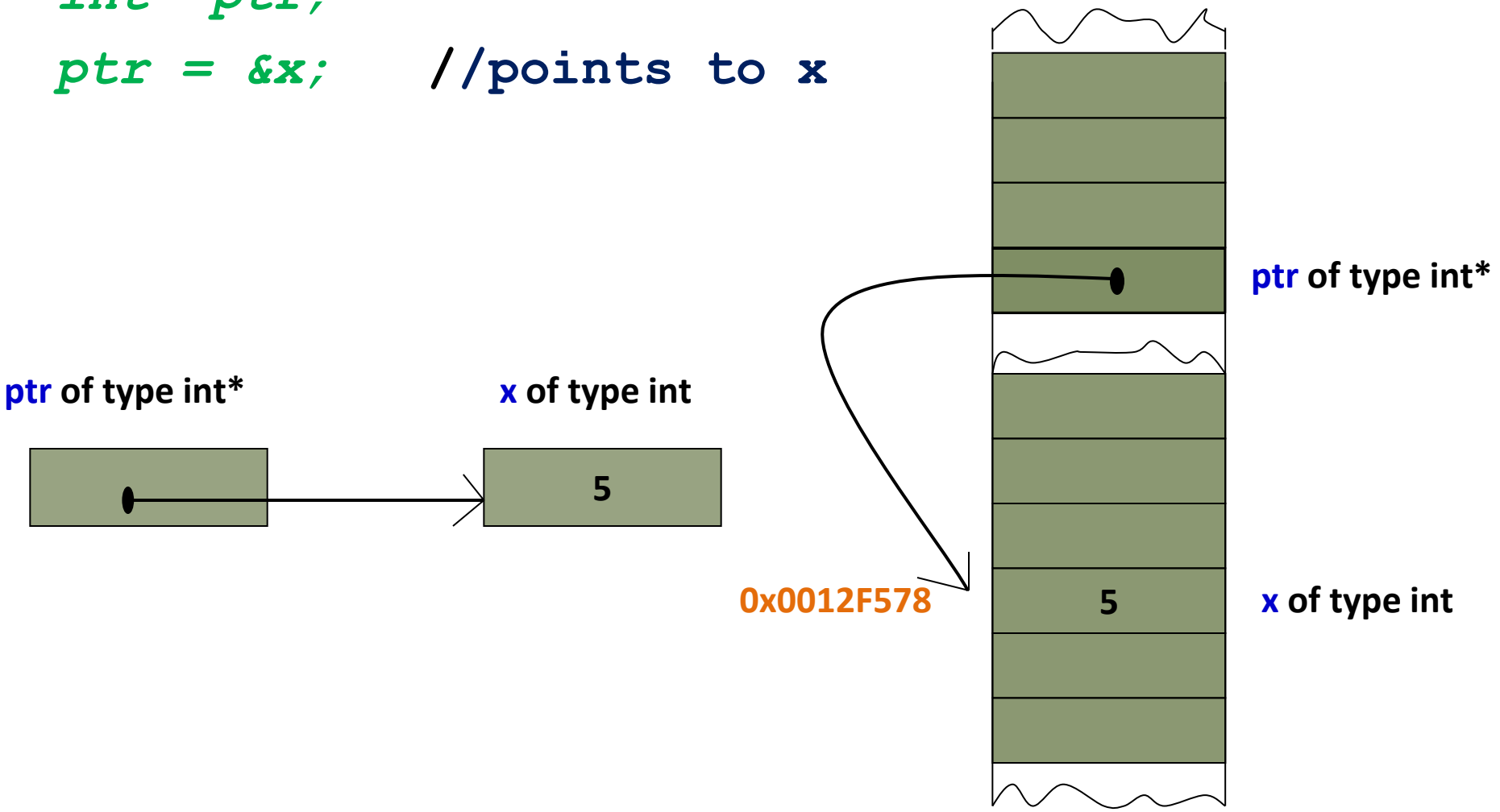


Back

C++ pointers

Pointers Example:

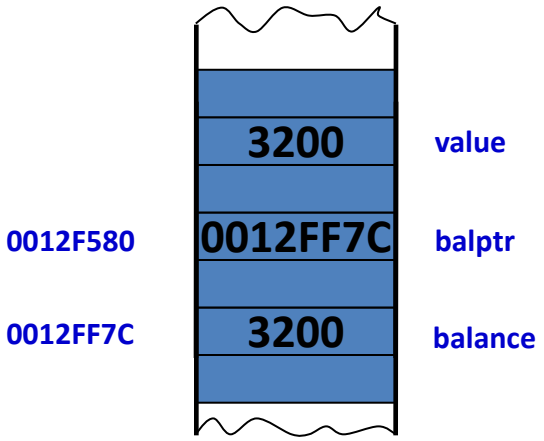
```
int x = 5;
int* ptr;
ptr = &x;    //points to x
```



Back

pointers examples

```
int main()
{
  int balance;
  int *balptr;
  int value;
  balance = 3200;
  balptr = &balance;
  value = *balptr;
  cout << "balance is: " << value << '\n';
  cout<< "Memory address where balance is
stored is:"<<balptr<<endl;
  return 0;
}
```




```
# include <iostream.h>
main()
{
 int var1 = 11;
 int var2 = 22;
 int* ptr;                  // pointer declaration
 ptr = &var1;               //initializing the pointer
 cout << ptr;               //Getting the address of
                            //var1
 cout << *ptr << endl;      //Dereferencing the pointer
 ptr = &var2;               //assigning 2nd value to the
                            //pointer
 cout << ptr;               //Getting the address of
                            //var2
 cout << *ptr << endl;      //Dereferencing the pointer
}
```

Back

structures examples

```
struct PERSON { // Declare PERSON struct type
int age; // Declare member types
long ss; float weight;
char name[25];
} family_member; // Define object of type PERSON
int main()
{
 struct PERSON sister; // C style structure declaration
 PERSON brother; // C++ style structure declaration
 sister.age = 13; // assign values to members
 brother.age = 7;
 family_member.age = 9;
}
```

```
 struct car
  {
    int model_number;
    char initials;
    double cost;
  };
 main()
  { car c1; ;/*There is now a c1 variable that has modifiable variables inside it*/
    c1.model_number = 4400;
    c1.initials = 'a';
    c1.cost = 500000;
  }
```

C++ Classes

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

class SimpleClass
{
  char *name;
  int value;

  public:
      SimpleClass(char *n, int v)
      {
         cout << " " << n << ".SimpleClass(" << n << ",
" << v << ")\n";
         name = n; value = v;
      }

      ~SimpleClass()
      {
         cout << " " << name << ".~SimpleClass()\n";
      }
```

```
void changeValue(int v)
   {
      cout << " " << name << ".changeValue(" << v <<
")\n";
      value = v;
   }

int readValue()
   {
      cout << " " << name << ".readValue()\n";
      return value;
   }

int copy(SimpleClass &sc)
   {
      cout << " " << name << ".copy(" << sc.name <<
")\n";
      value = sc.value;
   }
};
```

C++ Classes

```
int main()
{
        SimpleClass x("x", 12);
        SimpleClass y("y", 14);

        x.copy(y);
        cout << x.readValue() << "\n";

        y.changeValue(10);
        cout << y.readValue() << "\n";
}
```

Out Put:

```
 x.SimpleClass(x, 12)
 y.SimpleClass(y, 14)
 x.copy(y)
 x.readValue()
14
 y.changeValue(10)
 y.readValue()
10
 y.~SimpleClass()
 x.~SimpleClass()
```