THE UNIVERSITY OF UTAH
ASIA CAMPUS
SOUTH KOREA

Department of Electrical and Computer Engineering

Spring 2022

Introduction to Algorithms and Data Structure (CS 2420)

2. C++ Basics

Habtamu Minassie
Habtamu.aycheh@utah.edu

Simple C++ Program Structure

The // characters begin a comment

Includes header file iostream library that defines input/output. **#include directives** are enclosed in angle brackets (<… >), it refers to a part of the C++ library called a standard header files

The "entry point" of our program.

```
// C++ program structure

#include<iostream>

int main()

{
std::cout<<"Welcome to C++\n";
return 0;

}
```

Signify the start and end of a series of statements (block of code)

Semicolon(;) shows end of statement

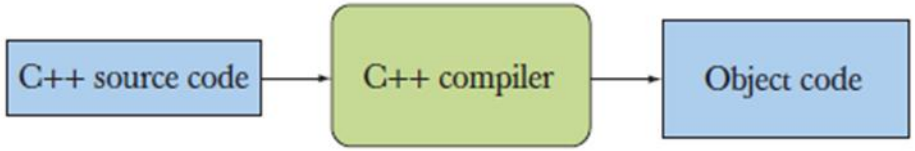
"\n" is an escape character that means "newline".

the program executed successfully [optional]

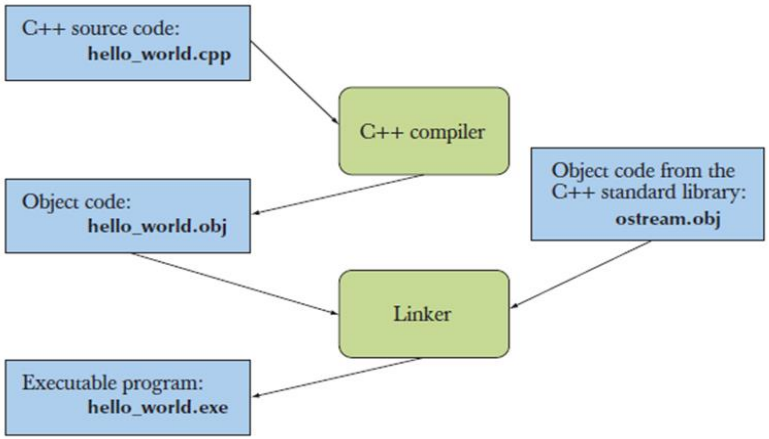
C ++ Compilers: g++ or Clang

https://www.stroustrup.com/compilers.html

Compile



Linking



g++ or clang compiler- from command line

Command Prompt

```
               2 Dir(s)  155,738,722,304 bytes free

D:\CPP>g++ --version
g++ (GCC) 11.2.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.


D:\CPP>g++ -c -o hello_world.obj hello_world.cpp

D:\CPP>g++  -o hello_world.exe hello_world.obj

D:\CPP>hello_world
Hello C++
D:\CPP>dir
 Volume in drive D is Data
 Volume Serial Number is A8AC-BCB9

 Directory of D:\CPP

02/04/2022  05:10 PM    <DIR>          .
02/04/2022  05:10 PM    <DIR>          ..
02/04/2022  12:07 PM               351 hello_world.cpp
02/04/2022  05:10 PM         2,965,139 hello_world.exe
02/04/2022  05:10 PM             1,839 hello_world.obj
               3 File(s)      2,967,329 bytes
               2 Dir(s)  155,735,752,704 bytes free
```

Developer Command Prompt for VS 2022

```
D:\CPP>clang --version
clang version 12.0.0
Target: i686-pc-windows-msvc
Thread model: posix
InstalledDir: C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Tools\Llvm\bin

D:\CPP>clang -c -o hello_world.obj hello_world.cpp

D:\CPP>clang  -o hello_world.exe hello_world.obj

D:\CPP>hello_world.exe
Hello C++
D:\CPP>dir
 Volume in drive D is Data
 Volume Serial Number is A8AC-BCB9

 Directory of D:\CPP

02/10/2022  04:12 PM    <DIR>          .
02/10/2022  04:12 PM    <DIR>          ..
02/09/2022  05:36 PM               351 hello_world.cpp
02/10/2022  04:12 PM           181,760 hello_world.exe
02/10/2022  04:12 PM            62,314 hello_world.obj
               3 File(s)        244,425 bytes
               2 Dir(s)  155,707,822,080 bytes free

D:\CPP>
```

Header files: #include <…>

- We need header files to add or include predefined libraries to our C/C++ program
- Header files contain definitions of functions and variables
- In C/C++ header files are imported by using the pre-processor #include<…> statement.
- C header files have an extension of ".h"
- Note that All C code is valid C++ code
- Example
  - #Include<iostream>
    - Tells the preprocessor to include standard input/output streams like cout and cin

Standard Input/output

```
//Example - standard input/output
#include <iostream>
/* This program inputs two numbers
x and y and outputs their sum */
int main( )
{
    int x, y;
    std::cout << "Please enter two numbers: ";
    std::cin >> x >> y; // input x and y
    int sum = x + y; // compute their sum
    std::cout << "Their sum is " << sum << std::endl;

}
```

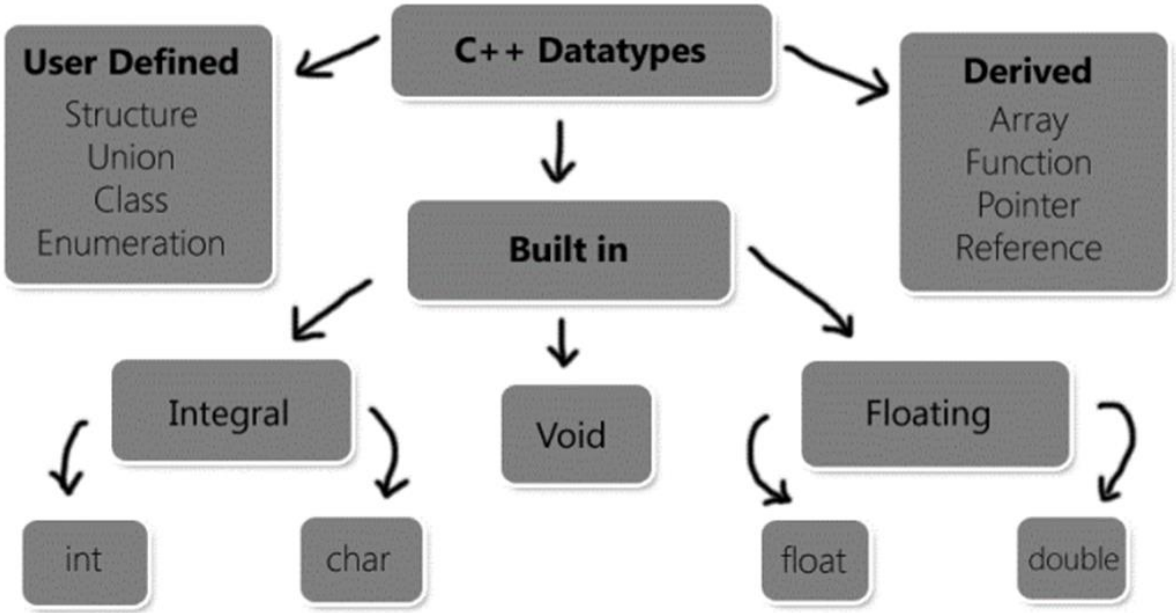
the extraction operator(<<).

insertion operator(>>)

```
Please enter two numbers: 25 30
Their sum is 55
```

Variables and Data Types

- Variables are named memory locations
- A data type defines a set of values and a set of operations that can be applied on those values.
  - Types are one of the most fundamental concepts in programming



Example

Value

```
int number = 20;
```

Data Type

Variable Name (identifier)

sizeof(int)

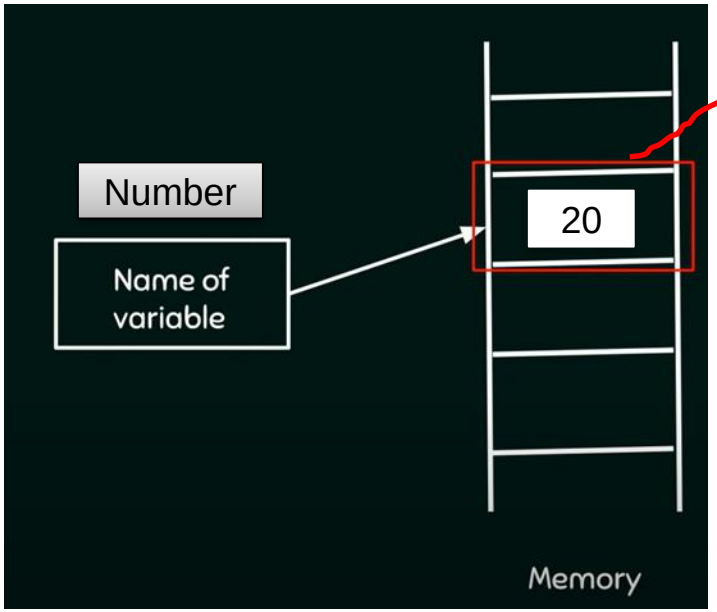
4 bytes = 32 bits

0001 0100

0000 0000

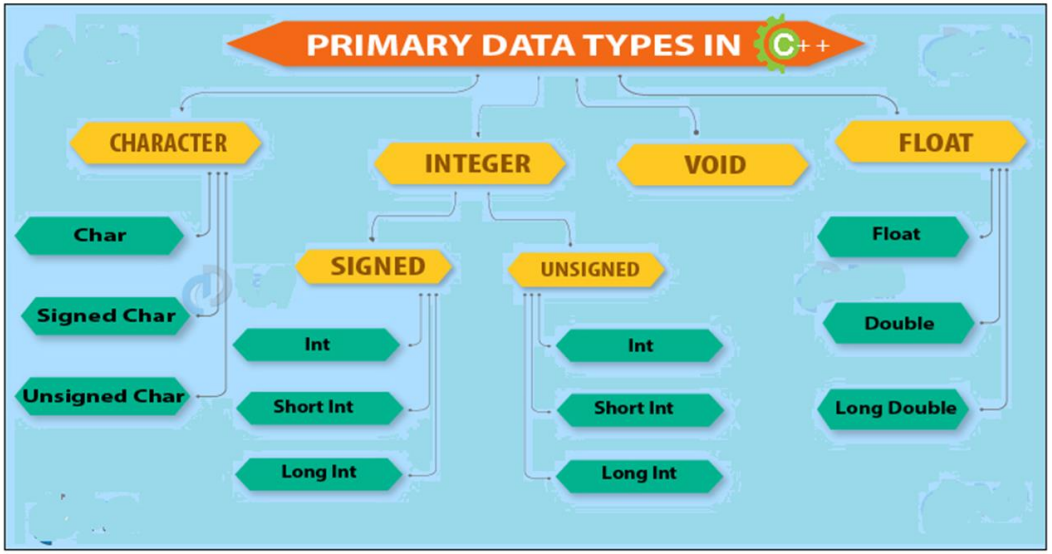
0000 0000

0000 0000



Type Modifiers

- C++ primitive data types can be modified using one or more types of suitable modifiers :
  - singed
  - unsinged
  - short
  - long



Size of C++ data types – sizeof(…)

https://en.cppreference.com/w/cpp/language/types

Data type	Size (in Bytes)	Description	Example
signed int / int	4	Stores integers values starting from -2,147,483,648 to 2,147,483,647	signed int x = -40;
unsigned int	4	Stores 0 and positive integers (0 to 4,294,967,295)	unsigned int x = 40;
short / signed short	2	Equivalent to short int or signed short int, stores small integers ranging from -32768 to 32767	short x = -2;
unsigned short	2	Equivalent to unsigned short int, stores 0 and small positive integers ranging from 0 to 65535	unsigned short x = 2;
long	4	Equivalent to long int, stores large integers	long x= 4356;
unsigned long	4	Equivalent to unsigned long int, stores 0 and large positive integers	unsigned long x = 562;
long long	8	Equivalent to long long int, stores very large integers	long long x= -243568;
unsigned long long	8	Equivalent to unsigned long long int, stores 0 and very large positive integers	unsigned long long x = 12459;
long double	16	Stores large floating-point values	long double x = 432.6781;
signed char / char	1	Stores characters ranging from -128 to 127	signed char ch = ‘b’;
unsigned char	1	Stores characters ranging from 0 to 255	unsigned char ch = ‘g’;

Variable declaration and Initialization

```
char c;      //character variable declaration.
int area;    //integer variable declaration.
float num;   //float variable declaration.
```

```
int a;  //integer variable declaration.
int b;  //integer variable declaration.
int c;  //integer variable declaration.
```

```
int a, b, c;    //more than one variable declaration.
```

```
// C++ variable initialization
int num1; // no initializer --> Default initialization
int num2 = 5; // initializer after equals sign-->Copy initialization
int num3( 6 ); // initializer in parenthesis-->Direct initialization
int num4 { 7 }; // initializer in braces -->Brace initialization
```

Best practice: Use Brace initialization in c++ for variable initialization

Example

```
// Example
#include <iostream>
int main()
{
    int sum1,sum2; // sum1 and sum2 variables declaration; not initialized
    int num1{5}; // num 1 variable declaration and initialized to 5
    int num2(10); // num2 variable declaration and initialized to 10
    int num3; // num3 variable declaratio but NOT initialized
    sum1=num1+num2;
    sum2=num1+num3;
    std::cout<<num1<<" + " <<num2<<" = "<<sum1<<std::endl;
    std::cout<<"SUM2 = "<<sum2<<std::endl; //prints some garbage value
}
```

Output

```
5 + 10 = 15
SUM2 = -800058475
```

Local and Global Scopes

- C++ statements are usually enclosed in curly braces ({...})
- Variables declared within a block are called local variables and only accessible from within the block
- Variable declared outside of any block called global variables and accessible from everywhere in the program.

```
#include<iostream>
using namespace std;
int x=10,y =10; //  x  and y are global variables
int main()
{
    int a=7,b=2; //x  and y are Local variables
    int x =5,y=5; //a  and b are local variables
    int sum1 = x+y+a+b;
    int sum2 = ::x + ::y + a + b; // The :: operator is scope resolution operator
    cout<<"Sum1 ="<<sum1<<endl;
    cout<<"Sum2 ="<<sum2<<endl;
}
```

Namespaces

- Global variables present many problems in large software systems because they can be accessed and possibly modified any where in the program => Name conflict
- namespace is a mechanism that allows a group of related names to be defined in one place.

```
#include<iostream>
using namespace std; // Standard Name space
namespace namespace1 // user defined namespace1
{
    int x=4,y=0;
}
namespace namespace2 // user defined namespace 2
{
    int x=14,y=10;
}
int main()
{
    int x=20; //Local variable
    cout<<"x from namespace1 ="<<namespace1::x<<endl;
    cout<<"y from namespace1 ="<<namespace1::y<<endl;
    cout<<"x from namespace2 ="<<namespace2::x<<endl;
    cout<<"y from namespace2 ="<<namespace2::y<<endl;
    cout<<"Local x ="<<x<<endl;
}
```

```
x from namespace1 =4
y from namespace1 =0
x from namespace2 =14
y from namespace2 =10
Local x =20
```

The using Keyword makes just std namespace accessible

```
using namespace std;
```

```
using std::cout,std::cin,std::endl;
```

or

```
std::cout<<"...";
```

C++ Constants

- In C++, we can create variables whose value cannot be changed. For that, we use the const keyword.

```
//Example
const int PI = 3.14;
const double MAX_VALUE = 2000.5;
PI= 3.142;  // error: assignment of read-only variable 'PI'
```

Type casting

- **Implicit type conversion**
  - the compiler can implicitly convert a value from one data type to another

```
double d = 10 / 4; // does integer division, initializes d with value 2.0
```


```
double d = 10 / 4; // does integer division, initializes d with value 2.0
```


```
int x { 10 };
int y { 4 };
double d = x / y; // does integer division, initializes d with value 2.0
```

$$T_C = (T_F - 32) \cdot \frac{5}{9}$$

```
double TF { 100.0 };
double TC {};

TC = ( TF - 32 ) * ( 5 / 9 );  // Compiles but does not compute what is expected...
cout << TF << " Fahrenheit = " << TC << " Celsius" << endl;
```

```
100 Fahrenheit = 0 Celsius
```

```
TC = ( TF - 32.0 ) * ( 5.0 / 9.0 );
```

```
100 Fahrenheit = 37.7778 Celsius
```

Explicit type conversion

- C++ major type casting methods are:
  - C-style casts and
  - static casts

C-style casts

```
#include <iostream>

int main()
{
    int x { 10 };
    int y { 4 };


    double d { (double)x / y }; // convert x to a double so we get floating point division
    std::cout << d; // prints 2.5

    return 0;
}
```

static casts

- C++ introduces a casting operator called **static_cast**, which can be used to convert a value of one type to a value of another type.

```
int main()
{
    char c { 'a' };
    std::cout << c << ' ' << static_cast<int>(c) << '\n'; // prints a 97

    return 0;
}
```

```
#include <iostream>
int main()
{
    int x { 10 };
    int y { 4 };
    // static cast x to a double so we get floating point division
    double d { static_cast<double>(x) / y };
    std::cout << d; // prints 2.5
    return 0;
}
```

Best practice: Use static_cast when you need to convert a value from one type to another type.

Expressions

- An expression is a sequence of operators and their operands, that specifies a computation.

Assignment Operators

Assignment Operator	Shorthand operation
a = a + b	a += b
a = a - b	a -= b
a = a * b	a *= b
a = a / b	a /= b
a = a % b	a %= b

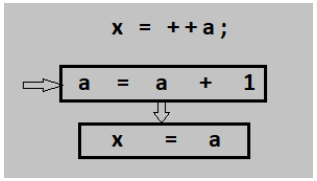
```
// Example
#include <iostream>
int main()
{
    int a=10;
    int c=a+=5;

    std::cout<<"c = "<<c <<endl;
return 0;
}
```

```
output = 15
```

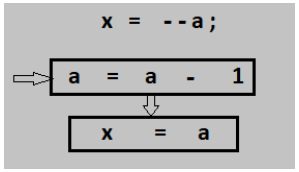
C++ Increment and decrement operator

Operator	Meaning
++	Increment Operator
--	Decrement Operator

Pre-increment

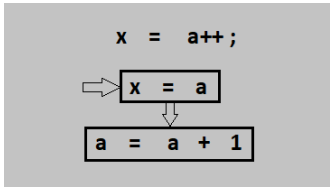


Pre-decrement

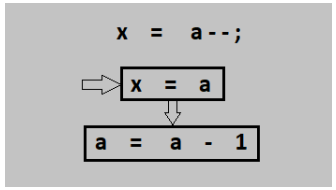


First, the value of the variable a incremented by 1 (++a) or decremented by 1(--a) and store in the memory location of variable **a**. Second, the value of variable **a** will be assigned to the variable **x**

Post-increment



Post-decrement

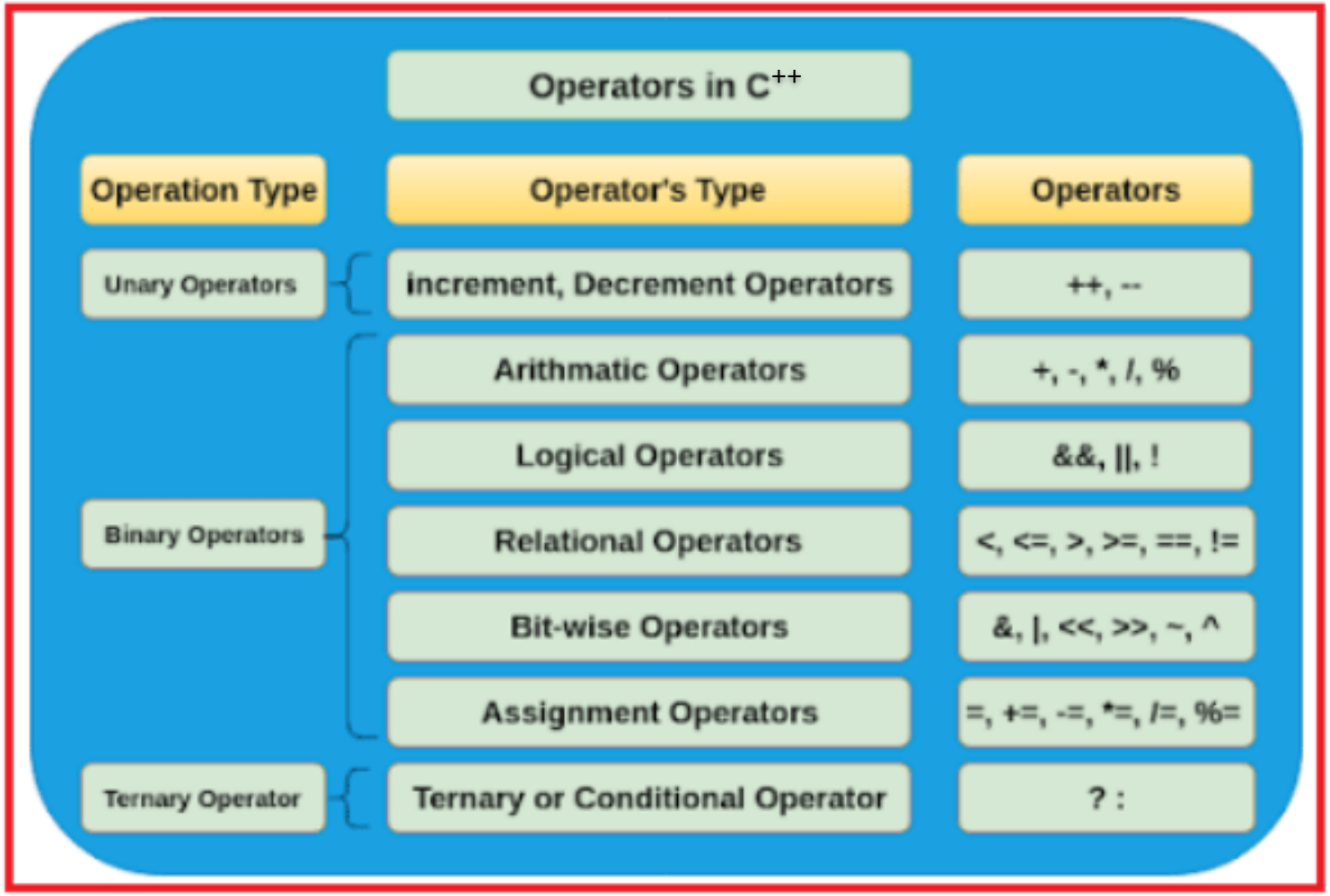



```cpp
// Example
#include <iostream>
int main()
{
    int a=5;
    int x=a;
    std::cout<<a++<<endl;
    std::cout<<a<<endl;
    std::cout<<++a<<endl;
    std::cout<<a<<endl;
return 0;
}
```

```
5
6
7
7
```

First, the value of the variable a will assign to the variable x. Second, the value of the variable a will be incremented by 1(a++) or decremented(a--) and store in the memory location of the variable a.

Summary of C++ Operators



Example : C++ Ternary Operator

```cpp
#include <iostream>
#include <string>
using namespace std;
int main() {
    double marks;
    // take input from users
    cout << "Enter your marks: ";
    cin >> marks;
    // ternary operator checks if
    // marks is greater than 40
    string result = (marks >= 40) ? "passed" : "failed";
    cout << "You " << result << " the exam.";
    return 0;
}
```

```
Enter your marks: 54
You passed the exam.
```

Control Flow - Conditions

- Every programming language includes a way of making choices

```
if ( condition )
        true_statement
else if ( condition )
        else_if_statement
else
        else_statement
```

While and Do-While Loops

while (*condition*)
    loop_body_statement

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

int main()
{
    int i = 1;
    int sum = 0;
    while (i < 100 )
    {
        sum += i;
        i++;
    }
    cout<<"sum ="<<sum<<endl;
}
```

do
    loop_body_statement
while (*condition*)

```
#include<iostream>
using namespace std;
int main()
{
    int i = 1;
    int sum = 0;
    do
    {
        sum += i;
        i++;
    } while(i < 100 );
    cout<<"sum ="<<sum<<endl;
}
```

For Loop

for (*initialization* ; *condition* ; *increment*)
loop_body_statement

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

int main()
{
    int sum = 0;
    for (int i =1; i < 100; i++)
        sum +=i;
    cout<<"Sum ="<<sum<<endl;
}
```

```
Sum = 4950
```

- **Pointers**
  - Each program variable is stored in the computer's memory at some location, or address
  - A pointer is a variable that holds the value of such an address

```
#include<iostream>
using namespace std;
int main()
{
char ch = 'Q';
char* p = &ch; // p holds the address of ch
cout << *p; // outputs the character 'Q'
ch = 'Z'; // ch now holds 'Z'
cout << *p; // outputs the character 'Z'
*p = 'X'; // ch now holds 'X'
cout << ch; // outputs the character 'X'
}
```

Caution

```
int* x, y, z; // same as: int* x; int y; int z;
```

Arrays

- An array is a collection of elements of the same type
- Each element of the array is referenced by its index
- Once declared, it is not possible to increase the number of elements in an array.

```
#include<iostream>
using namespace std;
int main()
{
double f[5]; // array of 5 doubles: f[0], . . ., f[4]
int m[10]; // array of 10 ints: m[0], . . ., m[9]
f[4] = 2.5;
m[2] = 4;
cout << f[m[2]]; // outputs f[4], which is 2.5
}
```

```
int a[ ] = {10, 11, 12, 13}; // declares and initializes a[4]
bool b[ ] = {false, true}; // declares and initializes b[2]
char c[ ] = {'c', 'a', 't'}; // declares and initializes c[3]
```

Pointers and Arrays

- There is an interesting connection between arrays and pointers

```
char c[ ] = {'c', 'a', 't'};
char* p = c; // p points to c[0]
char* q = &c[0]; // q also points to c[0]
cout << c[2] << p[2] << q[2]; // outputs “ttt”
```

Strings

- C++ provides a string type as part of its Standard Template Library (STL)

```
#include<iostream>
#include <string>
using namespace std;
using std::string;
int main()
{
    string s = "to be";
    string t = "not " + s; // t = “not to be”
    string u = s + " or " + t; // u = “to be or not to be”
    if (s > t) // true: “to be” > “not to be”
        cout << u; // outputs “to be or not to be”

}
```

```
#include<iostream>
#include <string>
using namespace std;
using std::string;
int main()
{
    string s = "John"; // s = “John”
    int i = s.size(); // i = 4
    char c = s[3]; // c = ’n’
    s += " Smith"; // now s = “John Smith”
    cout<<s<<endl;

}
```

C-Style Structures

- A structure is useful for storing an aggregation of elements.
- Unlike an array, the elements of a structure may be of different types

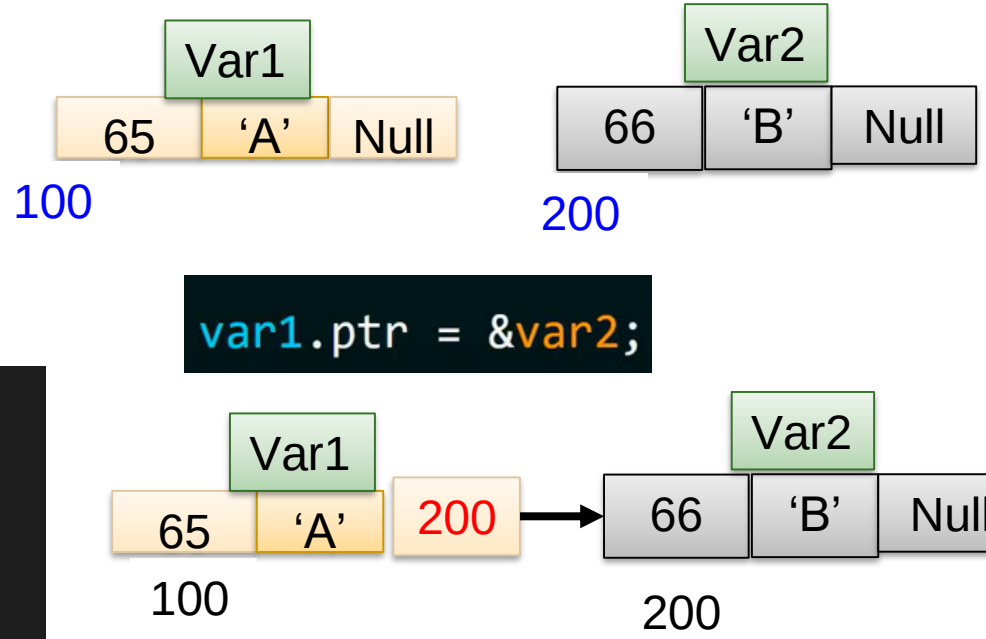
```cpp
#include <iostream>
using namespace std;
struct employee {
    int empID;
    char name[50];
    int salary;
    char department[50];
};
int main() {
    struct employee emp[3] = {
        { 1 , "Harry" , 20000 , "Finance" } ,
        { 2 , "Sally" , 50000 , "HR" } ,
        { 3 , "John" , 15000 , "IT" }
        };
    cout<<"The employee information is given as follows:"<<endl;
    cout<<endl;
    for(int i=0; i<3;i++) {
        cout<<"Employee ID: "<<emp[i].empID<<endl;
        cout<<"Name: "<<emp[i].name<<endl;
        cout<<"Salary: "<<emp[i].salary<<endl;
        cout<<"Department: "<<emp[i].department<<endl;
        cout<<endl;
    }
    return 0;
}
```

```
The employee information is given as follows:

Employee ID: 1
Name: Harry
Salary: 20000
Department: Finance

Employee ID: 2
Name: Sally
Salary: 50000
Department: HR

Employee ID: 3
Name: John
Salary: 15000
Department: IT
```

Self Referential Structures

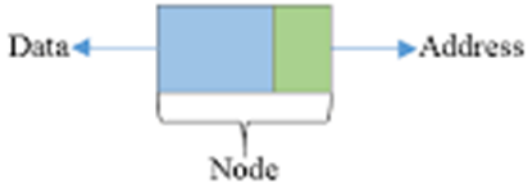
- Self referential structures are those structures in which one or more pointers points the structure of the same type

```
#include<iostream>
using namespace std;
struct code
{
    int num;
    char ch;
    code *ptr;
};
int main()
{
    code var1;
    code var2;
    var1.num=65;
    var1.ch ='A';
    var1.ptr =nullptr;

    var2.num=66;
    var2.ch ='B';
    var2.ptr =nullptr;
    var1.ptr =&var2;

    cout<<var1.ptr->num<<" "<<var1.ptr->ch<<endl;
}
```




```
OUTPUT: 66 B
```

Self Referential Structures very useful in linked list




```
#include <iostream>

using namespace std;

struct node
{
    int data;
    node *next;
};
```