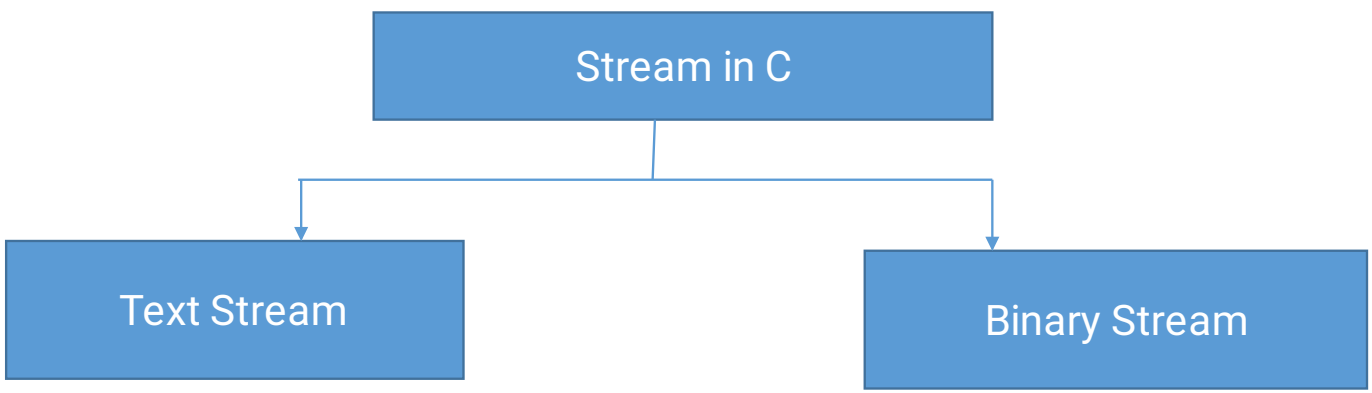
Session-2

Topics covered

- I/O Operation
- Data types
- Variable
- Static
- Constant
- Pointer
- Type conversion

I/O Operation

- C++ I/O operation occurs in streams, which involve transfer of information into byte
- It's a sequences of bytes
- Stream involved in two ways
- It is the source as well as the destination of data
- C++ programs input data and output data from a stream.
- Streams are related with a physical device such as the monitor or with a file stored on the secondary memory.
- In a text stream, the sequence of characters is divided into lines, with each line being terminated.
- With a new-line character (\n) . On the other hand, a binary stream contains data values using their memory representation.



Cascading of Input or Output Operators

- << operator –It can use multiple times in the same line.
- Its called Cascading
- Cout ,Cin can cascaded

- For example
- cout<<“\n Enter the Marks”;
- cin>> ComputerNetworks>>OODP;

Reading and Writing Characters and Strings

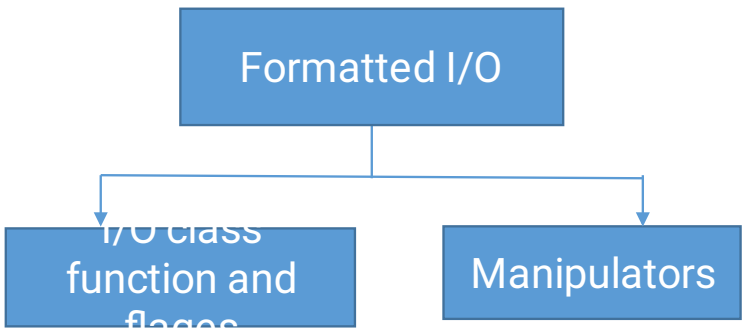
- char marks;
- cin.get(marks);//The value for marks is read
- OR
- marks=cin.get();//A character is read and assigned to marks
- string name;
- Cin>>name;

- string empname;
- cin.getline(empname,20);
- Cout<<“\n Welcome ,”<<empname;

Formatted Input and Output Operations

Table 2.7 Functions for Input/Output

Function	Purpose	Syntax	Usage	Result	Comment
width()	Specifies field size (no. of columns) to display the value	cout. width(w)	cout.width(6);cout<<1239;	1 2 3 9	It can specify field width for only one value
precision()	Specifies no. of digits to be displayed after the decimal point of a float value	cout. precision(d)	cout. precision(3);cout<<1.234567;	1.234	It retains the setting in effect until it is reset
fill()	Specify a character to fill the unused portion of a field	cout.fill(ch)	Cout.fill('#');cout. width(6);cout<<1239;	# # 1 2 3 9	It retains the setting in effect until it is reset




```
#include<iostream.h>
#define PI 3.14159
main()
{
cout.precision(3);
cout.width(10);
cout.fill('*');
cout<<PI;
Output
*****3.142
```

Formatting with flags

- The setf() is a member function of the ios class that is used to set flags for formatting output.
- **syntax -cout.setf(flag, bit-field)**
- Here, flag defined in the ios class specifies how the output should be formatted and bit-field is a constant (defined in ios) that identifies the group to which the formatting flag belongs to.
- There are two types of setf()—one that takes both flag and bit-fields and the other that takes only the flag .

Table 2.8 Types of setf()

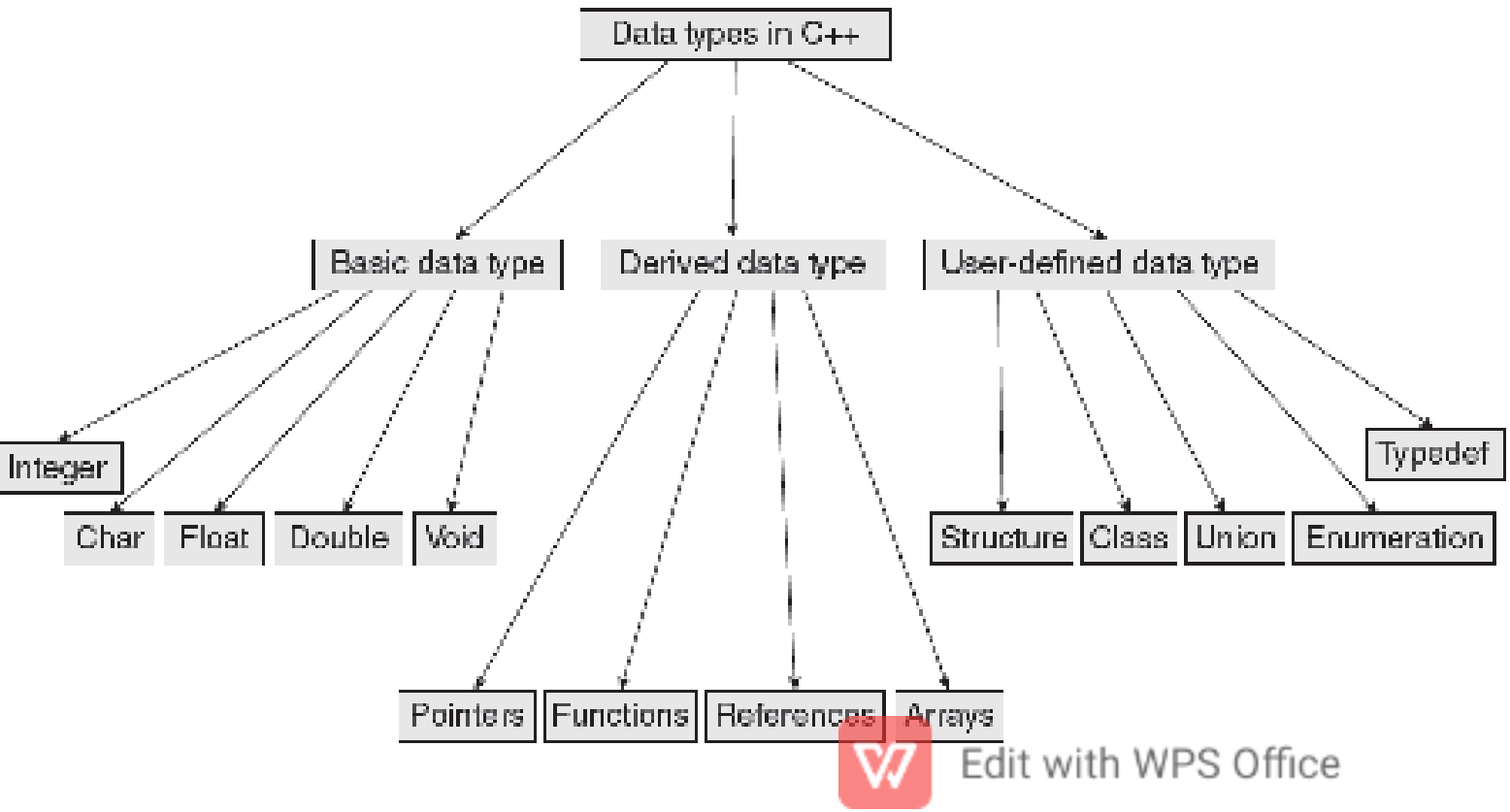
Flag	Bit-field	Usage	Purpose	Comment
ios :: left	ios :: adjustifield	cout.setf(ios :: left, ios :: adjustifield)	For left justified output	If no flag is set, then by default, the output will be right justified.
ios :: right	ios:: adjustifield	cout.setf(ios :: right, ios :: adjustifield)	For right justified output	
Ios :: internal	ios:: adjustifield	cout.setf(ios :: internal, ios :: adjustifield)	Left-justify sign or base indicator, and right-justify rest of number	
ios :: scientific	ios:: floatifield	cout.setf(ios :: scientific, ios :: floatifield)	For scientific notation	If no flag is set then any of the two notations can be used for floating point numbers depending on the decimal point
ios :: fixed	ios :: floatifield	cout.setf(ios :: fixed, ios :: floatifield)	For fixed point notation	
ios :: dec	ios :: basefield	cout.setf(ios :: dec, ios :: basefield)	Displays integer in decimal	If no flag is set, then the output will be in decimal.
ios :: oct	ios :: basefield	cout.setf(ios :: oct, ios :: basefield)	Displays integer in octal	
ios :: hex	ios :: basefield	cout.setf(ios :: hex, ios :: basefield)	Displays integer in hexadecimal	
ios :: showbase	Not applicable	cout.setf(ios :: showbase)	Show base when displaying integers	
ios :: showpoint	Not applicable	cout.setf(ios :: showpoint)	Show decimal point when displaying floating point numbers	
ios :: showpos	Not applicable	cout.setf(ios :: showpos)	Show + sign when displaying positive integers	
ios :: uppercase	Not applicable	cout.setf(ios :: uppercase)	Use uppercase letter when displaying hexadecimal (OX) or exponential numbers (E)	

Formatting Output Using Manipulators

- C++ has a header file iomanip.h that contains certain manipulators to format the output

Manipulator	Purpose	Usage	Result	Alternative
setw(w)	Specifies field size (number of columns) to display the value	cout<<setw(6) <<1239;	␣ ␣ 1 2 3 9	width()
setprecision(d)	Specifies number of digits to be displayed after the decimal point of a float value	cout<<setprecision(3) <<1.234567;	1.235	precision()
setfill(c)	Specify a character to fill the unused portion of a field	cout<<setfill('#') <<setwidth(6) <<1239;	# # 1 2 3 9	fill()

Data Types in C++



Data type	Size in bytes	Range
Char	1	−128 to 127
unsigned char	1	0 to 255
signed char	1	−128 to 127
Int	2	−32768 to 32767
unsigned int	2	0 to 65535
signed short int	2	−32768 to 32767
signed int	2	−32768 to 32767
short int	2	−32768 to 32767
unsigned short int	2	0 to 65535
long int	4	−2147483648 to 214748364
unsigned long int	4	0 to 4294967295
signed long int	4	−2147483648 to 214748364
Float	4	3.4E-38 to 3.4E+38
Double	8	1.7E-308 to 1.7E+308
long double	10	3.4E-4932 to 1.1E+4932

Variables

A variable is the content of a memory location that stores a certain value. A variable is identified or denoted by a variable name. The variable name is a sequence of one or more letters, digits or underscore, for example: character_

Rules for defining variable name:

- A variable name can have one or more letters or digits or underscore for example character_.
- White space, punctuation symbols or other characters are not permitted to denote variable name.
- A variable name must begin with a letter.
- Variable names cannot be keywords or any reserved words of the C++ programming language.
- Data C++ is a case-sensitive language. Variable names written in capital letters differ from variable names with the same name but written in small letters.

01 Local Variables

02 Static Variables

03 Instance variables

04 Final Variables

Variables

Local Variables

Local variable: These are the variables which are declared within the method of a class.

Example:

```
public class Car {
public:
 void display(int m){  // Method
        int model=m;
// Created a local variable model
        cout<<model;
   }
```

Instance Variables

Instance variable: These are the variables which are declared in a class but outside a method, constructor or any block.

Example:

```
public class Car {
     private: String color;
 // Created an instance variable color
 Car(String c)
  {
   color=c;
  }}
```


Static Variables

Static variables: Static variables are also called as class variables. These variables have only one copy that is shared by all the different objects in a class.

Example:

```
public class Car {
     public static int tyres;
// Created a class variable
void init(){
        tyres=4;
}}
```

Constant Variables

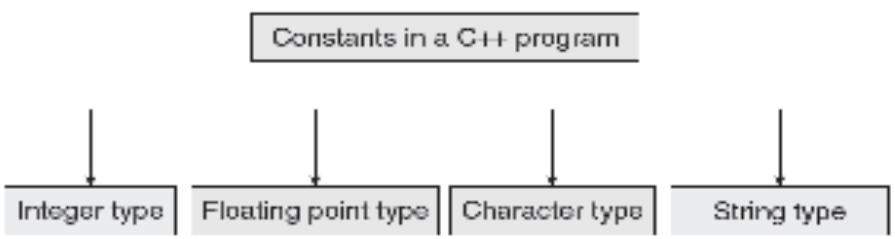
Constant is something that doesn't change. In C language and C++ we use the keyword const to make program elements constant.

Example:

```
const int i = 10;
void f(const int i)
class Test
{
    const int i;
};
```

CONSTANTS

- Constants are identifiers whose value does not change. While variables can change their value at any time, constants can never change their value.
- Constants are used to define fixed values such as Pi or the charge on an electron so that their value does not get changed in the program even by mistake.
- A constant is an explicit data value specified by the programmer.
- The value of the constant is known to the compiler at the compile time.



Declaring Constants

- Rule 1 Constant names are usually written in capital letters to visually distinguish them from other variable names which are normally written in lower case characters.
- Rule 2 No blank spaces are permitted in between the # symbol and define keyword.
- Rule 3 Blank space must be used between #define and constant name and between constant name and constant value.
- Rule 4 #define is a preprocessor compiler directive and not end with a semi-

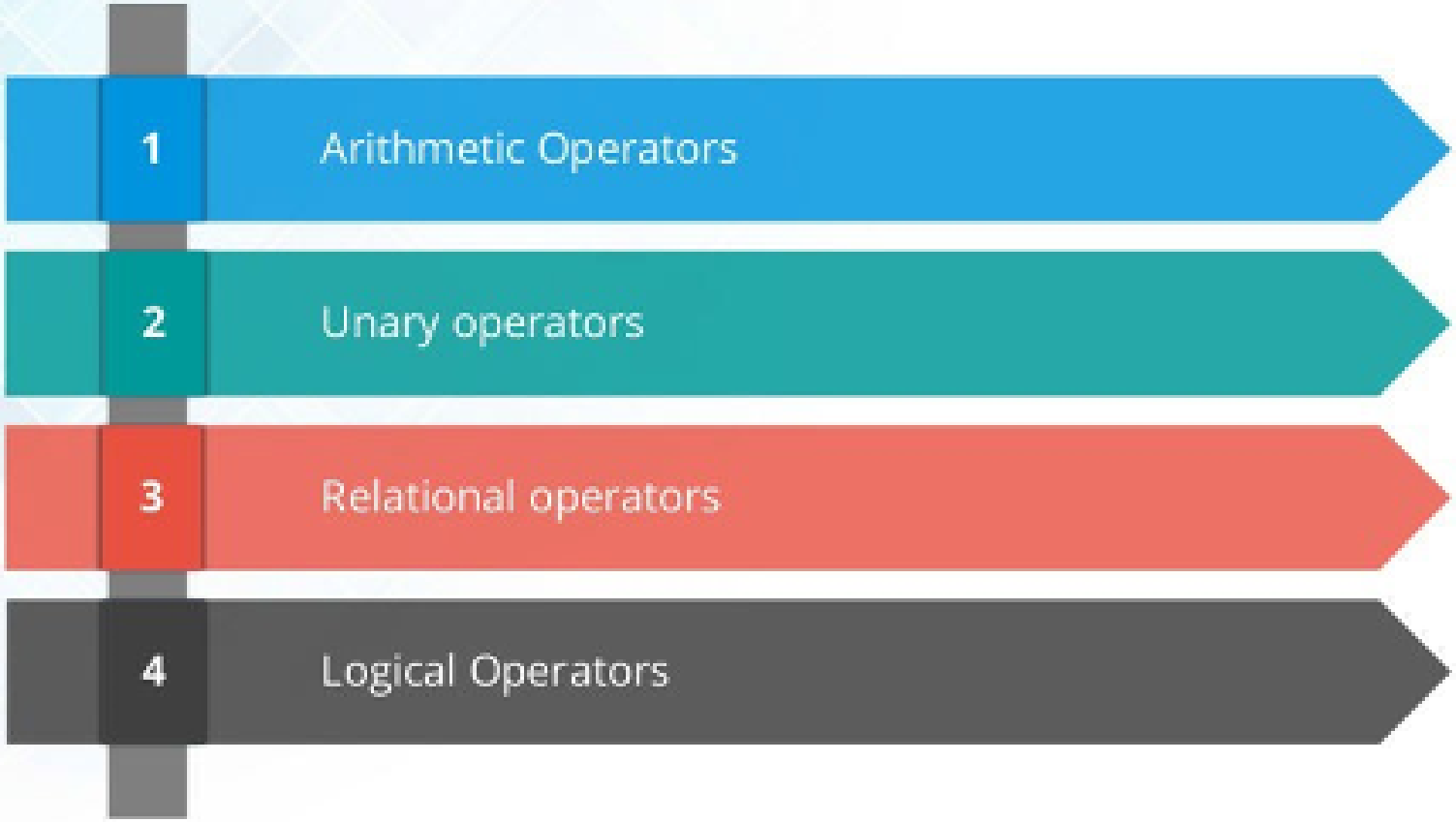
```
const data type const name = value;
```

The const keyword specifies that the value of pi cannot change.

```
const float pi = 4.14;
```

```
#define PI 3.14159
#define service tax 0.12
```

Data Operators



Arithmetic Operators

1. Arithmetic Operators
2. Unary operators
3. Relational operators:
4. Logical Operators

+	Add two operands or unary plus	>> 2 + 3 5 >> +2
-	Subtract two operands or unary subtract	>> 3 – 1 2 >> -2
*	Multiply two operands	>> 2 * 3 6
/	Divide left operand with the right and result is in float	>> 6 / 3 2.0
%	Remainder of the division of left operand by the right	>> 5 % 3 2

Unary operators

1. Arithmetic Operators
2. Unary operators
3. Relational operators:
4. Logical Operators

++

X++ - Post Increment
++X – Pre Increment

```
>> x = 5
>> ++x
>> print(x)
6
```

--

X-- - Post Decrement
--X – Pre Increment

```
>> x--
>> print(x)
4
```

Relational operators

1. Arithmetic Operators
2. Unary operators
3. Relational operators:
4. Logical Operators

>	True if left operand is greater than the right	>> 2 > 3 False
<	True if left operand is less than the right	>> 2 < 3 True
==	True if left operand is equal to right	>>2 == 2 True
!=	True if left operand is not equal to the right	>> x >>= 2 >>print(x) 1

Logical operators

1. Arithmetic Operators
2. Unary operators
3. Relational operators:
4. Logical Operators

and

Returns True if both x and y are True, False otherwise

```
>> True
&& False
False
```

or

Returns True if at least x or y are True, False otherwise

```
>> True ||
False
True
```

not

Returns True if x is False, True otherwise

```
>> ! 1
False
```

Special Operators

Scope resolution operator

1. In C, the global version of a variable cannot be accessed from within the inner block. C++ resolves this problem by using scope resolution operator (::), because this operator allows access to the global version of a variable.

New Operator

2. The new operator denotes a request for memory allocation on the Heap. If sufficient memory is available, new operator initializes the memory and returns the address of the newly allocated and initialized memory to the pointer variable.

Delete Operator

3. Since it is programmer's responsibility to deallocate dynamically allocated memory, programmers are provided delete operator by C++ language.

Member Operator

4. C++ permits us to define a class containing various types of data & functions as members. To access a member using a pointer in the object & a pointer to the member.

Operator Precedence

Rank	Operators	Associativity
1	++, –, +, -, !(logical complement) ,~(bitwise complement) ,(cast)	Right
2	*(mul) , /(div), %(mod)	Left
3	+ , - (Binary)	Left
4	>> , <<, >>> (Shift operator)	Left
5	>, <, >=,<=	Left
6	==, !=	Left
7	& (Bitwise and)	Left
8	^ (XOR)	Left
9	| (Bitwise OR)	Left
10	&& (Logical and)	Left
11	|| (Logical OR)	Left
12	Conditional	Right
13	Shorthand Assignment	Right

Pointers

- Pointer is variable in C++
- It holds the address of another variable
- Syntax data_type *pointer_variable;
- Example int *p,sum;

Assignment

- integer type pointer can hold the address of another int variable
- To assign the address of variable to pointer-**ampersand symbol** (&)
- p=∑

How to use it

- P=∑//assign address of another variable
- cout<<∑ //to print the address of variable
- cout<<p;//print the value of variable
- Example of pointer

```
#include<iostream.h>
using namespace std;
int main()
{ int *p,sum=10;
p=&sum;
cout<<“Address of sum:”<<&sum<<endl;
cout<<“Address of sum:”<<p<<endl;
cou<<“Address of p:”<<&p<<endl;
cout<<“Value of sum”<<*p;
}
```

Output:
Address of sum : 0X77712
Address of sum: 0x77712
Address of p: 0x77717
Value of sum: 10

Pointers and Arrays

- assigning the address of array to pointer don't use ampersand sign(&)

```
#include <iostream>
using namespace std;
int main(){
  //Pointer declaration
  int *p;
  //Array declaration
  int arr[]={1, 2, 3, 4, 5, 6};
  //Assignment
  p = arr;
  for(int i=0; i<6;i++){
    cout<<*p<<endl;
    //++ moves the pointer to next int position
    p++;
  }
  return 0;
}
```

```
OUTPUT:
0
1
2
3
4
5
6
```

This Pointers

- this pointer hold the adderss of current object
- int num;
- This->num=num;

Function using pointers

```
#include<iostream>
using namespace std;
void swap(int *a ,int *b );
//Call By Reference
int main()
{
  int p,q;
 cout<<"\nEnter Two Number You Want To Swap \n";
cin>>p>>q;
swap(&p,&q);
cout<<"\nAfter Swapping Numbers Are Given below\n\n";
cout<<p<<"   "<<q<<" \n";
    return 0;
}
```



```
void swap(int *a,int *b)
{
int c;
c=*a;
*a=*b;
*b=c;
}
```

Output:

```
Enter Two Number You Want to Swap
10 20
After Swapping Numbers Are Given
below
20 10
```

Type conversion and type casting

- Type conversion or typecasting of variables refers to changing a variable of one data type into another. While type conversion is done implicitly, casting has to be done explicitly by the programmer.

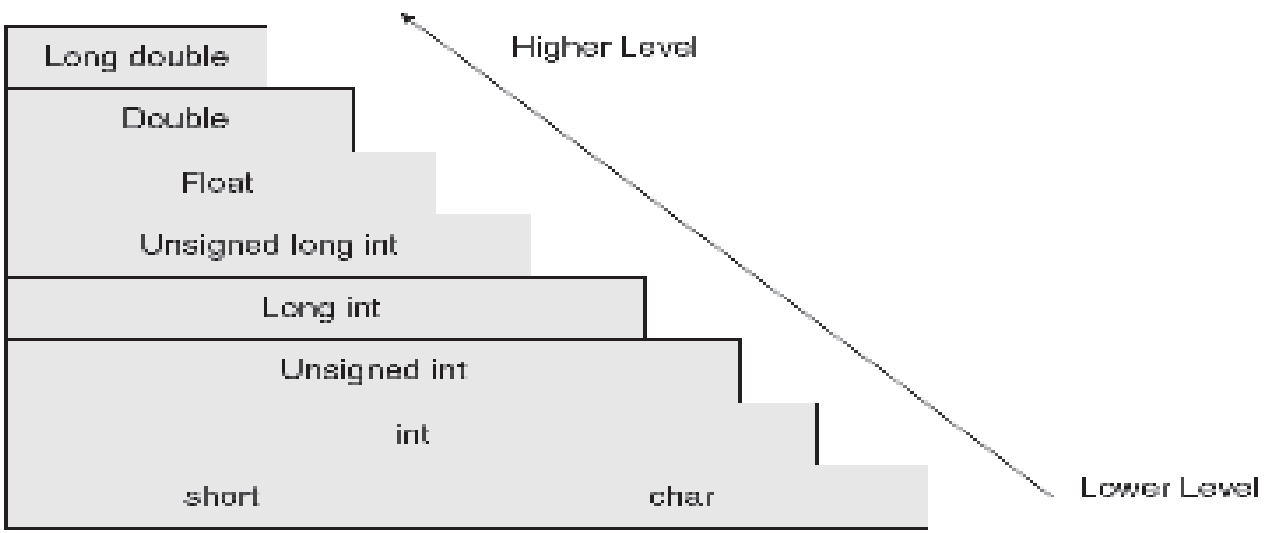


Figure 2.13 Conversion hierarchy of data types

Type Casting

- Type casting an arithmetic expression tells the compiler to represent the value of the expression in a certain format.
- It is done when the value of a higher data type has to be converted into the value of a lower data type.
- However, this cast is under the programmer's control and not under the compiler's control. The general syntax for type casting is destination_variable_name=destination_data_ty--pe(source_variable_name) ;

```
float sal=10000.00;
int income;
Income=int(sal);
```