C++ Basics

Simple C++ Program

A Simple C++ Program

```
#include <iostream> //include header file
using namespace std;
int main()
{
    cout << "Hello World"; // C++ statement
    return 0;
}
```

Program: Basic C++ program

```cpp
#include <iostream>
using namespace std;
int main()
{
	cout << "Name: XYZ";
	cout << "City: PUNE";
	cout << "Country: INDIA";
	return 0;
}
```

Output

```
Name: XYZCity: PUNECountry: INDIA
```

Program: Basic C++ program

```
#include<iostream>
using namespace std;
int main()
{
  int number1,number2;

  cout<<"Enter First Number: ";
  cin>>number1;                //accept first number

  cout<<"Enter Second Number: ";
  cin>>number2;                //accept first number

  cout<<"Addition : ";
  cout<<number1+number2;      //Display Addition
  return 0;
}
```

C++ Tokens

C++ Tokens

- The smallest individual unit of a program is known as **token**.
- C++ has the following tokens:
  - Keywords
  - Identifiers
  - Constants
  - Strings
  - Special Symbols
  - Operators

```
#include <iostream>
using namespace std;
int main()
{
	cout << "Hello World";
	return 0;
}
```

Keywords and Identifier

- C++ reserves a set of 84 words for its own use.
- These words are called **keywords** (or reserved words), and each of these keywords has a special meaning within the C++ language.
- **Identifiers** are names that are given to various user defined program elements, such as variable, function and arrays.
- Some of Predefined **identifiers** are cout, cin, main

❑ We cannot use Keyword as user defined identifier.

Keywords in C++

asm

auto

break

case

catch

char

class

const

continue

default

delete

do

double

else

enum

extern

float

for

friend

goto

if

inline

int

long

new

operator

private

protected

public

register

return

short

signed

sizeof

static

struct

switch

template

this

throw

try

typeof

union

unsigned

virtual

void

volatile

while

Rules for naming identifiers in C++

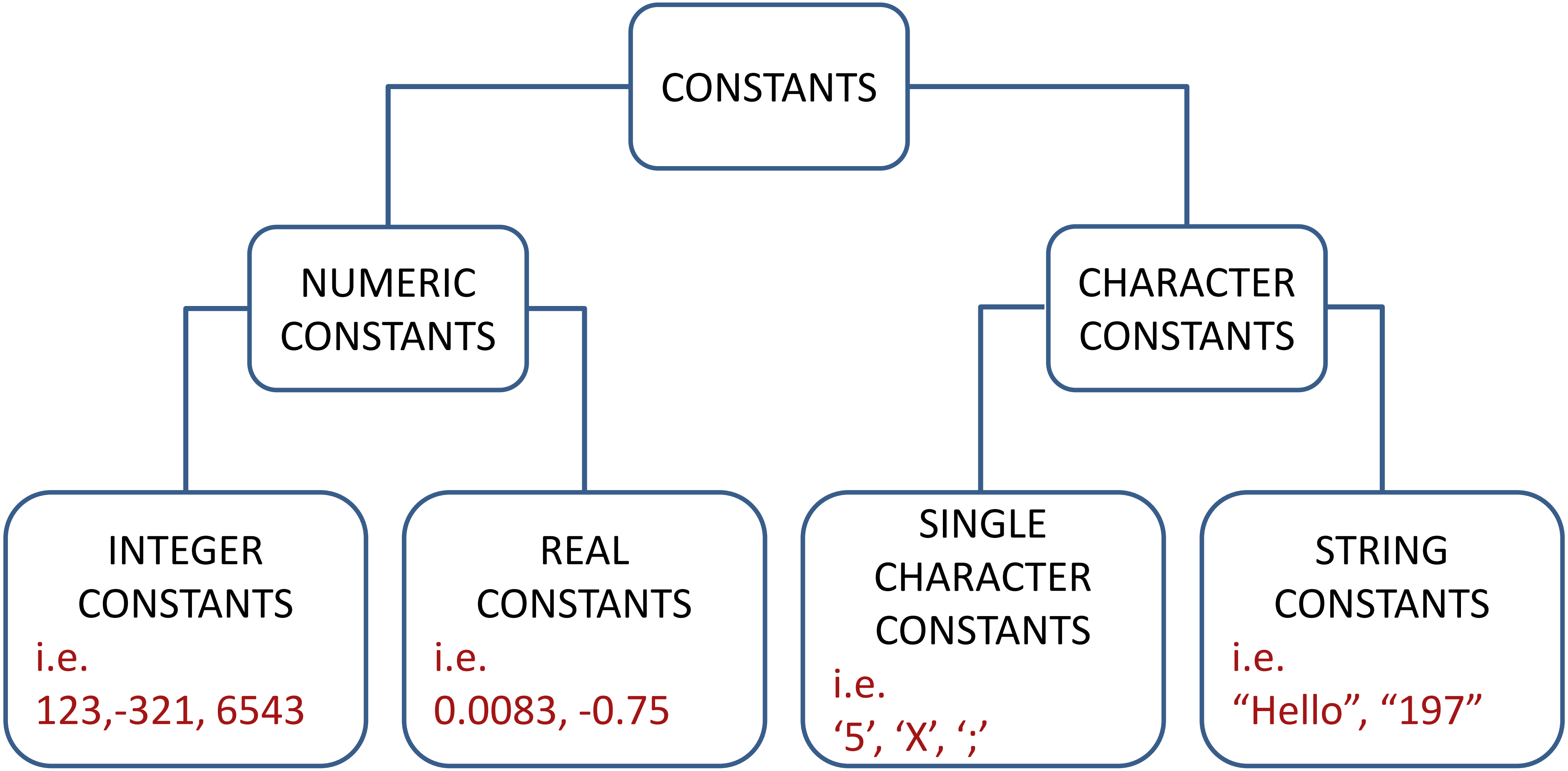
1. First Character must be an **alphabet or underscore.**
2. It can contain **only letters**(a..z A..Z), **digits**(0 to 9) or **underscore**(_).
3. Identifier name cannot be **keyword.**
4. Only first **31 characters** are significant.

Valid, Invalid Identifiers

1) Svnit	Valid	12) xyz123	Valid
2) A	Valid	13) part#2	Invalid
3) Age	Valid	14) "char"	Invalid
4) void	Reserved word	15) #include	Invalid
5) MAX-ENTRIES	Invalid	16) This_is_a_	Valid
6) double	Reserved word	17) _xyz	Valid
7) time	Valid	18) 9xyz	Invalid
8) G	Valid	19) main	Standard identifier
9) Sue's	Invalid	20) mutable	Reserved word
10) return	Reserved word	21) double	Reserved word
11) cout	Standard identifier	22) max?out	Invalid

Constants / Literals

- Constants in C++ refer to **fixed values** that do not change during execution of program.



CONSTANTS

NUMERIC CONSTANTS

CHARACTER CONSTANTS

INTEGER CONSTANTS

i.e.

123,-321, 6543

REAL CONSTANTS

i.e.

0.0083, -0.75

SINGLE CHARACTER CONSTANTS

i.e.

‘5’, ‘X’, ‘;’

STRING CONSTANTS

i.e.

“Hello”, “197”

C++ Operators

C++ Operators

- All C language operators are valid in C++.
  1. Arithmetic operators (+, - , *, /, %)
  2. Relational operators (<, <=, >, >=, ==, !=)
  3. Logical operators (&&, ||, !)
  4. Assignment operators (+=, -=, *=, /=)
  5. Increment and decrement operators (++, --)
  6. Conditional operators (?:)
  7. Bitwise operators (&, |, ^, <<, >>)
  8. Special operators ()

Logical Operators

Operator	**Meaning**
&&	Logical AND
||	Logical OR
!	Logical NOT

a	b	a && b	a || b
true	**true**		
true	**false**		
false	**true**		
false	**false**		

❑ a && b : returns false if any of the expression is false

❑ a | | b : returns true if any of the expression is true

Pre & Post Increment operator

Operator	Description
Pre increment operator (**++x**)	value of **x** is incremented before assigning it to the variable on the left

```
x = 10 ;
p = ++x;
```

First increment value of x by one

After execution
x will be **11**
p will be **11**

Operator	Description
Post increment operator (**x++**)	value of **x** is incremented after assigning it to the variable on the left

```
x = 10 ;
p = x++;
```

First assign value of x

After execution
x will be **11**
p will be **10**

What is the output of this program?

```
#include <iostream>
using namespace std;
int main ()
{
    int x, y;
    x = 5;
    y = ++x * ++x;
    cout << x << y;
    x = 5;
    y = x++ * ++x;
    cout << x << y;
}
```

(A)	749735
(B)	736749
(C)	367497
(D)	none of the mentioned

Conditional Operator

Syntax:

exp1 ? exp2 : exp3

Working of the ? Operator:

- exp1 is evaluated first
  - if exp1 is true(nonzero) then
    - exp2 is evaluated and its value becomes the value of the expression
  - If exp1 is false(zero) then
    - exp3 is evaluated and its value becomes the value of the expression

```
Ex:
m=2;
n=3;
r=(m>n) ? m : n;
```

Value of **r** will be **3**

```
Ex:
m=2;
n=3;
r=(m<n) ? m : n;
```

Value of **r** will be **2**

Bitwise Operator

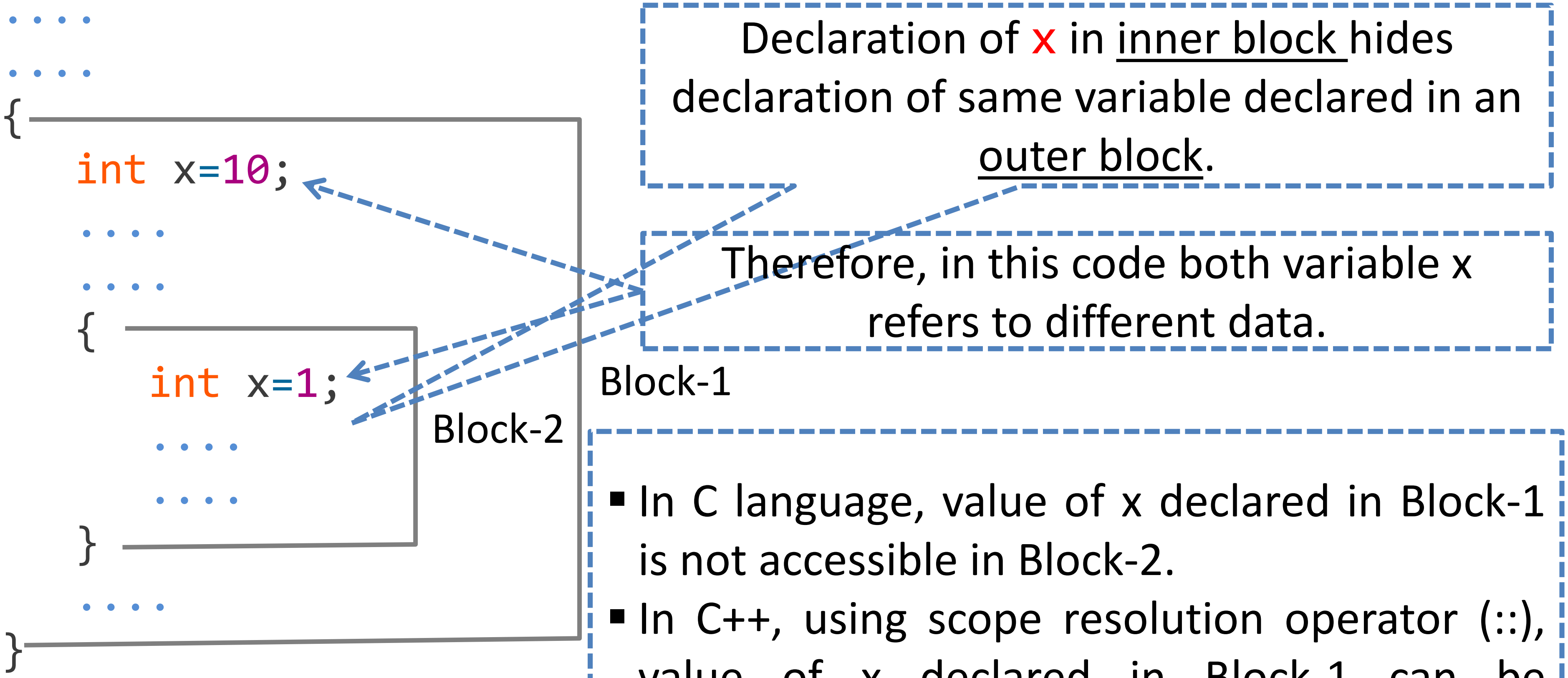
Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	Shift left
>>	Shift right

New Operators in C++

`::`	Scope Resolution	It allows to access to the global version of variable
`::*`	Pointer-to-member declarator	Declares a pointer to a member of a class
`->*`	Pointer-to-member operator	To access pointer to class members
`.*`	Pointer-to-member operator	To access pointer to data members of class
`new`	Memory allocation operator	Allocates memory at run time
`delete`	Memory release operator	Deallocates memory at run time
`endl`	Line feed operator	It is a manipulator causes a linefeed to be inserted
`setw`	Field width operator	It is a manipulator specifies a field width for printing value

Scope Resolution Operator

Scope Resolution Operator(::)



```
....
....
{
   int x=10;
   ....
   ....
   {
      int x=1;
      ....
      ....
   }
   ....
}
```

Declaration of x in inner block hides declaration of same variable declared in an outer block.

Therefore, in this code both variable x refers to different data.

- In C language, value of x declared in Block-1 is not accessible in Block-2.
- In C++, using scope resolution operator (::), value of x declared in Block-1 can be accessed in Block-2.

Scope resolution example

```cpp
#include <iostream>
using namespace std;
int m=10;
int main()
{
   int m=20;
   {
     int k=m;
     int m=3;
     cout<<"we are in inner block\n";
     cout<<"k="<<k<<endl;
     cout<<"m="<<m<<endl;
     cout<<"::m="<<::m<<endl;
   }
   cout<<"we are in outer block\n";
   cout<<"m="<<m<<endl;
   cout<<"::m="<<::m<<endl;
   return 0;
}
```

Global declaration of variable m

variable m declared , local to main

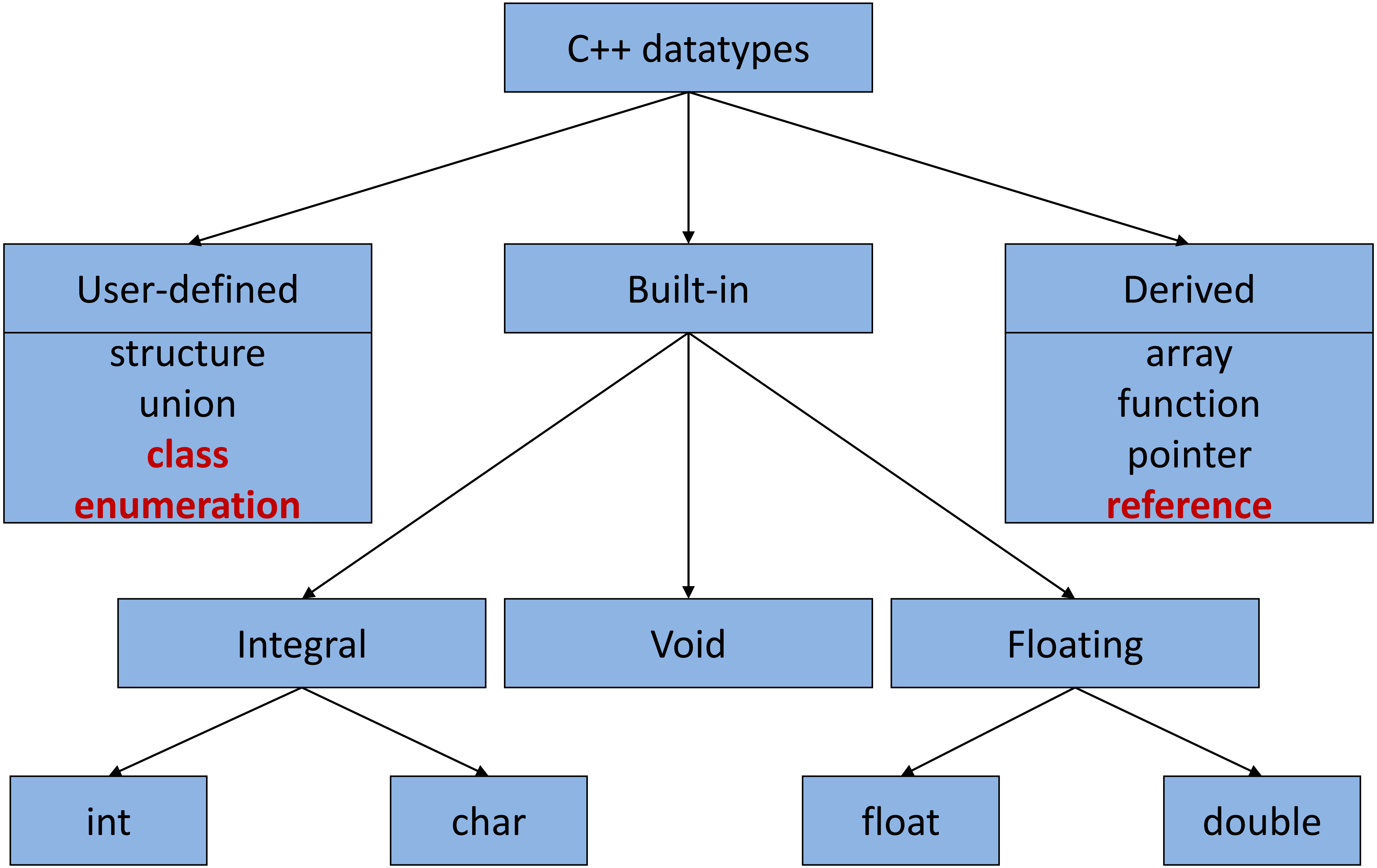
variable m
declared again local to inner block

Output:

```
we are in inner block
k=20
m=3
::m=10
we are in outer block
m=20
::m=10
```

C++ Data Types

Basic Data types



C++ datatypes

User-defined

structure

union

class

enumeration

Built-in

Derived

array

function

pointer

reference

Integral

Void

Floating

int

char

float

double

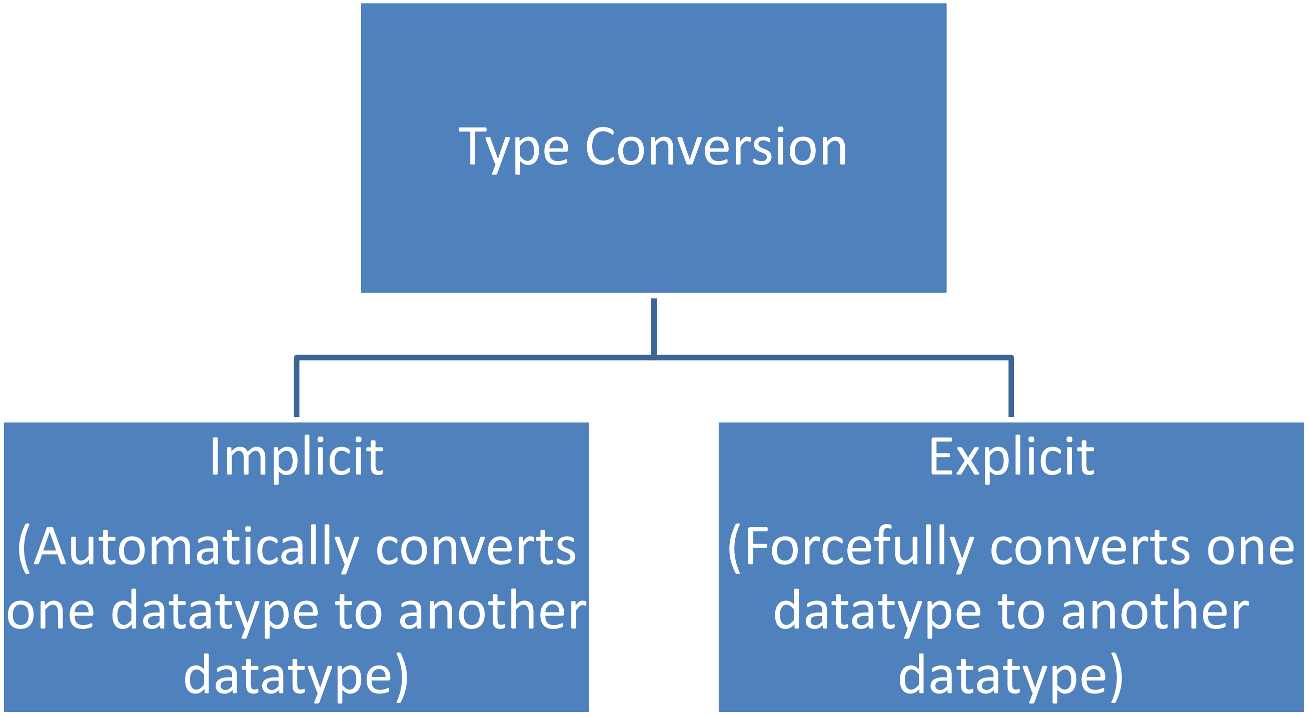
Built in Data types

Data Type	Size (bytes)	Range
char	1	-128 to 127
unsigned char	1	0 to 255
short or int	2	-32,768 to 32,767
unsigned int	2	0 to 65535
long	4	-2147483648 to 2147483647
unsigned long	4	0 to 4294967295
float	4	3.4e-38 to 3.4e+308
double	8	1.7e-308 to 1.7e+308
long double	10	3.4e-4932 to 1.1e+4932

Type Conversion

Type Conversion

- **Type Conversion** is the process of converting one predefined data type into another data type.



Type Conversion

Implicit

(Automatically converts one datatype to another datatype)

Explicit

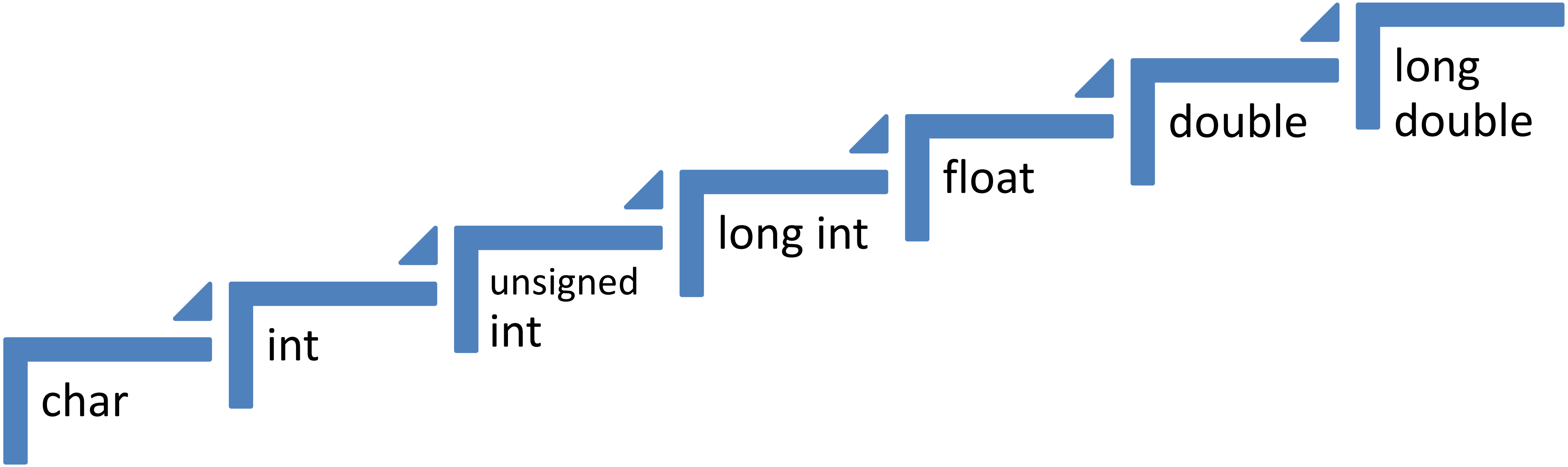
(Forcefully converts one datatype to another datatype)

- Explicit type conversion is also known as **type casting**.

Type Conversion(Cont…)

```
int a;

double b=2.55;

a = b; // implicit type conversion

cout << a << endl; // this will print 2

a = int(b); //explicit type conversion

cout << a << endl; // this will print 2
```

Implicit type conversion hierarchy

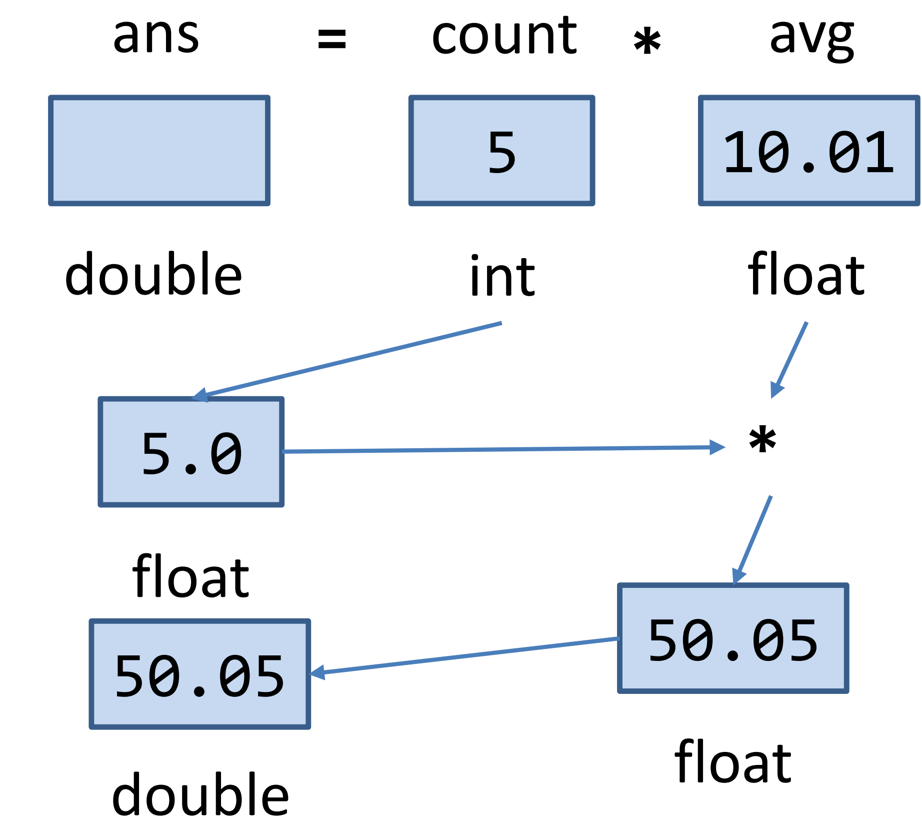


Implicit Type Conversion

```
#include <iostream>
using namespace std;
int main()
{
    int count = 5;
    float avg = 10.01;
    double ans;

    ans = count * avg;

    cout<<"Answer=:"<<ans;
    return 0;
}
```

Output:

```
Answer = 50.05
```



Type Casting

- In C++ explicit type conversion is called **type casting.**
- Syntax

```
type-name (expression) //C++ notation
```

- Example

```
average = sum/(float) i; //C notation
average = sum/float (i); //C++ notation
```

Type Casting Example

```cpp
#include <iostream>
using namespace std;
int main()
{
   int a, b, c;
   a = 19.99 + 11.99; //adds the values as float
                      // then converts the result to int
   b = (int) 19.99 + (int) 11.99;  // old C syntax
   c = int (19.99) + int (11.99); // new C++ syntax

   cout << "a = " << a << ", b = " << b;
   cout << ", c = " << c << endl;

   char ch = 'Z';
   cout << "The code for " << ch << " is ";  //print as char
   cout << int(ch) << endl;   //print as int
   return 0;
}
```

Output:

```
a = 31, b = 30, c = 30
The code for Z is 90
```

Reference Variable

Reference Variable

- A **reference** provides an alias or a different name for a variable.
- One of the most important uses for references is in passing arguments to functions.

```
int a=5;
int &ans = a;

cout<<"a="<<a<<endl;
cout<<"&a="<<&a<<endl;
cout<<"ans="<<ans<<endl;
cout<<"&ans="<<&ans<<endl;
ans++;
cout<<"a="<<a<<endl;
cout<<"ans="<<ans<<endl;
```

declares variable a

declares ans as reference to a

OUTPUT

```
a=5
&a=0x6ffe34
ans=5
&ans=0x6ffe34

a=6
ans=6
```

Its necessary to initialize the Reference at the time of declaration

Reference Variable(Cont…)

- C++ references allow you to create a second name for a variable.
- **Reference variable** for the purpose of accessing and modifying the value of the **original variable** even if the second name (the reference) is located within a **different scope**.

```
#include<iostream>
using namespace std;
int main()
{
int x = 10;
// ref is a reference to x.
int& ref = x;
// Value of x is now changed to 20
ref = 20;
cout << "x = " << x << endl ;
// Value of x is now changed to 30
x = 30;
cout << "ref = " << ref << endl ;
return 0;
}
```

```
x = 20
Ref = 30
```

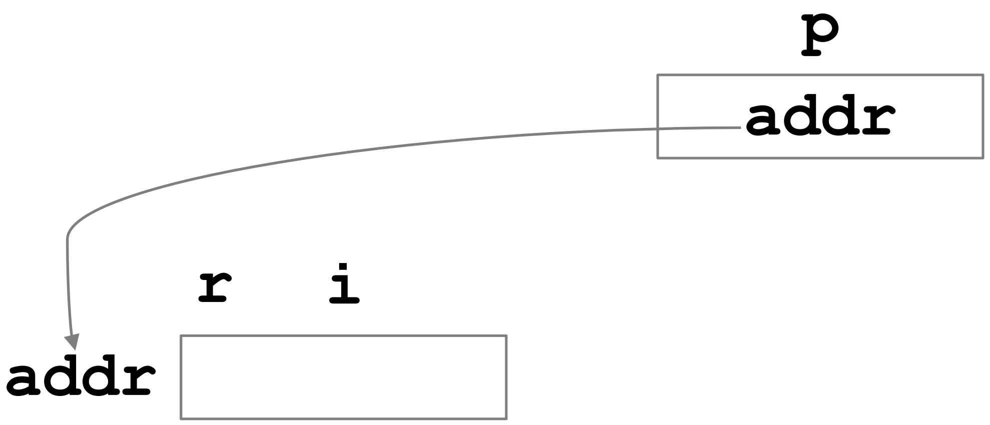
Reference Vs Pointer

References

```
int i;
int &r = i;
```

Pointers

```
int *p = &i;
```



- A reference is a variable which **refers** to another variable.

- A pointer is a variable which **stores the address** of another variable.

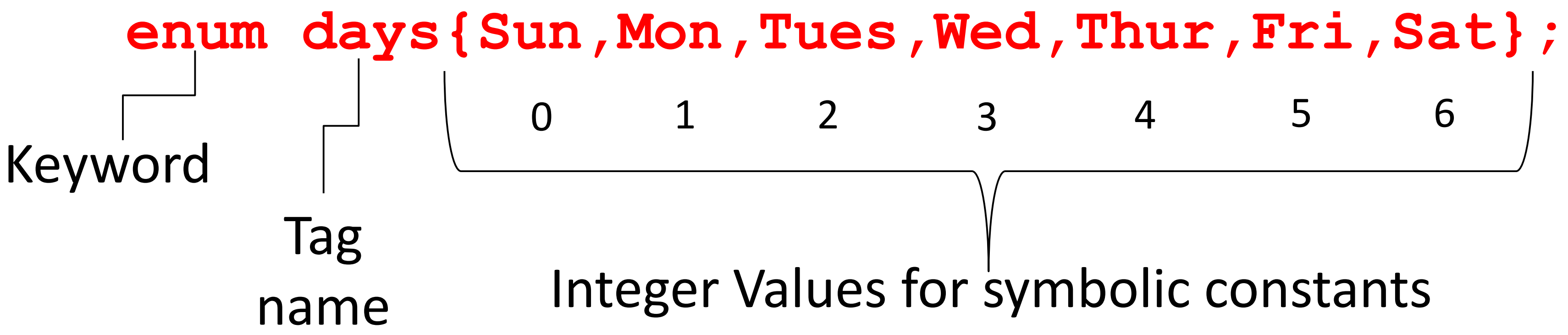
Reference Vs. Pointer

POINTER	REFERENCE
A pointer can be initialized to any value anytime after it is declared.	A reference must be initialized when it is declared.
A pointer can be assigned to point to a NULL value.	Main focus is on the data that is being operated
Various arithmetic operations can be performed on pointers	Reference Arithmetic not allowed.
Use pointers if pointer arithmetic or passing NULL-pointer is needed. For example for arrays and to implement data structures like linked list, tree, etc.	Use references in function parameters and return types.

Enumeration

Enumeration (A user defined Data Type)

- An **enumeration** is set of named **integer** constants.
- Enumerations are defined much like structures.



```
enum days{Sun,Mon,Tues,Wed,Thur,Fri,Sat};
```

0 1 2 3 4 5 6

Keyword

Tag name

Integer Values for symbolic constants

- Above statement creates `days` the name of datatype.
- By default, enumerators are assigned integer values starting with 0.
- It establishes `Sun`, `Mon`… and so on as symbolic constants for the integer values 0-6.

Enumeration Behaviour(Cont…)

```
enum coin { penny, nickel, dime, quarter=100,
            half_dollar, dollar};
```

The values of these symbols are

penny	0
nickel	1
dime	2
quarter	100
half_dollar	101
dollar	102

Enumeration Behaviour

```
enum days{ sun, mon, tue, wed, thu, fri, sat };
days today;
```

variable **today** declared of type **days**

```
today = tue;
```

Valid, because tue is an enumerator. Value 2 will be assigned in today

```
today = 6;
```

Invalid, because 6 is not an enumerator

```
today++;
```

Invalid, today is of type days. We can not apply ++ to structure variable also

```
today = mon + fri;
```

Invalid

```
int num = sat;
```

Valid, days data type converted to int, value 6 will be assigned to num

```
num = 5 + mon;
```

Valid, mon converted to int with value 1

Control Structures

Control Structures

- The **if** statement:
  - Simple **if** statement
  - **if…else** statement
  - **else…if** ladder
  - **if…else** nested
- The **switch** statement :
- The **do-while** statement: An exit controlled loop
- The **while** Statement: An entry controlled loop
- The **for** statement: An entry controlled loop

Thank You