Simple C++ Program

A Simple C++ Program

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

- **iostream** is just like we include **stdio.h** in c program.
- It contains declarations for the identifier **cout** and the insertion operator **<<**.
- **iostream** should be included at the beginning of all programs that use input/output statements.

A Simple C++ Program (Cont…)

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

- A namespace is a declarative region.
- A **namespace** is a part of the program in which certain names are recognized; outside of the namespace they're unknown.
- namespace defines a scope for the identifies that are used in a program.
- **using** and **namespace** are the keywords of C++.

A Simple C++ Program (Cont…)

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

- **std** is the namespace where ANSI C++ standard class libraries are defined.
- Various program components such as **cout, cin, endl** are defined within **std** namespace.
- If we don't use the **using** directive at top, we have to add the **std** followed by **::** in the program before identifier.

```
std::cout << "Hello World";
```

A Simple C++ Program (Cont…)

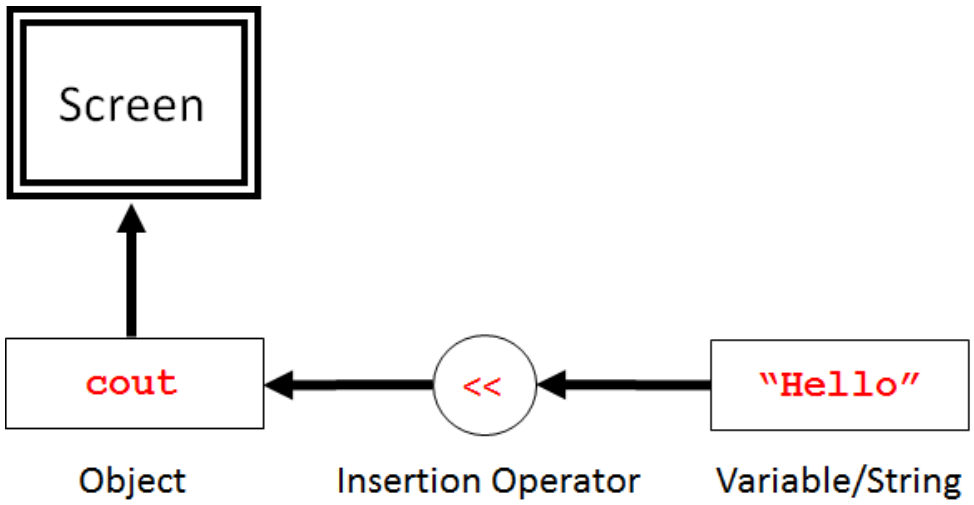
```
#include <iostream> //include header file
using namespace std;
int main()
{
      cout << "Hello World"; // C++ statement
      return 0;
}
```

- In C++, **`main()`** returns an integer type value.
- Therefore, every **`main()`** in C++ should end with a **`return 0;`** statement; otherwise error will occur.
- The return value from the **`main()`** function is used by the runtime library as the **`exit code`** for the process.

Insertion Operator <<

```
cout << "Hello World";
```

- The operator **<<** is called the insertion operator.
- It inserts the contents of the variable on its right to the object on its left.
- The identifier **cout** is a predefined object that represents standard output stream in C++.
- Here, Screen represents the output. We can also redirect the output to other output devices.
- The operator **<<** is used as bitwise left shift operator also.



Output Using Insertion Operator

Program: Basic C++ program

Write a C++ Program to print following

```
Name: Darshan
City: Rajkot
Country: India
```

Program: Basic C++ program

```
#include <iostream>
using namespace std;
int main()
{
      cout << "Name: Darshan";
      cout << "City: Rajkot";
      cout << "Country: India";
      return 0;
}
```

Output

```
Name: DarshanCity: RajkotCountry: India
```

Program: Basic C++ program(Cont…)

```cpp
#include <iostream>
using namespace std;
int main()
{
 cout << "Name: Darshan\n";
 cout << "City: Rajkot\n";
 cout << "Country: India";
 return 0;
}
```

```cpp
#include <iostream>
using namespace std;
int main()
{
 cout << "Name: Darshan"<<endl;
 cout << "City: Rajkot"<<endl;
 cout << "Country: India"<<endl;
 return 0;
}
```

Output

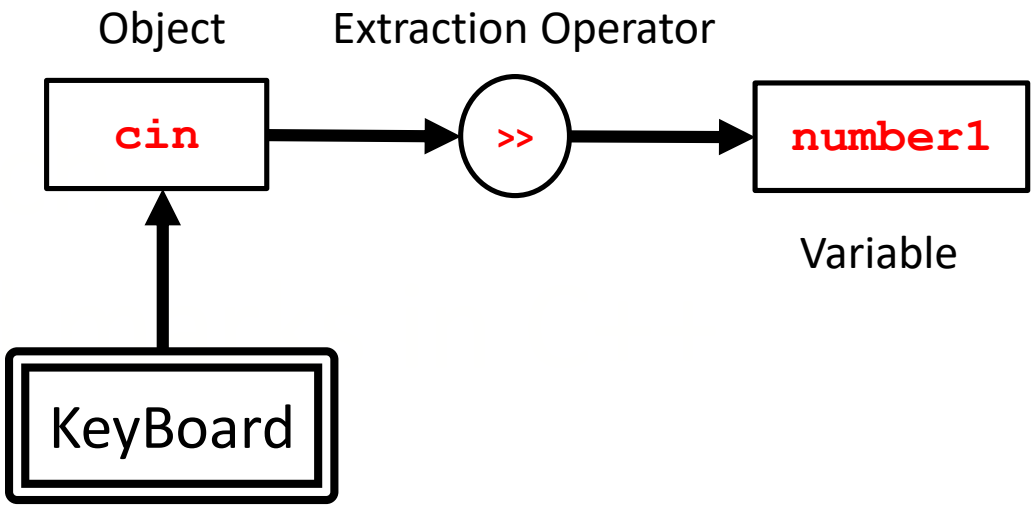
```
Name: Darshan
City: Rajkot
Country: India
```

- The **endl** manipulator and **\n** has same effect. Both inserts new line to output.
- But, difference is **endl** immediate flush to the output while **\n** do not.

Extraction Operator >>

```
cin >> number1;
```

- The operator **>>** is called the extraction operator.
- It extracts (or takes) the value from keyboard and assigns it to the variable on its right.
- The identifier **cin** is a predefined object that represents standard input stream in C++.
- Here, standard input stream represents the Keyboard.
- The operator **>>** is used as bitwise right shift operator also.



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	double	**new**	switch
auto	else	**operator**	**template**
break	enum	**private**	**this**
case	extern	**protected**	**throw**
catch	float	**public**	**try**
char	for	register	typeof
class	**friend**	return	union
const	goto	short	unsigned
continue	if	signed	**virtual**
default	**inline**	sizeof	void
delete	int	static	volatile
do	long	struct	while

Rules for naming identifiers in C++

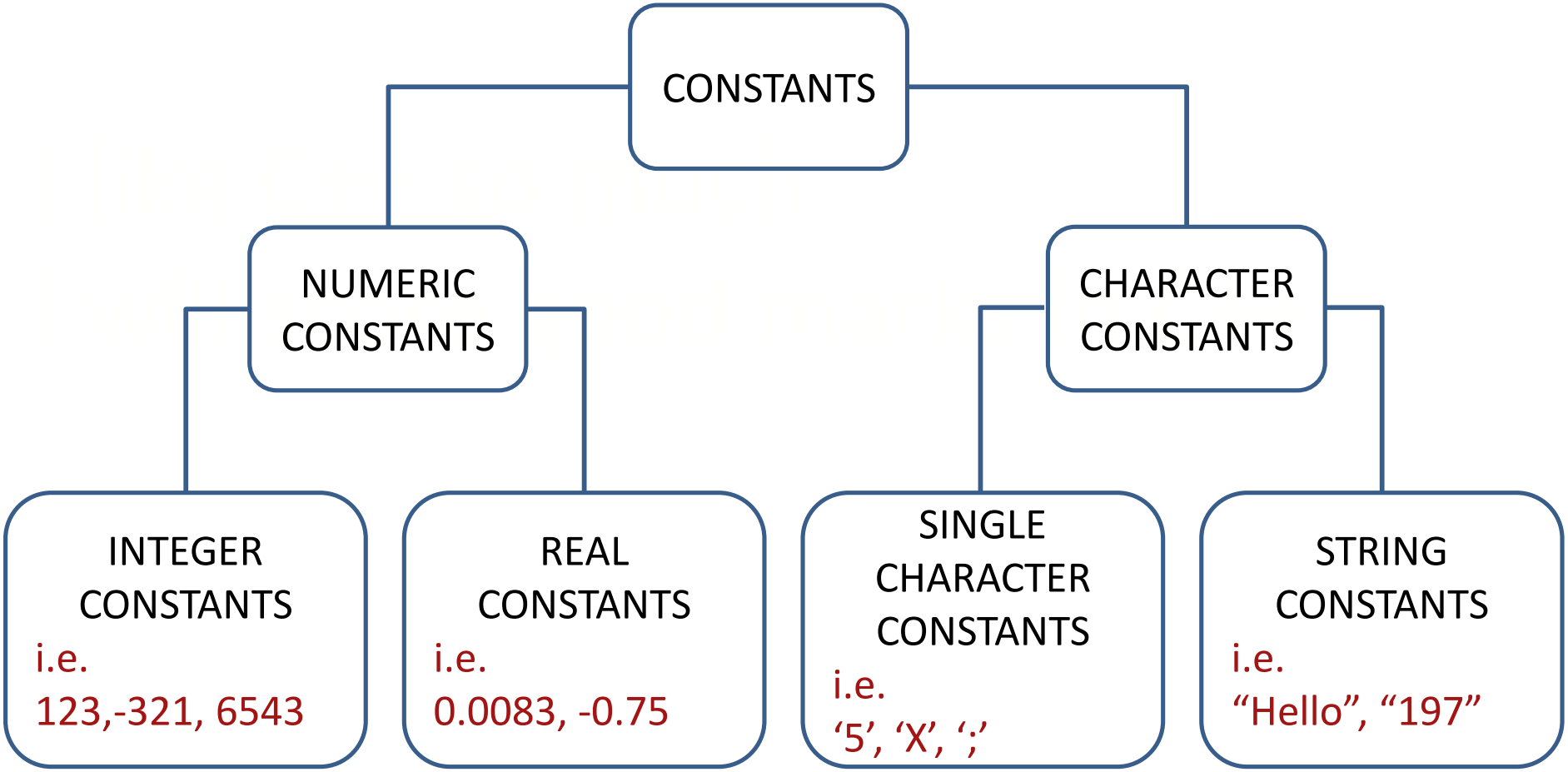
1. First Character must be an **alphabet or underscore.**
2. It can contain **only letters**(a..z A..Z), **digits**(0 to 9) or **underscore**(_).
3. Identifier name cannot be **keyword.**
4. Only first **31 characters** are significant.

Valid, Invalid Identifiers

1) Darshan	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.



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 ()

Arithmetic Operators

Operator	example	Meaning
+	a + b	Addition
-	a – b	Subtraction
*	a * b	Multiplication
/	a / b	Division
%	a % b	Modulo division- remainder

Relational Operators

Operator	Meaning
<	Is less than
<=	Is less than or equal to
>	Is greater than
>=	Is greater than or equal to
==	Equal to
!=	Not equal to

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

Assignment operator

- We assign a value to a variable using the basic assignment operator (=).
- Assignment operator stores a value in memory.
- The syntax is

```
leftSide = rightSide ;
```

Always it is a *variable identifier.*

It is either a *literal* | a *variable identifier* | an *expression.*

```
Literal: ex. i = 1;
Variable identifier: ex. start = i;
Expression: ex. sum = first + second;
```

Assignment Operators (Shorthand)

Syntax:

```
leftSide  Op= rightSide ;
```

It is an *arithmetic operator.*

```
Ex:
    x=x+3;
    x+=3;
```

Simple assignment operator	Shorthand operator
a = a+1	a += 1
a = a-1	a -= 1
a = a * (m+n)	a *= m+n
a = a / (m+n)	a /= m+n
a = a % b	a %= b

Increment and Decrement Operators

- **Increment ++**
  The ++ operator used to increase the value of the variable by one
- **Decrement – –**
  The – – operator used to decrease the value of the variable by one

Example:

```
x=100;
x++;
```

After the execution the value of x will be 101.

Example:

```
x=100;
x--;
```

After the execution the value of x will be 99.

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

Bitwise Operator Examples

```
8 = 1000 (In Binary)
6 = 0110 (In Binary)
```

Bitwise & (AND)

```
int a=8,b=6,c;
c = a & b;
cout<<"Output ="<< c;
```

```
Output = 0
```

Bitwise | (OR)

```
int a=8,b=6,c;
c = a | b;
cout<<"Output ="<< c;
```

```
Output = 14
```

Bitwise << (Shift Left)

```
int a=8,b=6,c;
c = a << 1;
cout<<"Output ="<< c;
```

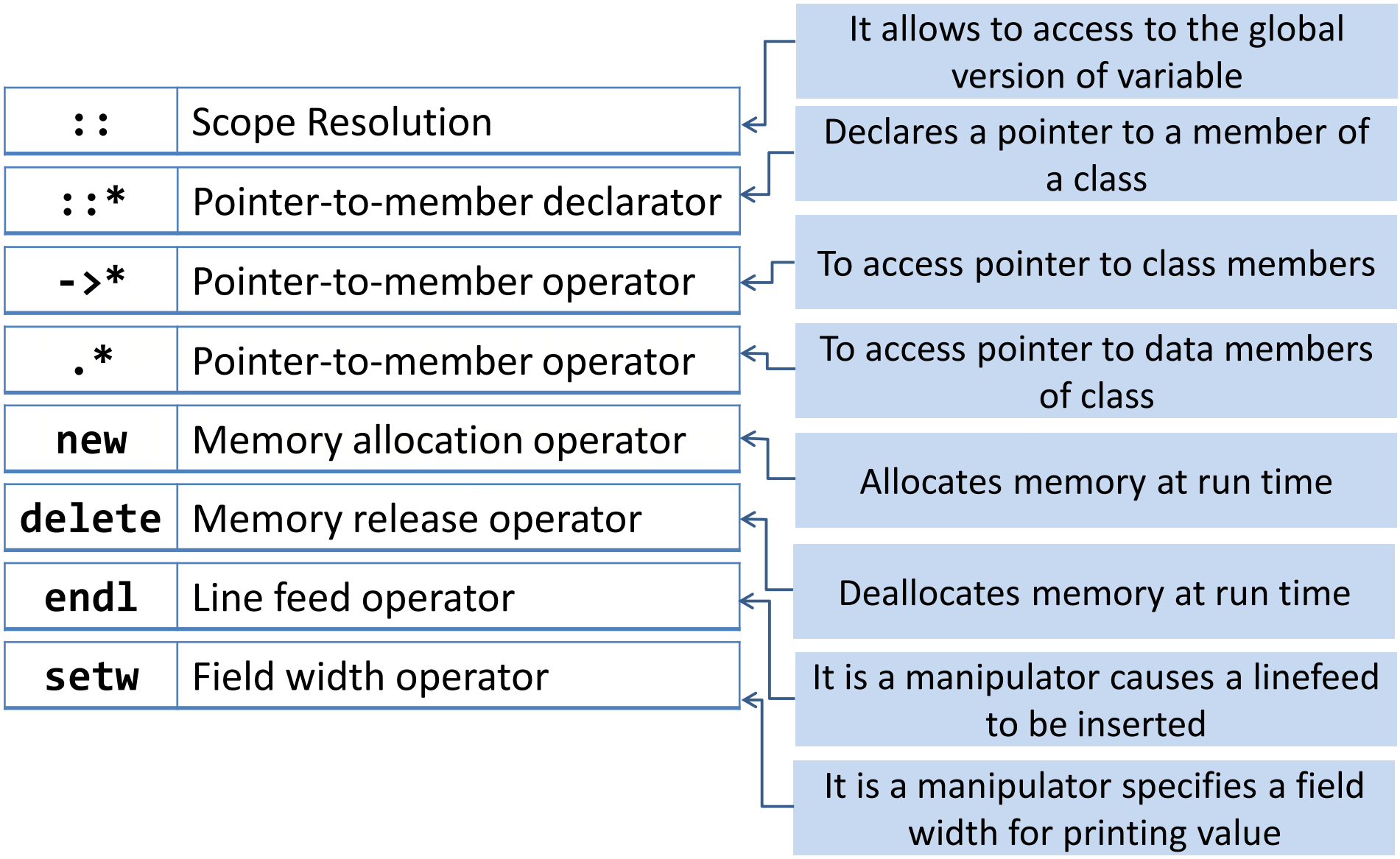
```
Output = 16
left shifting is the equivalent of
multiplying a by a power of two
```

Bitwise >> (Shift Right)

```
int a=8,b=6,c;
c = a >> 1;
cout<<"Output ="<< c;
```

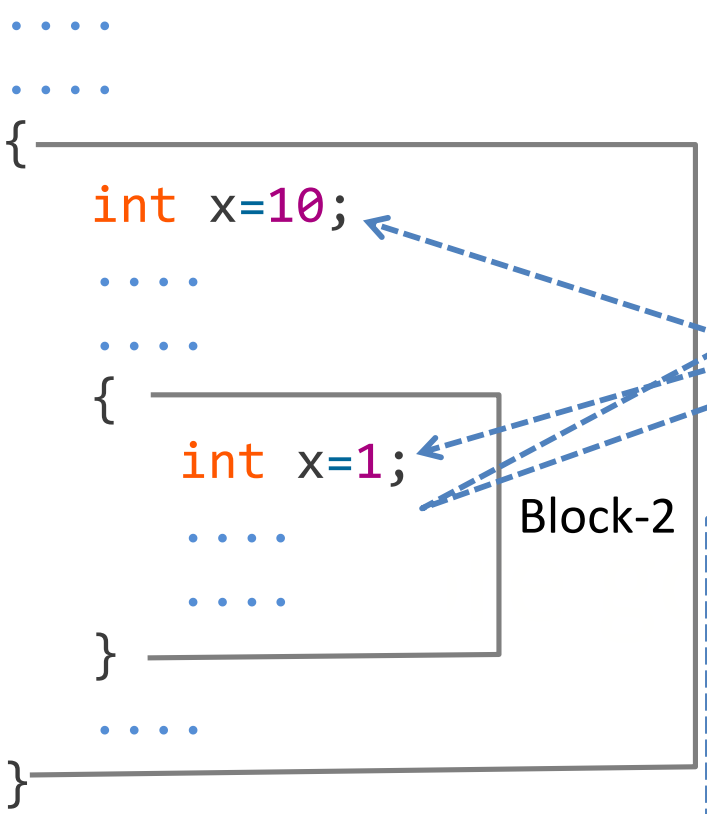
```
Output = 4
right shifting is the equivalent
of dividing a by a power of two
```

New Operators in C++



Scope Resolution Operator

Scope Resolution Operator(::)



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

```
#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

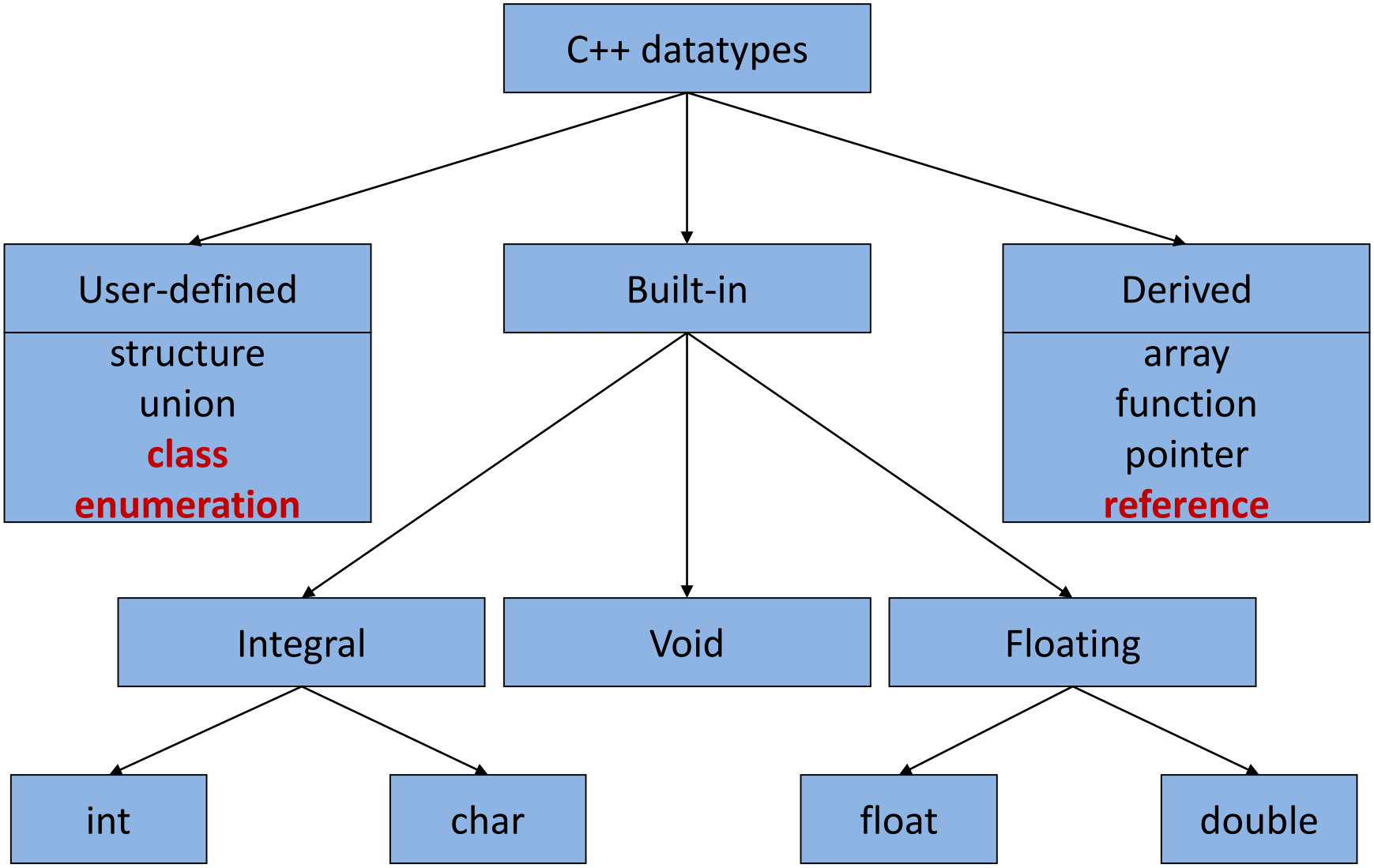
variable m
declared again local to inner block

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



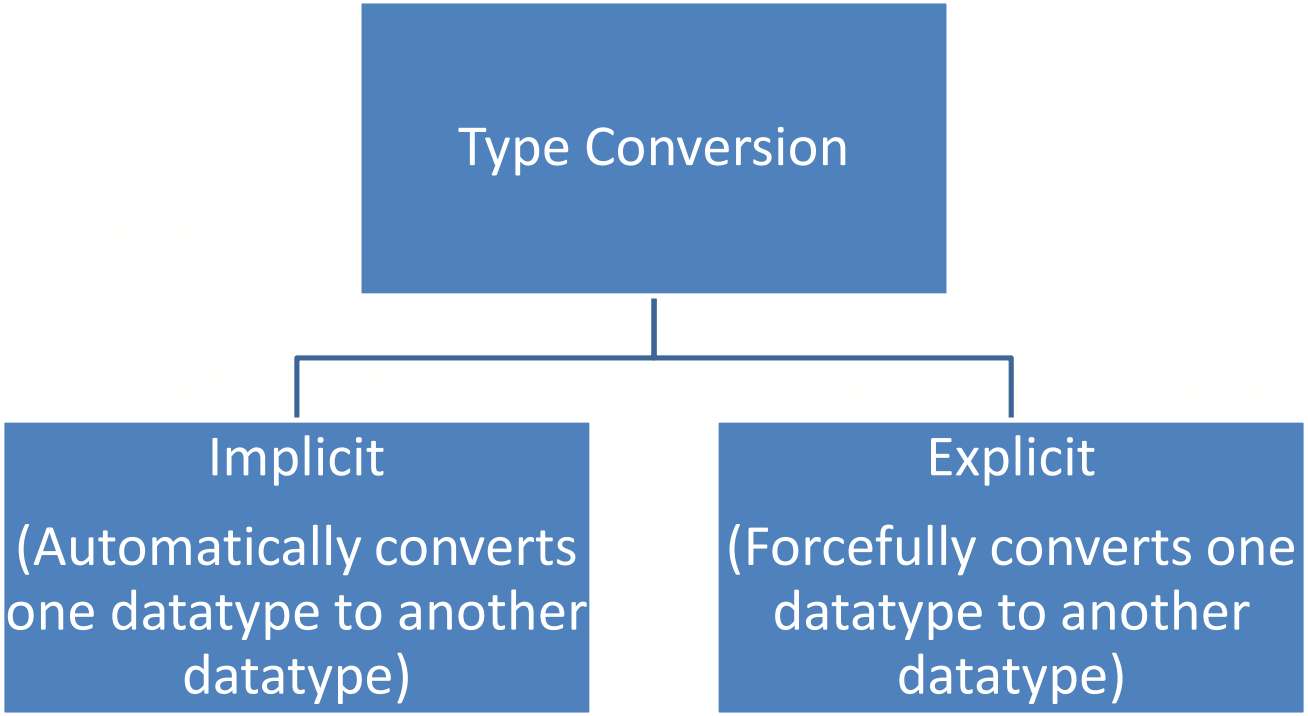
Built in Data types

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.

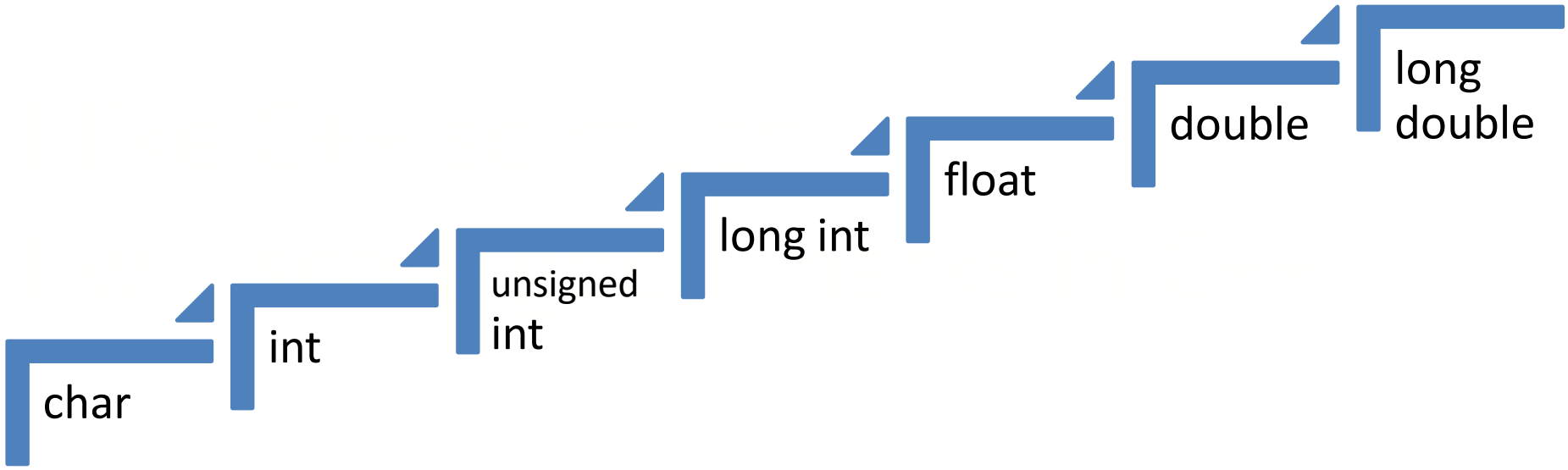


- Explicit type conversion is also known as **type casting**.

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



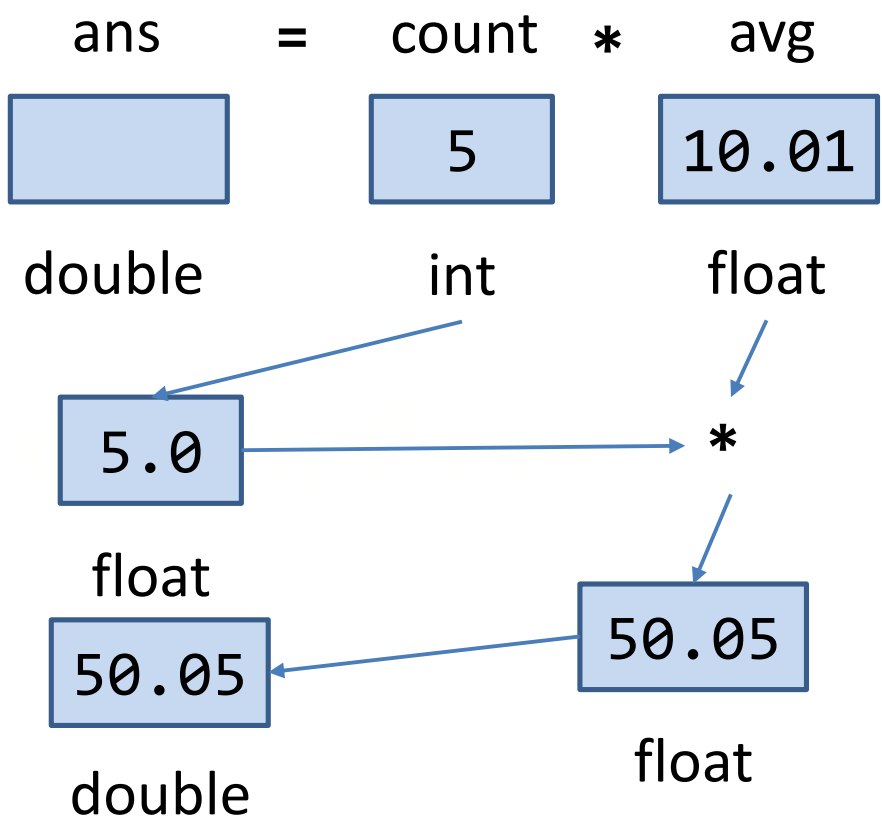
Implicit Type Conversion

```
#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

```
#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;
```

declares variable a

declares ans as reference to 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;
```

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 the 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**.

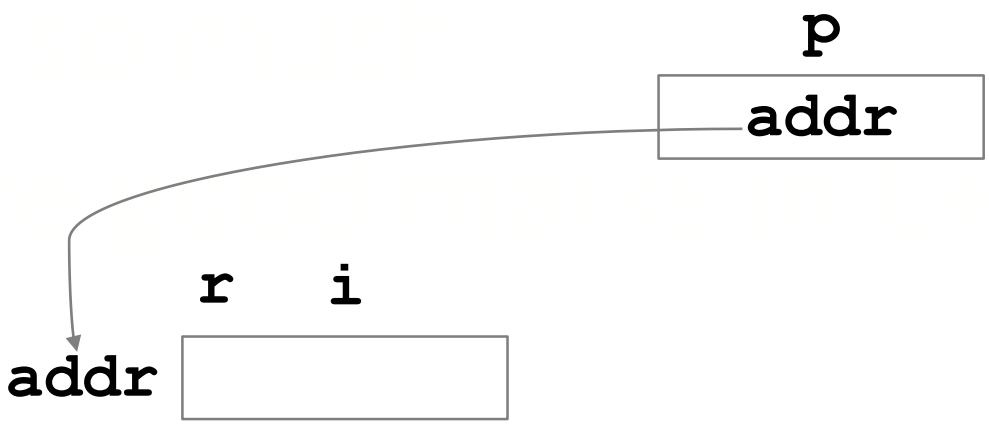
Reference Vs Pointer

References

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

Pointers

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



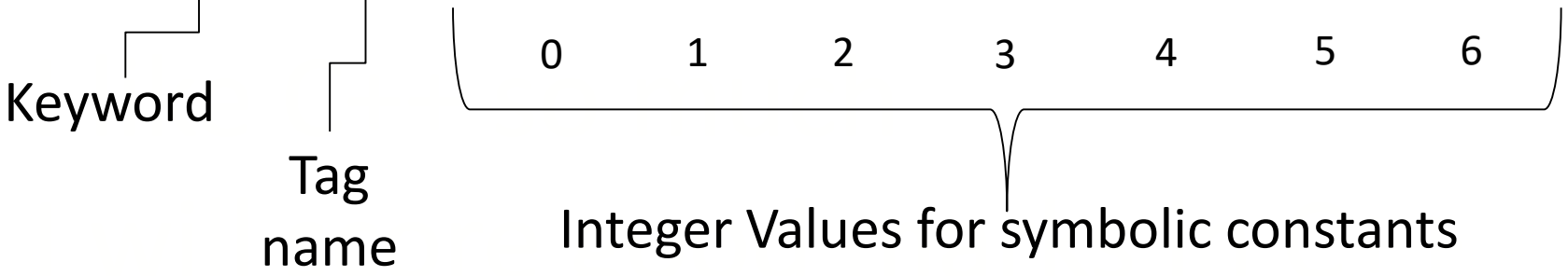
- A reference is a variable which **refers** to another variable.
- A pointer is a variable which **stores the address** of another variable.

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};
```



- 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.

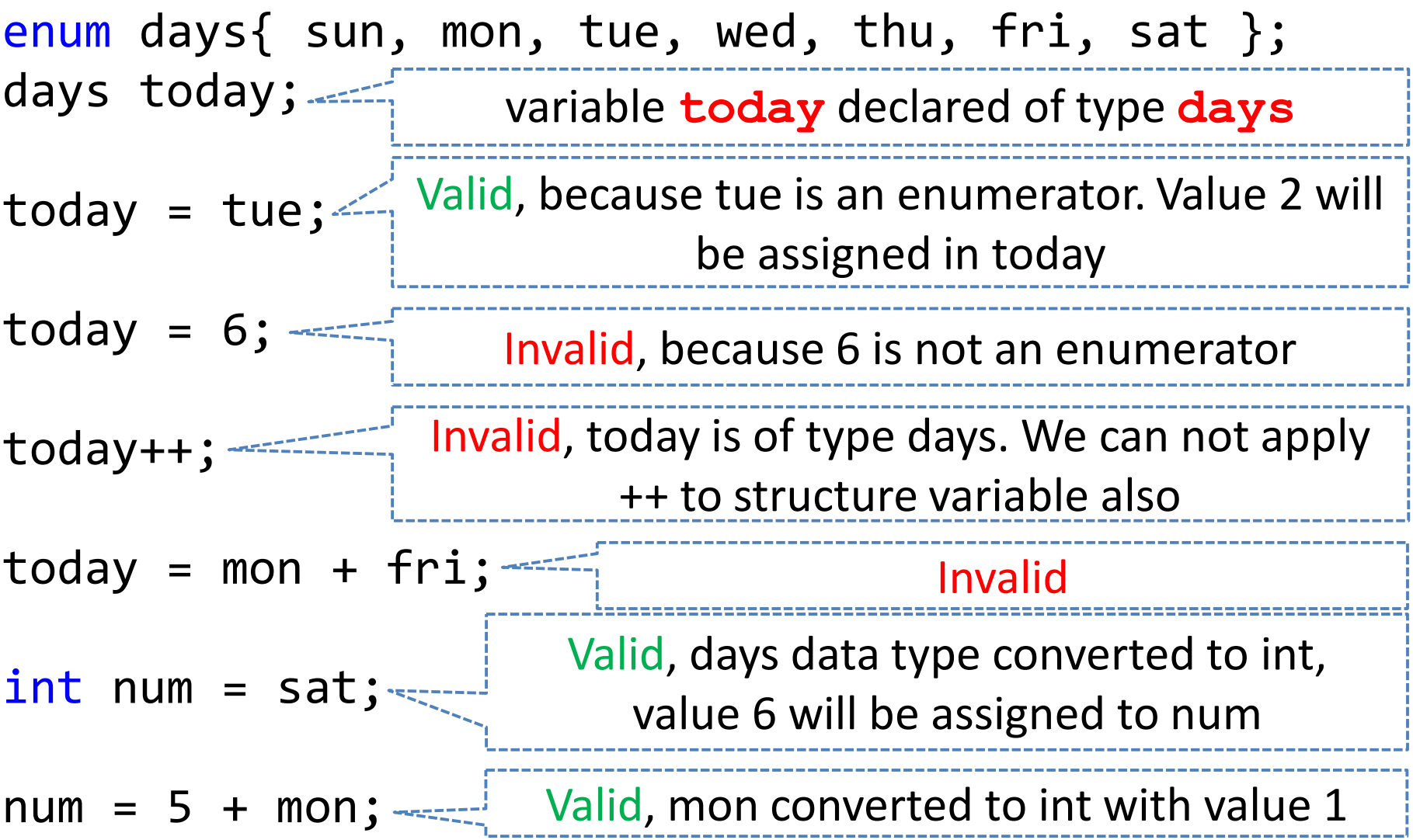
Enumeration Behaviour(Cont…)

```
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



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