Programming Language Concepts

Lecture 3

Dated: 13/02/2024

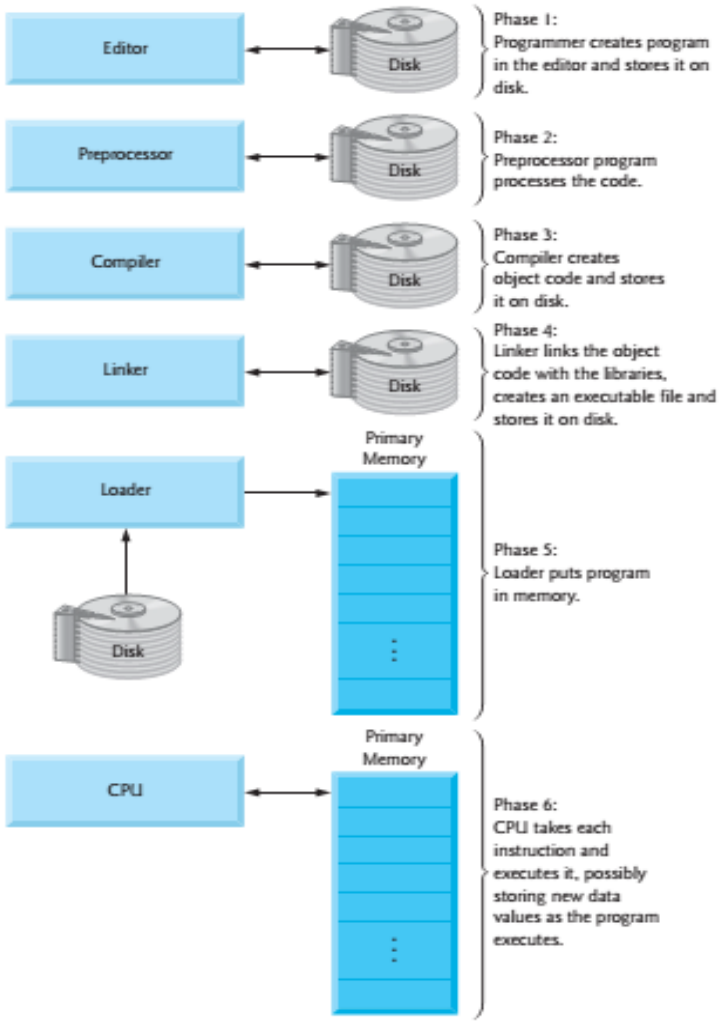
C++ Basics

- C++ programs typically go through six phases.

1. **Phase 1: Creating a Program**
2. **Phase 2: Preprocessing**
3. **Phase 3: Compilation**
4. **Phase 4: Linking**
5. **Phase 5: Loading**
6. **Phase 6: Execution**

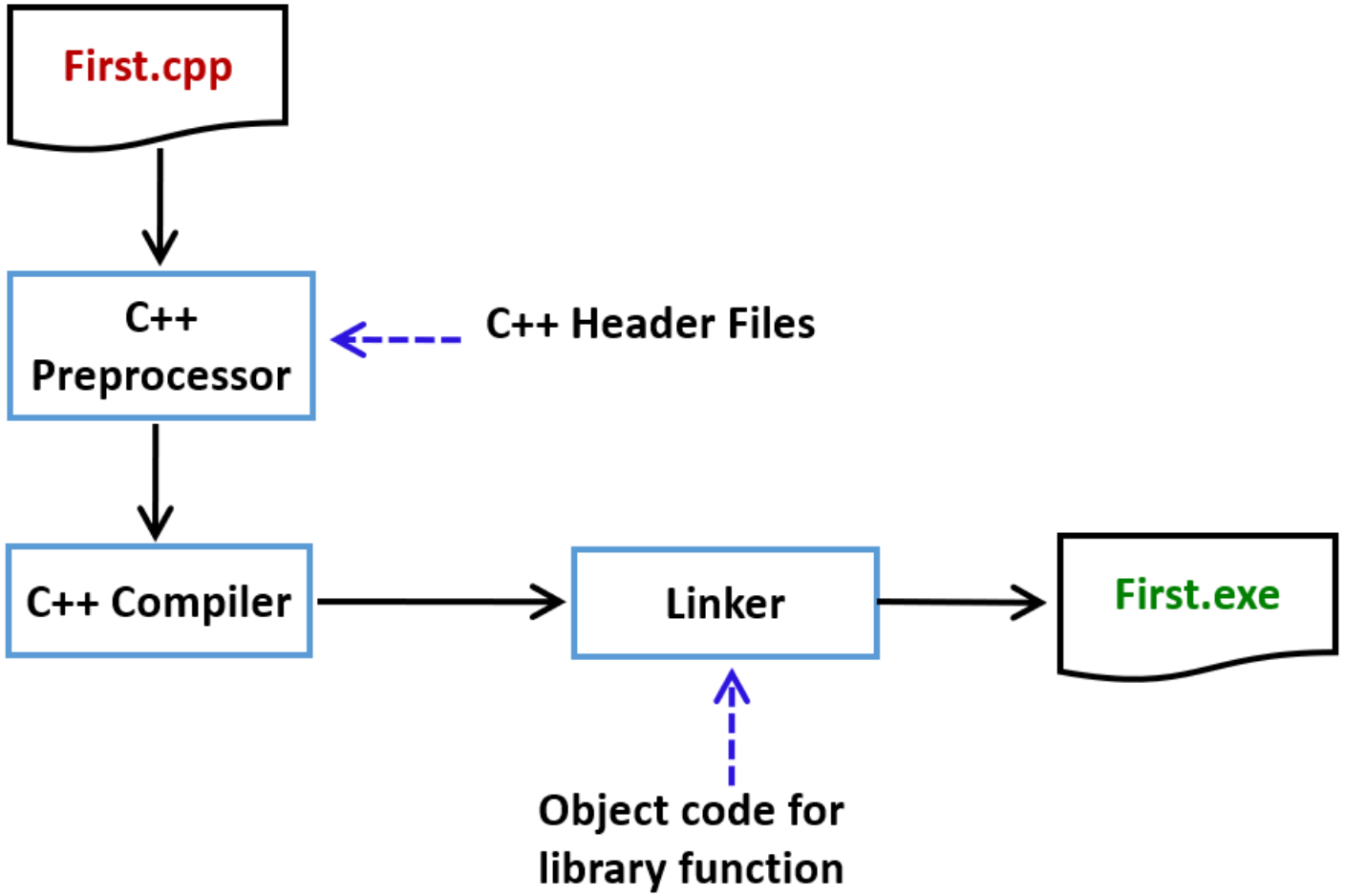
C++ Basics

- Phases in Execution Process of a C++ Program



C++ Basics

- Compilation Steps



C++ Basics

STRUCTURE OF A C++ PROGRAM

```
#include <iostream.h>
#include <string.h>
using namespace std;

void main ( )
{
    string name; //Name of student
    cout<< "Enter you name";
    cin>>name;
     /* Now print hello , and students name */
    cout<< "Hello " << name;
}
```

C++ Basics

- **Preprocessor Directives**

```
#include<iostream>
```

- # is a preprocessor directive.
- The preprocessor runs before the actual compiler and prepares your program for compilation.
- Lines starting with # are directives to preprocessor to perform certain tasks, e.g., ***"include"*** command instructs the preprocessor to add the *iostream* library in this program

C++ Basics

- Header files (Functionality declarations)

(Turbo C++)		(Visual C++)
• #include<iostream.h>	or	#include <iostream>
• #include<stdlib.h>	or	#include<stdlib>
• …		

C++ Basics

std::Prefix

- std::cout<“Hello World”;
- std::cout<<Marks;
- std::cout<<“Hello ”<<name;
- Scope Resolution Operator ::

If no 'using namespace std',
Then use std:: prefix

- std is a namespace, Namespaces ?

```
using namespace std;
  cout<<“Hello World”;
  cout<<Marks;
  cout<<“Hello ”<<name;
```

C++ Basics

- Namespaces

What is namespace pollution?

  - Occurs when building large systems from pieces
  - *Identical globally-visible names clash*
  - How many programs have a “print” function?
  - Very difficult to fix

C++ Basics

- Namespaces

can make our own namespaces
and call it directly within the main (if declared outside)

```
namespace Mine
{
        const float pi = 3.1415925635;
}


void main(){
        float x = 6 + Mine::pi;
        cout<<x;
}
```

like a function in Java

C++ Basics

- Omitting std::Prefix

- *using* directive brings namespaces or its sub-items into current scope

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

void main()
{
      cout<<"Hello World!"<<endl;
      cout<<"Bye!";
}
```

C++ Basics

main() function

- Every C++ program start executing from main ( )
- A function is a construct that contains/encapsulates statements in a block.
- Block starts from “{“ and ends with “}” brace
- Every statement in the block must end with a semicolon ;

C++ Basics

cout and ***endl***

- **cout** (*console output*) and the operator
- **<<** referred to as the stream insertion operator
- **<<** "Inserts values of data-items or string to screen."
- **>>** referred as stream extraction operator, extracts value from stream and puts into "Variables"

- A string must be enclosed in double-quotation marks.

- **endl** stands for *end line*, sending 'endl' to the console outputs a new line

C++ Basics

- **Input and Type**
  - **cin>>name;** reads characters until a whitespace character is seen
  - Whitespace characters:
    - space,
    - tab,
    - newline {*enter key*}

C++ Basics

Variables

- **Variables** are identifiers which represent some unknown, or variable-value.

- A variable is named storage (some memory address's contents)

```
x = a + b;
Speed_Limit = 90;
```

C++ Basics

Variable Declaration

TYPE <Variable Name> ;

Examples:

```
int marks;
double Pi;
int suM;
char grade;
```

- **NOTE:** Variable names are case sensitive in C++ ??

C++ Basics

Variable declaration

- C++ is **case sensitive**
  - Example:

area
Area
AREA
ArEa

are all seen as **different** variables

C++ Basics

Variable Valid Names

- Start with a letter
- Contains letters
- Contains digits
- Contains underscores

- **Do not start names with underscores: _age**

- Don’t use C++ ***Reserve Words***

C++ Basics

- C++ Reserve Words

auto	**break**	**int**	**long**
case	**char**	**register**	**return**
const	**continue**	**short**	**signed**
default	**do**	**Sizeof**	**static**
double	**else**	**struct**	**switch**
enum	**extern**	**typedef**	**union**
float	**for**	**unsigned**	**void**
goto	**if**	**volatile**	**while**

C++ Basics

Variable Names

- Choose meaningful names
  - Don’t use abbreviations and acronyms: mtbf, TLA, myw, nbv

- Don't use overly long names
  - **Ok:**

    partial_sum
    element_count
    staple_partition

  - **Too long (valid but not good practice):**

    remaining_free_slots_in_the_symbol_table

C++ Basics

- Which are legal identifiers?

☑ AREA

☒ 2D

☒ Last Chance

☑ x_yt3

☒ Num-2

☒ Grade***

☑ area_under_the_curve

☑ _Marks

☒ #values

☑ areaoFCirCLe

☒ %done

☒ return

☑ Ifstatement

not recommended

C++ Basics

- **String input (Variables)**

```
// Read first and second name
#include<iostream>
#include<string>
void main() {
  string first;
  string second;
  cout << "Enter your first and second names:";
  cin >> first >> second;
   cout << "Hello " << first << " " << second;
  }
```

C++ Basics

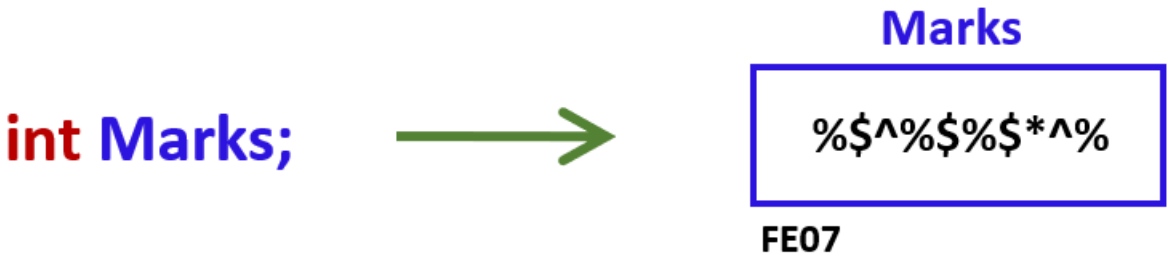
- **Declaring variables**

Before **using** you must **declare** the variables

C++ Basics

Declaring Variables

- When we declare a variable, what happens ?
  - Memory allocation
    - How much memory (*data type*)
  - Memory associated with a name (variable name)
  - The allocated space has a unique **address**

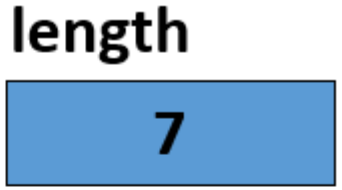


C++ Basics

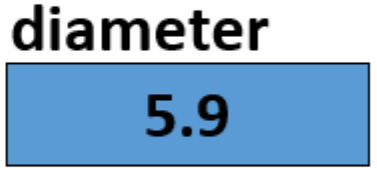
Using Variables: Initialization

- Variables may be given initial values, or **initialized**, when declared. Examples:

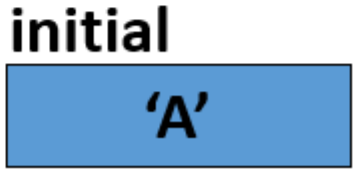
```
int length = 7 ;
```




```
float diameter = 5.9 ;
```




```
char initial = 'A' ;
```



C++ Basics

rvalue and lvalue

- Are the two occurrences of **"a"** in this expression the same?
  - **a = a + 1;**
  - ➢ One on the ***left*** of the assignment refers to the **location** of the **variable** (whose name is a, or address);
  - ➢ One on the ***right*** of the assignment refers to the **value** of the **variable** (whose name is a);

- Two attributes of variables *lvalue* and *rvalue*
  - The ***lvalue*** of a variable is **its address**
  - The ***rvalue*** of a variable is **its value**

C++ Basics

rvalue and lvalue

- Assignment Rule: On the left side of an assignment there must be a *lvalue* or a variable (address of memory location)

```
int i, j;
i = 7;
7 = i;
j * 4 = 7;
```

C++ Basics

Data Types

Three basic PRE-DEFINED data types:

1. To store whole numbers
   - **int, long int, short int, unsigned int**
2. To store real numbers
   - **float, double**
3. To store characters
   - **char**

C++ Basics

- **Built-in types**
  - Boolean type
    - **bool**
  - Character types
    - **char**
  - Integer types
    - **int**
      - **and short** and **long**
  - Floating-point types
    - **double**
      - and **float**

- **Standard-library types**
  - **string**

Literals

- Boolean: **true**, **false**

- Character literals
  - **'a', 'x', '4', '\n', '$'**
- Integer literals
  - **0, 1, 123, -6,**

- Floating point literals
  - **1.2, 13.345, 0.3, -0.54,**

- String literals
  - **"asdf", "Helllo", Pakistan"**

C++ Basics

Types

- C++ provides a set of types
  - E.g. **bool**, **char**, **int**, **double** called **“built-in types”**

- C++ programmers can define new types
  - Called ***“user-defined types”***

- The C++ standard library provides a set of types
  - E.g. **string**, **vector**, **..**
  - (for vector type → #include<vector>)

C++ Basics

C++ Data Types

Data types in C++ are mainly divided into three types:

- Primitive Data Types
- Derived Data Types
- Abstract or User-Defined Data Types

C++ Basics

C++ Data Types

- **Primitive Data Types**

These data types are built-in or predefined data types for example: int, char , float, bool etc. Primitive data types available in C++ are:

- Integer / int
- Character / char
- Boolean / bool
- Floating Point / float
- Double Floating Point / double
- Valueless or Void / void
- Wide Character

C++ Basics

C++ Data Types

- Derived Data Types

The data-types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types. These can be of four types namely:

- Function
- Array
- Pointer
- Reference

C++ Basics

C++ Data Types

- Abstract or User-Defined Data Types

These data types are defined by user itself. Like, defining a class in C++ or a structure. C++ provides the following user-defined datatypes:

- Class
- Structure
- Union
- Enumeration
- Typedef defined DataType

C++ Basics

Type Casting

Numeric Type Conversion

Consider the following statements:

```
short i = 10;
long k = i * 3 + 4;
double d = i * 3.1 + k / 2;
cout<<d;
```

C++ Basics

- **Auto Conversion of Types in C++**

1. If one of the operands is long double, the other is converted into long double

2. Otherwise, if one of the operands is double, the other is converted into double.

3. Otherwise, if one of the operands is unsigned long, the other is converted into unsigned long.

4. Otherwise, if one of the operands is long, the other is converted into long.

5. Otherwise, if one of the operands is unsigned int, the other is converted into unsigned int.

6. Otherwise, both operands are converted into int.

C++ Basics

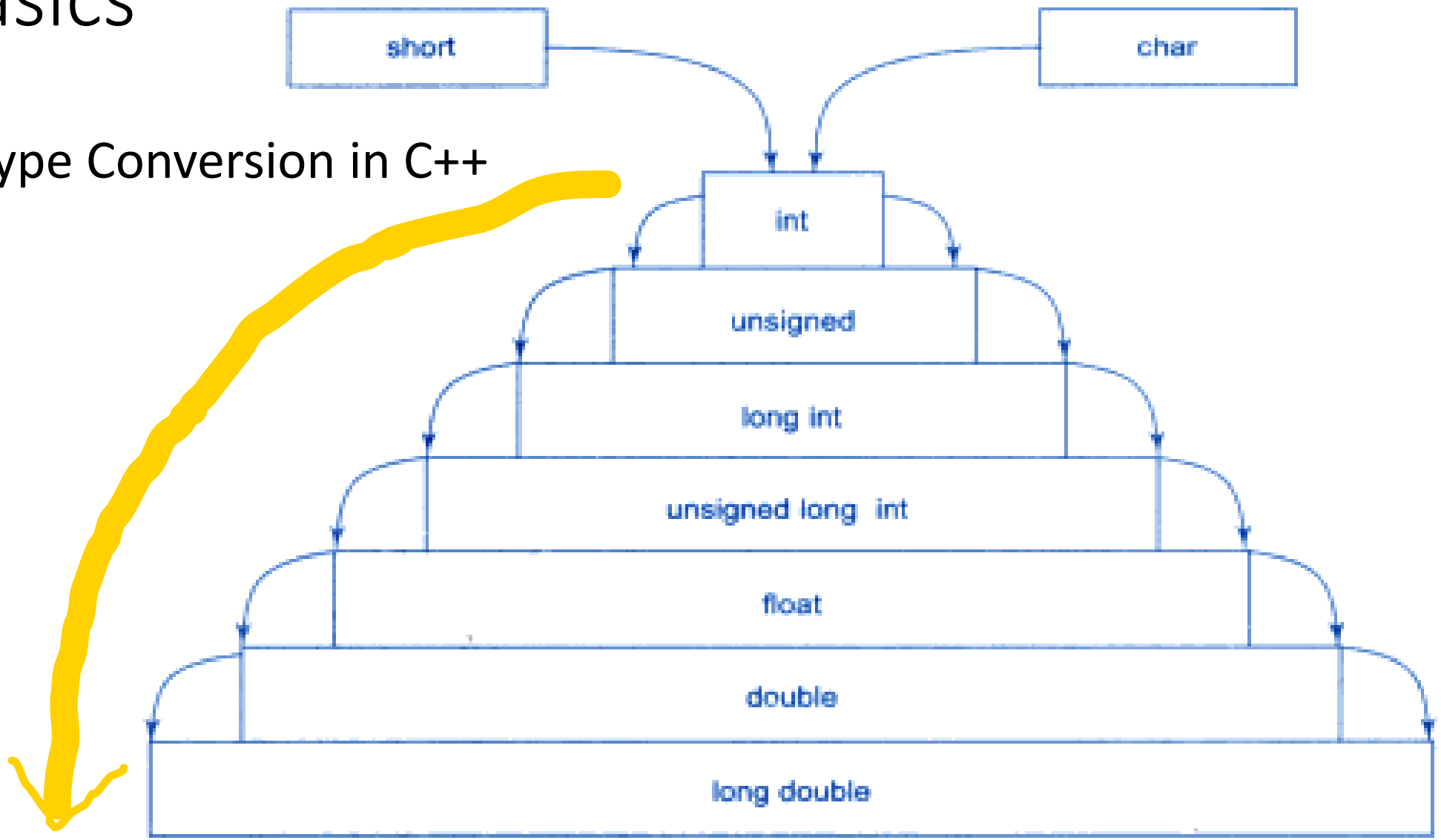
TYPECASTING

- A mechanism by which we can change the data type of a variable (no matter how it was originally defined)

- **Two ways:**
  1. Implicit type casting (done by compiler)
  2. Explicit type casting (done by programmer)

C++ Basics

- Implicit Type Conversion in C++



C++ Basics

- Implicit Type Casting

```
void main()
{
    char c = 'a';
    float f = 5.0;
    float d = c + f;
    cout<<d<<“ ”<<sizeof(d)<<endl;
    cout<<sizeof(c+f);
```

sizeof(d) = 102
sizeof(c+f) = 102

C++ Basics

Explicit Type Casting

- Explicit casting performed by programmer. It is performed by using cast operator

```
float a=5.0, b=2.1;
int  c = a%b;   // → ERROR
```

compiler will not convert float to integer

- Three Styles

```
int c = (int) a  %  (int) b;
int c = int(a)   %   int(b);
int c = static_cast<int>(a) % static_cast<int>(b);
cout<<c;
```

C++ Basics

- Explicit Type Casting

Casting does not change the variable being cast.

For example, **d** is not changed after casting in the following code:

```
double d = 4.5;
int j = (int) d;   //C-type casting
int i = static_cast<int>(d);  // d is not changed
cout<<j<<" "<<d;
```

C++ Basics

- **Widening Type Casting**

A "**widening**" cast is a cast from one type to another, where the "**destination**" type has a **larger range** or **precision** than the "**source**“

Example:

```
int  i = 4;
double d = i;
```

C++ Basics

Narrowing Type Casting

- A “**narrowing**" cast is a cast from one type to another, where the "**destination**" type has a **smaller range** or **precision** than the "**source**“

Example:

```
double d = 787994.5;
int j = (int) d;

// or

int i = static_cast<int>(d);
```

C++ Basics

- Casting between char and numeric types

```
int i = 'a';        // Same as    int i = (int) 'a';

char c = 97;    // Same as    char c = (char)97;
```

C++ Basics

int to string conversion

- C++ style:

```
#include<sstream>

void main()  {
    int val=0;
    stringstream ss;
    cout<<“Enter Value: “;   cin>>val;
    ss << val;     //Using stream insertion op.
   string str_val= ss.str();
   cout<<“\n Output string is: “<<str_val;
}
```

Have a Nice Day