CS201 FINAL TERM NOTES

LECTURE 23 TO 45

PROVIDE BY ORANGE MONKEY TEAM

Lecher number # 1

What is programming?

` Computer programming is a medium for us to communicate with computers. Just like we use "Urdu" or "English" to communicate with each other, programming is a way for us to deliver our instructions to the computer.

What is C programming?

C is a programming language. C is one of the oldest and finest programming language. C was developed by the Dennis Ritchie of AT & T"S Bell labs USA in 1971.

The BCPL language was developed in 1967 by Martin Richards as a language for writing operating systems software and compilers.

prayers remember me

What is C++?

C++ is an update version of C. It is also called C with close. It is an object and oriented language. It was developed by at AT& T"S Bell lobs

Uses of C and C++

C and C++ language is used to program a wide variety of system. Some of the uses of the C and C ++ are as follows:

- Major parts of windows, Linux and operating system are written in C.
- C and C++ are used to write drive programs for devices like tablets, printers etc.
- C and C++ language is used to program embedded system where properly need to run faster in limited memory (Microwave, Camera etc.).
- C and C++ used to develop fames an area where latency is very important. I.e. Computer has to react quickly on user input.

Variables, Constant, and Keywords Variables:

A variable is a constant which stores " value"

For example

In kitchen, we have containers storing Rice, Dal, Sugar, etc. Similar to that variable in C stores value of a constant.

Means that

```
a = 3;        // a is assigned "3"
b = 4.7;      // b is assigned "4.7"
c = „A";      // c is assigned "A"
```

Rules for naming variables in C

First character must be an alphabet or underscore (_) No commas, blanks allowed.

No special symbol other than (_) allowed. Variables names are case sensitive.

We must create meaning variables names in our programs. This enhances read easily in our programs constant.

An entity whose value does not change is called as a constant.

A variable is an entity whose value can be changed.

Lecher number # 02

Categories of software

- System Software
- Application Software

System software

System software is a type of computer program that is designed to run a computer's hardware and application programs

Sub categories of system software are:

- Operating system
- Device drivers

prayers remember me

- Utilities

What is operating system?

Operating System controls the basic operations of the computer. It prepares the computer to provide an environment in which user communicates with the computer.

What is a Device Driver?

We will see that the operating system has installed special software to control these devices. This piece of software is called device driver software.

TWAIN stands for Technol
Utility software is software designed to help to analyze, configure, optimize or maintain a computer.

Application Software
Application software (app for short) is computing software designed to carry out a specific task other than one relating to the operation of the computer itself, typically to be used by end-users.

What is an Editor?

First of all we need a tool for writing the code of a program. For this purpose we used Editors in which we write our code.

What are Compiler and Interpreter? Or Difference b/w Compiler and Interpreter?

*Interpreter***The drawback of the interpreter is that the program executes slowly as the interpreter translates the program line by line. Another drawback is that as interpreters are reading the program line by line so they cannot get the overall picture of the program hence cannot optimize the program making it efficient.**

Compiler **Compilers also translate the English like language (Code written in C) into a language. The Compiler read the whole program and translates it into machine language completely. The difference between interpreter and compiler is that compiler will stop translating if it finds an error.**

IDES stand for (Integrated Development Environment)

These IDEs contain editor, compilers, debugger, linker and loader. The benefit of an IDE is that we have all these things at the same place

prayers remember me

made by orange monkey team 👍

Editor ↔ Disk

Preprocessor ↔ Disk — Preprocessor program processes the code.

Compiler ↔ Disk — Compiler creates object code and stores it on disk.

Linker ↔ Disk — Linker links the object code with the libraries

Primary Memory

Loader ← Disk — Loader puts program in memory.

Primary Memory

CPU — CPU takes each instruction and executes it, possibly storing new data values as the program executes.

made by orange monkey team 👍

Lecture No # 3

So here is our very first program in C.

Program shape

```
1 #include<iostream>
2 using namespace std;
3 int main ()
4 {
5
6
7
8         return 0;
9}
```

***#include*: This is a pre-processor directive. The sign # is known as HASH and also called SHARP.**

***Next*, there is a curly bracket also called braces ("{ }").**

There is a semicolon (;) at the end of the above statement.

What is cout?

This is known as output stream in C and C++. cout takes data from computer and sends it to the output. The sign <<indicates the direction of data.

The thing between the double quotes ("") is known as character string.

What isVariables?

During programming we need to store data.

prayers remember me

In a program every variable has

1. Name 2. Type 3. Size 4. Value

What is Assignment Operator?

In C language equal-to-sign (=) is used as assignment operator.
Example

Where as in C language X = 2 (where X is a variable name)

Define Data Types?

A variable must have a data type associated with it, for example it can have data types like integer, decimal number characters etc.

1. Define int Data Type?

The data type int is used to store whole numbers (integers). Space of 4 bytes in memory. In programming before using any variable name we have to declare that variable with its data type. Example

int x ; x = 5; **The declaration statement int i ; reserves 4 bytes of memory**

Define short Data type/ long Data Type?

Short Data type

The C provides another data type for storing small whole numbers which is called short. The size of short is two bytes and it can store numbers in range of -32768 to 32767.

Long Data Type

A very large whole number that cannot be stored in an int then we use the data type long provided by C. for example Long x = 300500200;

2. Define Float Data Type?

To store real numbers, float data type is used. The float data type uses four bytes to store a real number.

Float x; X=35.36;

3. Define char Data Type?

prayers remember me

For storing the character data C language provides char data type. Used around the character as ‘a’.

made by orange monkey team 👍

Lecture #4

What is cin?

cin is the counter part of the cout. Here cin is the input stream that gets data form the user and assigns it to the variable on its right side. We know that the sign >>.

Program

```
1. #include<iostream>

2. Using namespace std;

3. int main ()

4 .{      // declaration of variables, the age will be in whole numbers int age1, age2, age3, age4, age5, age6, age7, age8, age9, age10; int Total Age, Average Age;

5.   // take ages of the students from the user

6   .  cout<< “Please enter the age of student 1: ”;

7.  cin>> age1;

8.   cout<< “Please enter the age of student 2: ”;

9. cin>> age2;

10. cout<< “Please enter the age of student 3: ”;

11. cin>> age3;

12. cout<< “Please enter the age of student 4: ”;

13. cin>> age4;
```

prayers remember me

14. cout<< “Please enter the age of student 5: ”;

15. cin>> age5;

16. cout<< “Please enter the age of student 6: ”;

17. cin>> age6;

18. cout<< “Please enter the age of student 7: ”;

19. cin>> age7;

20. cout<< “Please enter the age of student 8: ”;

22. cin>> age8;

23. cout<< “Please enter the age of student 9: ”;

24. cin>> age9;

25. cout<< “Please enter the age of student 10: ”;

26. cin>> age10;

27. // calculate the total age and average age Total Age = age1 + age2 + age3 + age4 + age5 + age6 + age7 + age8 + age9 +age10;

28. Average Age = Total Age / 10;

29. // Display the result (average age)

30. cout<< “Average age of class is: “ << Average Age;

return 0; }

avers remember me

A sample output of the above program is given below.

Please enter the age of student 1: 12

Please enter the age of student 2: 13

Please enter the age of student 3: 11

Please enter the age of student 4: 14

Please enter the age of student 5: 13

Please enter the age of student 6: 15

Please enter the age of student 7: 12

Please enter the age of student 8: 13

Please enter the age of student 10: 11

Lecture No. 5

We have ages of two students (say for the time being we have got these ages in variables). These variables are- age1 and age2. Now we say that if the age1 is greater than age2, then display the statement 'Student 1 is older than student 2'. The coding for this program will be as below

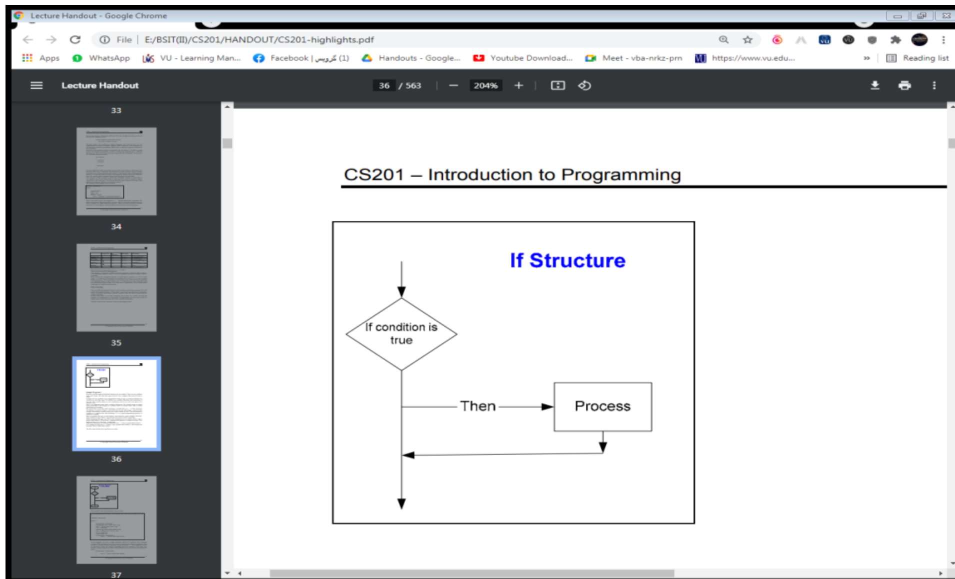
```
#include <iostream>

using namespace std;

int main()

{

int age1, age2;

age1 = 12;

age2 = 10;

if (age1 > age2)

cout<<"Student 1 is older than student 2";

}
```

What is Flow Charting?

A flow chart helps in correctly designing the program by visually showing the sequence of instructions to be executed.

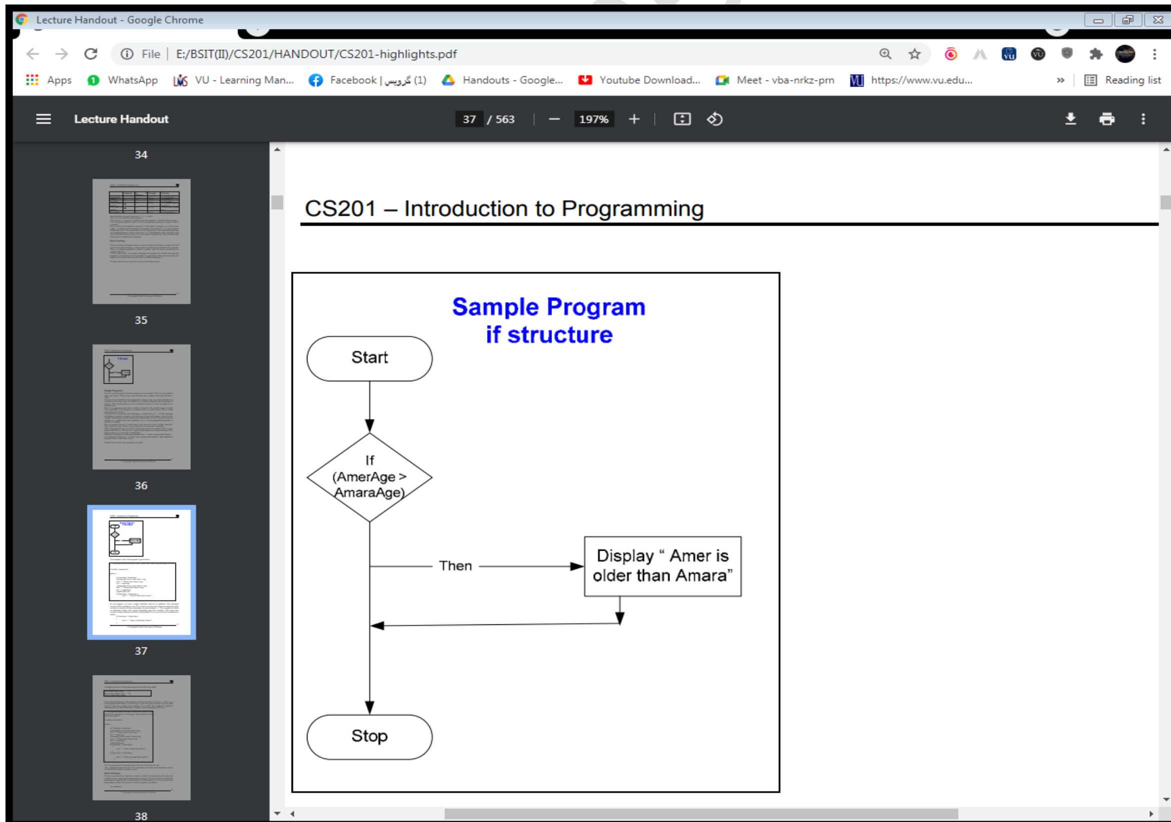
The flow chart for the if structure is shown in the figure below.

prayers remember me



Sample Program if stretcher

There are two students Amar and Amara. We take their ages from the user, compare them and tell who is older?



#include <iostream>

```
using namespace std;

int main()

{

int AmerAge, AmaraAge;

//prompt the user to enter Amer's age

cout<<"Please enter Amer's age";

cin>>AmerAge;

//prompt the user to enter Amara's age

cout<<"Please enter Amara's age";

cin>>AmaraAge;

//preform the test

if (AmerAge>AmaraAge)

cout<< "Amer is older than Amara";

}
```

Define Logical Operators?

In programming we use logical operators (&& and ||) for AND and OR respectively with relational operators. These are binary operators and take two operands.

<table>
<tr><th>Expression 1</th><th>Expression 2</th><th>Expression1 && Expression 2</th><th>Expression 1 || Expression 2</th></tr>
<tr><td>T</td><td>T</td><td>F</td><td>t</td></tr>
<tr><td>T</td><td>F</td><td>T</td><td>t</td></tr>
<tr><td>F</td><td>T</td><td>F</td><td>F</td></tr>
<tr><td>F</td><td>F</td><td>f</td><td>T</td></tr>
</table>

The truth table for the logical negation operator (!) is given below.

Expression	! Expression

prayers remember me

True	False
False	True

made by orange monkey team

made by orange monkey team👍

Lecture No. 6

What is while loop?

In while loop, condition is evaluated first and if it returns true then the statements inside while loop execute, this happens repeatedly until the condition returns false. When condition returns false, the control comes out of loop and jumps to the next statement in the program after while loop.

'While' is also a key word of 'C

While means, 'do it until the condition is true'

The syntax of while construct is as under

While (Logical Expression)

{

statement1;

statement2;

…………

}

----------------Program-----------------

```
#include <iostream>

using namespace std;

int main()

{

        int num1;

        int num2;
```

prayers remember me

```
cout<<"plz enter 1st number";
cin>>num1;
cout<<"plz enter 2st number";
cin>>num2;
if (num1 > num2)
{
cout<<"the number is grater";
}
else if (num1 < num2)
{
cout<<"the number is less";
}
else
{
cout<<"the numbers are equll";
}
        return 0;
}
```

prayers remember me

while block.

Flow Chart:

The basic structure of while loop in structured flow chart is:

While Loop

While

Is Loop Condition true?

No

Exit

Yes

Processes

At first, we will draw a rectangle and write while in it. Then

and use the decision symbol i.e. diamond diagram. Write the

Always use the self-explanatory variable names

Practice a lot. Practice makes a man perfect

While loop may execute zero or more time
Make sure that loop test (condition) has an acceptable exit.

Lecture No. 7

What is Do-While Statement?

The C do while statement creates a structured loop that executes as long as a specified condition is true at the end of each pass through the loop. The syntax for a do while statement is: ... If the value of the expression is "false" (i.e., compares equal to zero) the loop is exited

The syntax of do-while do

```
{
 Statement;
}
```

While (condition);

Define For Loop?

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

There are three things in a loop structure i.e.

(i) **initialization,**
(ii) **a continuation/termination condition and**
(iii) **Changing the value of the condition variable, usually the increment of the variable value.**

The syntaxof for loop is given below.

```
For (initialization condition; continuation condition;incrementing condition)
{
Statement;
}
```

Imp

Comments should be meaningful, explaining the task

Don't forget to affect the value of loop variable in while and do-while loops

Make sure that the loop is not an infinite loop

Don't affect the value of loop variable in the body of for loop, the for loop does this by itself in the form statement.

Use pre and post increment/decrement operators cautiously in expressions.

prayers remember me

Lecture No. 8

Switch Statement(break ;)

Sometimes, we have multiple conditions and take some action according to each condition.

To avoid this expensiveness, an alternate of multiple if statements can be used that is if/else statements. We can write an if statement in the body of an if statement which is known as nested if. We can write the previous code of if statements in the following nested if/else form.

The syntax of switch statement is as follows.

Switch (variable/expression)

{

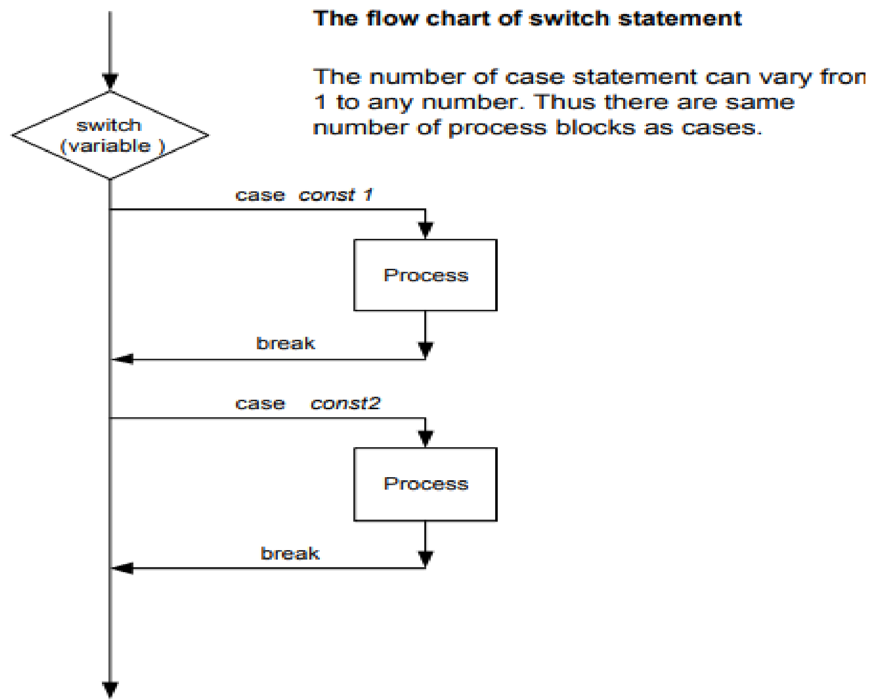
case constant1 : statementLlist1 ;

case constant2 : statementLlist2 ;

case constantN : statementListN ;

default : statementList ;

}



remember me

goto Statement

The goto is an unconditional branch of execution

```
#include <iostream>

#include <conio.h>

using namespace std;

int ShowMatrix()

{

int day = 4;

switch (day) {

  case 1:

cout<< "Monday";

    break;

  case 2:

cout<< "Tuesday";

    break;

  case 3:

cout<< "Wednesday";

    break;

  case 4:

cout<< "Thursday";

    break;

  case 5:
```

prayers remember me

```
cout<< "Friday";
   break;
 case 6:
cout<< "Saturday";
   break;
 case 7:
cout<< "Sunday";
   break;
}
}
// Outputs "Thursday" (day 4)
```

Lecture No. 9 Introduction back lecher

Introduction

We have a main ( ) in every C program. 'Main ( )' is also afunction.

Structure of a Function

```
{
Return 0
}
```

return-value_type:(short question or quiz)

The keyword is return which is used to return some value from the function.There may be some functions which do not return any value. For such functions, the return_value_type is void. 'Void' is a keyword of 'C' language. The default return_value_type is of int data type i.e. if we do not mention any return_value_type with a function; it will return an int value.

Calling Mechanism:

For now, let us call the function swap() by passing values by reference as in the following example −

```
#include <iostream>
```

prayers remember me

```
using namespace std;

// function declaration

void swap(int &x, int &y);

int main () {

   // local variable declaration:

   int a = 100;

   int b = 200;

cout<< "Before swap, value of a :" << a <<endl;

cout<< "Before swap, value of b :" << b <<endl;

  /* calling a function to swap the values using variable reference.*/

 swap(a, b);

cout<< "After swap, value of a :" << a <<endl;

cout<< "After swap, value of b :" << b <<endl;

 return 0;

}
```

Sample Program

C is called function-oriented language. It is a very small language but there are lots of functions in it. Function can be on a single line, a page or as complex as we want.

Problem statement:

Calculate the integer power of some number (x^n).

```
#include <iostream>


using namespace std;


int main()
```

prayers remember me

```
{
   int base, exp, i, result = 1;

cout<< "Enter base and exponent\n";
cin>> base >> exp;

   // Calculate base^exponent by repetitively multiplying base
   for(i = 0; i< exp; i++){
      result = result * base;
   }

cout<< base << "^" << exp << " = " << result;

   return 0;
}
```

made by orange monkey team

Lecture No. 10

Header files

Header files contain definitions of Functions and Variables, which is imported or used into any C++ program by using the pre-processor #include statement. ... h" which contains C++ function declaration and macro definition. Each header file contains information (or declarations) for a particular group of functions.

Scope of identifiers

Scope of an Identifier. Any identifier used in a C++ program (such as the name of a variable or object, the name of a type or class, or the name of a named constant) has a scope, i.e., a region of the program in which that name can be used.

prayers remember me

Suppose we write the function:

```
#include <iostream>


using namespace std;


const double RATE = 10.50;

int z;

double t;


void one(int x, char y);

void two(int a, int b, char x);

void three(int one, double y, int z);


int main()

{

        int num, first;

        double x, y, z;

        char name, last;

        return 0;

}


Void one (int x, char y)

{

}
```

prayers remember me

```
int w;


Void two (int a, int b, char x)
{
}


Void three (int one, double y, int z)
{
        char ch;
        int a;
        {
                int x;
                char a;
        }
}
```

What is function?

It will be risky to tell the address of some variables to the called function. Also, see the code above for some special arrangements for call by reference in C language.

What is Recursive Function?

A recursive function is a function that calls itself during its execution. The process may repeat several times, outputting the result and the end of each iteration. ... The result could be used as a roundabout way to subtract the number from 10.

```
Long fact (long n)
{
 If (n <= 1)
     return 1;
```

```
else
 return n* fact(n-1);
}
```

- **Lecture No. 11**

What is Arrays?

Arrays are a special data-type. If we have a collection of data of same type as in the case.IN C language every array has a data type i.e. name and size. Data type can be any valid data type.

Declaration:

The declaration of arrays is as follows:

data_typearray_name [size] ;

For example:

int ages[10];

What is subscript?

In C language, the index of array starts from zero and is one less than array's size. Index of array is also called subscript.

Programing

```
int i, age [10];
for ( i = 0; i< 10 ; i++ )
{
Age[i]=0;
}
```

With the help of this simple loop, we have initialized all the elements of array age to zero.

Sample program

prayers remember me

For example, take an integer array 'n'.

Solution:

```
 #include <iostream>
int main()
{

        using namespace std;

        int marks[3];
        float average;
        cout<< "Enter marks of first student" <<endl;
        cin>> marks[0];
        cout<< "Enter marks of second student" <<endl;
        cin>> marks[1];
        cout<< "Enter marks of third student" <<endl;
        cin>> marks[2];

        average = ( marks[0] + marks[1] + marks[2] )/ 3.0;
        cout<< "Average marks : " << average <<endl;

        return 0;
}
```

Linear Search

linear search (Searching algorithm) which is used to find whether a given number is present in an array and if it is present then at what location it occurs. It is also known as sequential search.

prayers remember me

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

int main() {
cout<<"Enter The Size Of Array:   ";
int size;
cin>>size;



int array[size], key,i;

// Taking Input In Array
 for(int j=0;j<size;j++){
cout<<"Enter "<<j<<" Element: ";
cin>>array[j];
 }

//Your Entered Array Is
 for(int a=0;a<size;a++){
cout<<"array[ "<<a<<" ]  =  ";
cout<<array[a]<<endl;
 }

cout<<"Enter Key To Search  in Array";
cin>>key;
```

prayers remember me

```
  for(i=0;i<size;i++){
     if(key==array[i]){
cout<<"Key Found At Index Number :  "<<i<<endl;
 break;
   }
 }



if(i != size){
cout<<"KEY FOUND at index :  "<<i;
}
else{
cout<<"KEY NOT FOUND in Array  ";
}
  return 0;
}
```

The Keyword 'const':

The keyword const can be used with any data type and is written before the data type as:

int age [arraySize];

Now in the loop condition, we can write like this:

For (i = 0; i<arraySize; i++)

- **Lecture No. 12**

Character Arrays

prayers remember me

An array is an indexed collection of data elements of the same type.
1) Indexed means that the array elements are numbered (starting at 0).
2) The restriction of the same type is an important one, because arrays are stored in consecutive memory cells. Every cell must be the same type (and therefore, the same size).

That marker is a special character, Called null character. The ASCII code of null character is all zeros.

In C language, we represent the null character as "\0". C uses this character to terminate a string.

While the twelfth is the null character inserted automatically at the end of the string.

Can someone please explain to me why the output from the following code is saying that arrays are not equal?

- **Lecture No. 13**

What is array manipulation?

.Arrays are sequences of numerical data, with each element having the same underlying data type – integers, real (floating point) numbers, or complex numbers. These arrays can be multi-dimensional, have their dimensionality change dynamically, a can be sliced (subsets of elements taken).

Lecture No. 23 Deitel & Deitel - C++ How to Program

Preprocessor

Being a concise language, C needs something for its enhancement. So a preprocessor is used to enhance it. We have so far been using *#include* preprocessor directive like *#include<iostream>*.

What actually *#include* does?

When we write *#include<somefile>*, this *somefile* is ordinary text file of C code. The line where we write the *#include* statement is replaced by the text of that file. We can't see that file included in our source code. However, when the compiler starts its work, it sees all the things in the file. Almost all of the preprocessor directives start with # sign. We have included iostream.h, stdlib.h, fstream.h, string.h and Here 'h' stands for header files

prayers remember me

Include directive

If the first directive is #include, the compiler will search this file in the include directory. This directive is read by the preprocessor and orders it to insert the content of a user-defined or system header file into the following program.

Define Directive:

We can define macros with the *#define* directive. Macro is a special name, which is substituted in the code by its definition, and as a result, we get an expanded code.

For example,

We are writing a program, using the constant Pi. Pi is a universal constant and has a value of 3.1415926. We have to write this value 3.1415926 wherever needed in the program. It will be better to define Pi somewhere and use Pi instead of the actual value

#Define Pi 3.1415926

Program Example

```
#include <iostream>
#define PI 3.1415926              // Defining PI
Int main()
{
   int radius = 5;
cout << "Area of circle with radius " << radius << " = " << PI * radius * radius;}
```

Macros:

A preprocessor feature that supports parameter substitution and expansion of commonly used code sequences. Also see inline function.
Macros are classified into two categories. The first type of macros can be written using *#define*. The value of PI can be defined as:

```
#define PI 3.1415926
```

In contrast, the second type of macros takes arguments. *It is also called parameterized macros.* Consider the following:

```
#define square () x * x
```

Program Example:

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

#define square(x) ((x) * (x))    //Definition of macro square   //Inline Function(To be discussed)
```

```
int main()

{
   int x;
cout << " Please enter the value of x to calculate its square ";

cin >> x;

cout << " Square of x = " << square(x) ;

cout << " Square of x+2 = " << square(x+2) ;

cout << " Square of 7 = " << square(7);

}
```

Lecture No. 24   Deitel & Deitel - C++ How to Program

Memory Allocation

Memory allocation is the process of setting aside sections of memory in a program to be used to store variables, and instances of structures and classes. There are two basic types of memory allocation: When you declare a variable or an instance of a structure or class.

Dynamic Memory Allocation

In static memory, when we write the things like int i, j, k ; these reserve a space for three integers in memory. Similarly the typing of char s[20] will result in the allocation of space for 20 characters in the memory. This type of memory allocation is static allocation. It is also known as compile time allocation. This memory allocation is defined at the time when we write the program while exacting knowing how much memory is required.
There is another part of memory, called heap. The dynamic memory allocation uses memory form the heap.

1) Calloc Function:

The syntax of the calloc function is as follows:

```
void *calloc (size_t n, size_t el_size)
```

This function takes two arguments.

- The first argument is the required space in terms of numbers.

- Second one is the size of the space.

So we can say that we require n elements of type int. We have read a function sizeof. This is useful in the cases where we want to write a code that is independent of the particular machines that we are running on.

So if we write like void *calloc(1000, sizeof(int))

It will return a memory chunk from the heap of 1000 integers. By using sizeof (int) we are not concerned with the size of the integer on our machine whether it is of 4 bytes or 8 bytes. We will get automatically a chunk that can hold 1000 integers.

2) Malloc Function:

The malloc function takes one argument i.e. the number of bytes to be allocated.

The syntax of the function is:

```
void *malloc (size_t size) ;
```

It returns a void pointer to the starting of the chunk of the memory allocated from the heap in case of the availability of that memory. If the memory is not available or is fragmented (not in a sequence), malloc will return a NULL pointer. While using malloc, we normally make use sizeof operator and a call to malloc function is written in the following way:

```
malloc (1000 * sizeof(int)) ;
```

Programing example

```
//This program calculates the average age of a class of students
//using dynamic memory allocation

#include <iostream>
using namespace std;
int main( )
{
 int numStuds, i, totalAge, *iptr, *sptr;

 cout <<"How many students are in the class ? " ;
 cin >> numStuds;

 // get the starting address of the allocated memory in pointer iptr
```

```
iptr = (int *) malloc(numStuds * sizeof(int));

if (iptr == NULL)      //Check for the success of memory allocation

{
cout << "Unable to allocat space for " << numStuds << " students\n";

return 1;

// A nonzero return is usually used to indicate an error

}

sptr = iptr ; //sptr will be used for pointer arithmetic/manipulation

i = 1 ;

totalAge = 0 ;

//use a loop to get the ages of students
for (i = 1 ; i <= numStuds ; i ++)

{

cout << "Enter the age of student " << i << " = " ;

cin >> *sptr ;

totalAge = totalAge + *sptr ;

sptr ++ ;

}
```

```
cout << "The average age of the class is " << totalAge / numStuds << endl;

//now free the allocated memory, that was pointed by iptr

free (iptr) ;

sptr = NULL ;
}
```

3. Realloc Function:

Sometimes, we have allocated a memory space for our use by malloc function. But we see later that some additional memory is required.

For example, in the previous example, where (for example) after allocating a memory for 35 students, we wanted to add one more student. So we need same type of memory to store the new entry. Now the question arises ‘Is there a way to increase the size of already allocated memory chunk ? Can the same chunk be increased or not? The answer is yes. In such situations, we can reallocate the same memory with a new size according to our requirement. The function that reallocates the memory is realloc.

The syntax of realloc is given as:

```
void realloc (void * ptr, size_t size ) ;
```

Suppose we have allocated a memory for 20 integers by the following call of malloc and a pointer iptr points to the allocated memory.

```
(iptr *) malloc (20 * sizeof(int)) ;
```

Now we want to reallocate the memory so that we can store 25 integers. We can reallocate the same memory by the following call of realloc.

```
Realloc (iptr, 25 * sizeof(int)) ;
```

Memory leak:

When we use a raw pointers in our program, we are actually allocating memory for it in the heap. This memory space in the heap remains occupied until the object in the heap is deleted.

If we don’t do so and the pointer is destroyed, the space in the memory remains occupied, causing a **memory leak**.

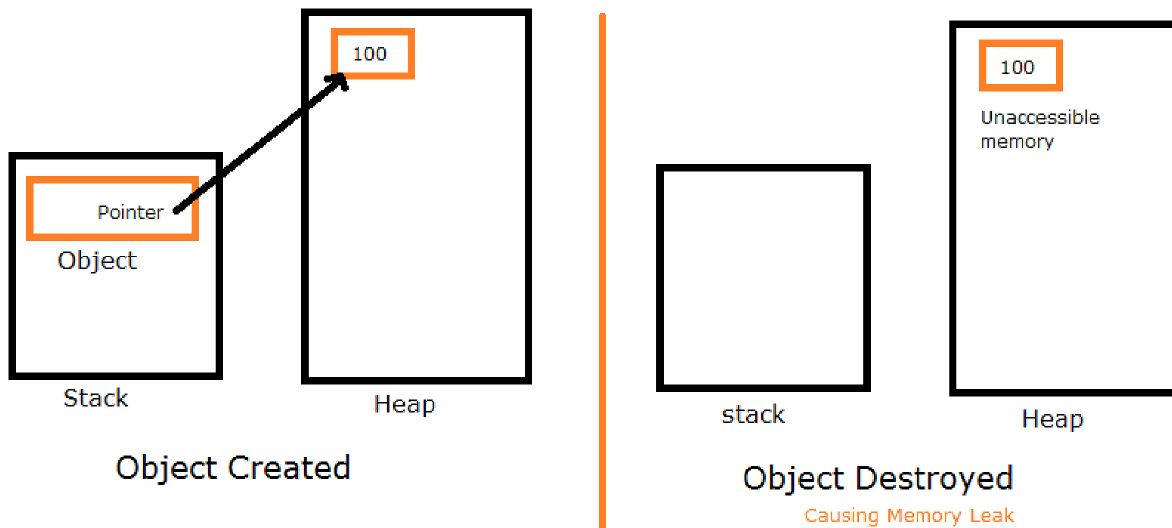
For example:

Suppose, we have a class in our C++ program that has a raw pointer as one of its properties:

```
main()
```

```
{
int *p;
   Myclass (int n){
       p = new int {n};  //Allocating space in heap, needs to be free when not used
   }
};
```

Sometimes the programmer fails to initialize the pointer with a valid address, then this type of initialized pointer is known as a dangling pointer in C. Dangling pointer occurs at the time of the object destruction when **the object is deleted** or de-allocated from memory without modifying the value of the pointer.



What are Dangling Pointers?

Sometimes the programmer fails to initialize the pointer with a valid address, then this type of initialized pointer is known as a dangling pointer in C. Dangling pointer occurs at the time of the object destruction when the object is deleted or de-allocated from memory without modifying the value of the pointer.

But what's to be noted is that the memory which we've used for the pointer is still containing data but can no longer be accessed (because the object is destroyed). This inaccessible memory is termed as a **leak of memory**.

Lecture No. 25

Structured programming:

- In structured programming a problem is broken into small pieces or modules and each small piece corresponds to a function.
- This is the top-down structured programming approach.

Default Function argument:

While writing and calling functions, you might have noticed that sometimes the parameter values remain the same for most of the calls and others keep on changing.

For example, we have a function:

Power (long x, int n)

Where x is the number to take power of and n is the power to which x is required to be raised. Suppose while using this function you came to know that 90% of the calls are for squaring the number x in your problem domain. So, the default value of a parameter could be provided inside the function prototype or function definition.

Program example:

```
void f ( int i = 1, double x = 10.5 )
{
cout << "The value of i is: " << i;
cout << "The value of x is: " << x;
}
```

Now this function can be called 0, 1 or 2 arguments.

Suppose we call this function as: f();

See we have called the function f() without any parameters, although, it has two parameters. It is perfectly all right and this is the utility of default function arguments. What do you think about the output. Think about it and then see the output below:

The value of i is: 1

The value of x is: 10.5

In the above call, no argument is passed; therefore, both the parameters will use their default values. Now if we call this function as:

f(2);

In this case, the first passed in argument is assigned to the first variable (left most variable) i and the variable x takes its default value. In this case the output of the function will be as under:

The value of i is: 2 The value of x is: 10.5

The important point here is that your passed in argument is passed to the first parameter (the left most parameter).

The first passed in value is assigned to the first parameter, second passed in value is assigned to the second parameter and so on. The value 2 cannot be assigned to the variable x unless a value is explicitly passed to the variable i.

See the call below: f(1, 2);

The output of the function will be as under:

The value of i is: 1 The value of x is:

// A program with default arguments in a function prototype

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

void show( int = 1, float = 2.3, long = 4 );

int main()

{
 show(); // All three arguments default
 show( 5 ); // Provide 1st argument
 show( 6, 7.8 ); // Provide 1st and 2nd
 show( 9, 10.11, 12 ); // Provide all three argument

}

void show( int first, float second, long third )

{

 cout << "\nfirst = " << first;

 cout << ", second = " << second;
```

```
 cout << ", third = " << third;

}
```

The output of the program is:

first = 1, second = 2.3, third = 4

first = 5, second = 2.3, third = 4

first = 6, second = 7.8, third = 4

first = 9, second = 10.11, third = 12

Placement of Variable Declarations

This has to do with the declaration of the variables inside the code. In C language, all the variables are declared at the top of the function or code block and then we can use them later on in the code.

Inline Function:

Difference in macros and Inline function:

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

#define MAX( A, B ) ((A) > (B) ? (A) : (B))
 //Defining macros

inline int max( int a, int b )
//Defining Inline Function
{
 if ( a > b )

return a;

 return b;
}

int main()
{
```

```
int i, x, y;

x = 23; y = 45;

i = MAX( x++, y++ ); // Side-effect:

// larger value incremented twice

cout << "x = " << x << " y = " << y << '\n';
x = 23; y = 45;

i = max( x++, y++ ); // Works as expected

cout << "x = " << x << " y = " << y << '\n';
}
```

Here the output of inline function would be correct. But the output of macros would be wrong. Macros would first make an increment if the statement is true and then run the program again to produce output.

Function Overloading:

You have already seen overloading many times.

For example, when we used cout to print our string and then used it for int, long, double and float etc.

cout << “This is my string”; cout << myInt ;

This magic of cout that it can print variables of different data types is possible because of overloading.

What is overloading?

“Using the same name to perform multiple tasks or different tasks depending on the situation is known as overloading.”

cout is doing exactly same thing that depending on the variable type passed to it, it prints an int or a double or a float or a string. That means the behavior is changing but the function cout << looks identical.

Lecture No. 26

Classes and Objects

"A class includes both data members as well as functions to manipulate that data"
These functions are called 'member functions'. We also call them methods. So a class has data (the variables) and functions to manipulate that data. A class is a 'user defined' data type.

"Instances of a class are called objects"

When we take a variable of a class, it becomes the instance of that class, called object.

```
class name_of_class
{
// definition of class
 }
```

Structure of a class

"The default visibility of all the data members and member function of a class is hidden and private"
'Private' is also a keyword.

Private:

Now all the data and functions following this statement will have the private visibility. To define the public data, we need to write the keyword public with a colon as;

Public:

Now all the data and functions following the public keyword will have the public visibility

```
Class Date

{

Private:

// private data and functions

Public:

// public data and functions

};
```

prayers remember me

Sample program

```
Class Date
{
Private:
Void display ();
Date (int, int, int);
Public:
int day, month, year;
};
```

The double colon is called scope resolution operator. It resolves the scope and tells that this function belongs to whom.

Lecture No. 27

Object Oriented Programming (OOP):

- C++ was initially named as C with classes because it is very similar to C but it deals with objects and classes.
- A bigger program results into less readability, maintainability and consists of more bugs.
- It helps creating real world scenarios.

OOP works on concept of classes and objects.

Class is basically a template used to create an object. A class does not take space in memory until an object is made.

There are certain rules to access data in Classes and object system.

In classes and objects, we decompose the problem into objects and build data and functions on them.

prayers remember me

Class – Basic template for creating object

Object – Basic run time entity

Data encapsulation – Securing data and functions in form of a capsule

Inheritance – The data of a class can be shared to another similar class

Benefits:

- Better code reusability using objects and inheritance
- Principles of data binding help secure data
- Multiple objects can co-exist without any interference

Example of Class & Object:

We have two persons.

Person a and Person b

Let's talk about the basic properties which are found in both of them like

- **Hair color**
- **Eye color**
- **Weight**
- **Height**

Now these four properties are basically representing the data of person a and b.

On the other hand we have some actions performed by these two persons.

- **Sleep( )**
- **Eat( )**
- **Run( )**

All the data of person a and b can be stored into a class. The basic properties represent the data and are known as data members. The actions represent the functions done on them and are known as function members. And Person a and b themselves are Obejcts.

Structure of Class:

Class person

prayers remember me

{

Data

Functions

};

Making an Object:

Making an object is same as making an integer.

While making an integer a, we used to type:

int a;

Similarly for an object:

Person a; → Person is the name of Class. Name of Object is “a”

Person b; → Person is the name of Class. Name of Object is “b”

After creating a Class, you can create as much objects as you want.

Data Privacy:

In Classes and objects system, the data is secured. The developer can choose either to make the data of program public or private.

Public data can be seen and changed by anyone. Private data cannot be seen or changed.

Program#1

```
#include<iostream>

using namespace std;

class Employee

{

private:

int a,b;
```

prayers remember me

```
public:
int c,d;
void setData(int a1, int b1);
void getData()
{
cout<<"The value of a is  "<<a;
cout<<"\nThe value of b is  "<<b;
cout<<"\nThe value of c is  "<<c;
cout<<"\nThe value of d is  "<<d;
}
};
void Employee :: setData(int a1, int b1)
{
a=a1;
b=b1;
}
int main()
{
Employee Ali;
Ali.setData(1,4);
Ali.getData();
return 0;
}
```

Constructor:

prayers remember me

It is used to initialize the value of the class data. It is used instead of setData function.

It is always declared in public part.

It does not return value.

Example:

```
Class test
{
Private:
int a,b;
Public:
  test( )      // This is a Simple constructor
{
a = 55;
b= 32;
}


};
```

Default arguments with constructors:

You can assign a value to a constructor within the constructor.

Program#2

```
#include<iostream>
using namespace std;
class Simple {
   int data1;
   int data2;
   public:
```

prayers remember me

```
    Simple()
    {
    }
    void printData();
};

void Simple::printData() {
    cout<<"The value of data is  "<<data1<<" & "<<data2;
}
int main()
{
Simple s(1);
s.printData();
return 0;
}
```

Construction Overload:

- In Construction overload, many parameters are passed at the same time.

Program#3

```
#include<iostream>
using namespace std;
class Simple {
    int data1;
    int data2;
    public:
```

prayers remember me

```
    Simple(int a,int b)        // This is constructor Overloading
    {
        data1 = a;
        data2 = b;
    }
    Simple (int a)
    {
        data1 = a;
        data2 = 5;
    }
    void printData();
};

void Simple::printData() {
    cout<<"\nThe value of data is  "<<data1<<" & "<<data2;
}
int main()
{
Simple s1(1,5);
s1.printData();

Simple s2(5);
s2.printData();
return 0;
}
```

prayers remember me

Destructor:

- It clears the memory when you remove an object.

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

// Destructor never takes an argument nor does it return any value
int count=0;

class num{
    public:
        num(){
            count++;
            cout<<"This is the time when constructor is called for object number"<<count<<endl;
        }

        ~num(){
            cout<<"This is the time when my destructor is called for object number"<<count<<endl;
            count--;
        }
};

int main(){
    cout<<"We are inside our main function"<<endl;
    cout<<"Creating first object n1"<<endl;
```

prayers remember me

```
    num n1;
    {
      cout<<"Entering this block"<<endl;
      cout<<"Creating two more objects"<<endl;
      num n2, n3;
      cout<<"Exiting this block"<<endl;
    }
    cout<<"Back to main"<<endl;
    return 0;
}
```

Lecture No. 28

Memory Allocation in C

We studied about few functions of memory allocation in C: malloc(), calloc() and realloc(). Using these functions, memory is allocated while the program is running. This means while writing your program or at compile time, you don't need to know the size of the memory required. You can allocate memory at runtime (dynamically) that has many benefits.

Memory Allocation in C++:

The memory allocation in C++ is carried out with the use of an operator called new. Notice that new is an operator while the malloc() was a function. Let's see the syntax of new operator through the following example:

```
new int;
```

In our program, we can write it as:

```
int *iptr;
iptr = new int;
```

So while using new operator, we don't need to supply the number of bytes allocated. New operator can be used for other data types like char, float and double etc.

Delete:

prayers remember me

The operator to free the allocated memory using new operator is delete. So whenever, we use new to allocate memory, it will be necessary to make use of 'delete' to deallocate the allocated memory. delete iptr;

Remember, new is an operator, it is not a function. Whenever we use new, we don't use parenthesis with it, no number of bytes or sizeof operator is required and no cast is applied to convert the pointer to the required type.

New Operator & Classes:

As we declare a pointer to a primitive datatype, similarly, we can have a pointer to a class object.

Date *dptr;

//dptr is a pointer to an object of type Date.

Now, we create the object using the new operator

dptr = new Date;

The new operator in the above statement (dptr = new Date;) has automatically determined the size of the Date object and allocated memory before returning a pointer of Date * type. Is this all what new is doing? Actually, it is doing more than this. It is also creating an object of type Date.

Whenever new operator is used to create an object, following actions are performed by it:

- It automatically determines the size of the memory required to store that object, leaving no need for the use of sizeof operator.
- Calls the constructor of the Class, where the programmers normally write initialization code.
- Returns pointer of the class type that means no casting is required.

Main () Function and Classes:

As you go along and write your own classes, you will realize that almost 90% of your program's code lies inside the class definitions. So firstly we write our classes and main() function is written after classes have been defined. That is why the main() function is very small.

Class Abstraction:

The users do not need to know how the functions or interfaces are implemented, what are the variables, how is the data inside and how is it being manipulated inside a class, it is abstract to the users.

Messages and Methods:

When we create an object, we ask that object to do something by calling a function. This way of asking objects in Windows operating system is called Messaging or in other words function calling is sending a message to the object. Sending a message is a synonym of calling a method of an object.

Lecture No. 29

Friend:

A type of declaration used within a class to grant other classes or functions access to that class.

Function

A C++ entity that is a sequence of statements. It has its own scope, accepts a set of argument values, and returns a value on completion.

Friend Function:

This function makes a connection with class to access its data. It is not a part of object; it only accesses the data it has been allowed to.

Syntax:

Friend return_type friend_function_name(int, char);

Properties of friend functions:

- Not in the scope of class
- Since it is not in the scope of the class, it cannot be called from the object of that class.
  c1.sumComplex() == Invalid
- Can be invoked without the help of any object
- Usually contains the objects as arguments
- Can be declared inside public or private section of the class
- It cannot access the members directly by their names and needs object_name.member_name to access any member.

prayers remember n...

Program Example:

```
#include<iostream>

using namespace std;

class Complex{

    int a, b;

    friend Complex sumComplex(Complex o1, Complex o2);

    public:

        void setNumber(int n1, int n2){

            a = n1;

            b = n2;

        }


             void printNumber(){

            cout<<"Your number is "<<a<<" + "<<b<<"i"<<endl;

        }

};

Complex sumComplex(Complex o1, Complex o2){

    Complex o3;

    o3.setNumber((o1.a + o2.a), (o1.b+o2.b))

    ;

    return o3;
```

prayers remember me

```
    }

    int main(){

        Complex c1, c2, sum;

        c1.setNumber(1, 4);

        c1.printNumber();

        c2.setNumber(5, 8);

        c2.printNumber();

        sum = sumComplex(c1, c2);

        sum.printNumber();

        return 0;

    }
```

Friend Classes

We have seen that a class can define friend functions for itself. Similarly a class can be declared as a friend class of the other class. In that case, the function of a class gets complete access to the data members and functions of the other class.

The syntax of declaring a friend class is that within the class definition, we write the keyword friend with the name of the class. It is going to be a friend class. i.e.

friend class-name;

We can also write the word class after the keyword friend and before the class name as:

```
friend class class-name;

class ClassOne

{

friend OtherClass;

private:
```

prayers remember me

```
//here we write the data members of ClassOne

};
```

In this sample program, we declared "*Other Class"* as a friend class of "*ClassOne"*.

Lecture No. 30

Reference

Another name for an object Access to an object via a reference is like manipulating the object itself. References are typically implemented as pointers in the underlying generated code.

Reference data type

In reference data type, we basically initialize two variables as one. We use "&" for this operation.

How do we declare a reference?

We declare it by using & operator.

For example:

```
int &i;
```

It means that i is a reference to an integer.

Here we are not initializing two variables. We declared an integer reference and initialized it. Now j is another name for i. Does it mean that it creates a new variable? No, its not creating a new variable. Its just a new name for the variable which already exists.

```
#include<iostream>

using namespace std;

int main()

{

int i;

 int &j = i;

 i = 123;

 cout << "\n The value of i = " << i;
```

prayers remember me

```
 cout << "\n The value of j = " << j;
 return 0;
}
```

Output:

i = 123

j = 123

Swap:

A swap function is used to interchange values of two variables.

Example Program:

```
#include<iostream>
using namespace std;
int main()
{
int a=5;
 int b=4;
swap(a,b);
cout<<a;
cout<<"\n"<<b;
}
```

Difference in Reference and Pointer:

Pointer	Reference
A pointer is a variable that holds a memory address.	A reference is a synonym of an existing variable.
Pointer can be initialized at any time.	Reference has to be initialized at the point of definition.

prayers remember me

A pointer can point to many different objects.	A reference can point to only one object.
A pointer can have a NULL value.	Reference can never be NULL.

References as Return Values:

A function itself can return a reference. The syntax of declaration of such a function will be as under.

The syntax

datatype& function_name (parameter list)

Suppose we have a function myfunc that returns the reference to an integer. The declaration of it will be as:

int & myfunc() ;

Dangling Reference:

You return reference to local variable of function. When function returns, local variable ceases to exist, and the returned reference becomes dangling: it does not reference a valid variable. Solution: return value, not reference, or allocate object from heap and return a (smart) pointer

Lecture No. 31 important QUIZ and short long question

Operator overloading:

- Operator Overloading is quite similar to Function Overloading.
- There are two types of operators to overload: unary and binary.
- Unary operators are the ones that require only one operator to work. Unary operators are applied to the left of the operand. For example, ^ , &, ~ and!.
- Binary operators require two operands on both sides of the operator. +, -,* , /, %, =, < and > are examples of binary operators.
- A unary operator takes only one operand & one argument (++,--), while binary operator takes two operators & two arguments.
- There are some restrictions while performing Operator Overloading. For example, only existing C++ operators are overloaded without creating a new one in the language.
- Also, it should not impact the type, semantics (behavior), arity (number of operands required), precedence (high/low precedence) and associativity (left/right) of the operator.

prayers remember me

- For binary member operators, operands on the left drives (calls) the operation. Operator functions written as non-members but friends of the class get both the operands as their arguments.
- Operators can be written as non-members and even without making them friends. But this is tedious and less efficient way, therefore, it is not recommended.

"In C++, we can change the way operators work for user-defined types like objects and structures. This is known as operator overloading."

For example, important

Suppose we have created three objects c1, c2 and result from a class named complex that represents complex numbers.

Since operator overloading allows us to change how operators work, we can redefine how the + operator works and use it to add the complex numbers of c1 and c2 by writing the following code:
Result = c1 + c2

This way we can add two complex numbers easily.

Non-member Operator Functions:

There are two situations under which operator overloading must be done by functions that are not members of a specific class. The situation we are talking about right now is:

- The class to which the member function "should" be added is not available for modification. This frequently occurs with classes that are in standard class libraries. An example is the stream I/O library.

To allow operator overloading by non-member functions, the rules used by the compiler involve two steps. If an expression of the form "x op y" is encountered, the compiler will check:

- Is there a member function in the class of object x of the form "operator op(Y) where Y is the class of object y, and, if not,
- is there a non-member function of the form "operator op(X,Y) where X is the class of object x and Y is the class of object y.

The rules are applied in order. An overloaded operator will be used if either of the two rules is satisfied.

When operator overloading is achieved using non-member function there are two cases to be considered:

prayers remember me

- The overloaded operator uses only the public interface of the class(es) involved in the overloading, or
- The overloaded operator requires access to the private data of the class(es).

Program Example:

```
#include <iostream>

using namespace std;

class Clock

{

    int hours;

    int minutes;public:

    Clock() {};

    Clock(int h, int m)

    {

        hours = h; minutes = m;

      }

    Clock operator+(Clock & t);

    int hr() { return hours; }

    int min() { return minutes;


    }


};

Clock Clock::operator+(Clock & t)

{        Clock sum;

sum.hours = hours + t.hours + (minutes + t.minutes)/60;

sum.minutes = (minutes + t.minutes) % 60;
```

prayers remember me

```
return sum;

    }

ostream & operator<<(ostream & os, Clock & t){     os << t.hr() << " hours, " << t.min() << " minutes";

return os;

    }

int main()

{          Clock a(1, 40);

Clock b(3, 29);

Clock c(2, 19);

cout << a << endl << b << endl << c << endl;

a = a + b + c;   cout << a << endl;

return 0;

}
```

Lecture No. 32

Overloading Minus Operator

The operator keyword declares a function specifying what operator-symbol means when applied to instances of a class. This gives the operator more than one meaning, or "overloads" it. The compiler distinguishes between the different meanings of an operator by examining the types of its operands.

The unary operators operate on a single operand and following are the examples of Unary operators:

- The increment (++) and decrement (--) operators.
- The unary minus (-) operator.
- The logical not (!) operator

Program Example:

```
#include <iostream>
```

prayers remember me

```
using namespace std;
class Distance {
  private:
  int feet;
  int inches;
  public:
  // Constructor
  Distance(int f, int i) {
    feet = f;
    inches = i;
  }
  // method to display distance
  void display() {
    cout << "F: " << feet << " I:" << inches <<endl;
  }
  // overloaded minus(-) operator
  Distance operator-() {
    feet = -feet;
    inches = -inches;
    return Distance(feet, inches);
  }
};
int main() {
```

prayers remember me

```
    Distance D1(3, 4), D2(-1, 10);

    D1;

    D1.display();

    D2;

    D2.display();

    return 0;

}
```

Unary Operator:

Unary operator: are operators that act upon a single operand to produce a new value.

Types of unary operators:

1. Unary minus(-)
2. Increment(++)
3. Decrement(- -)
4. NOT (!)
5. Addressof operator(&)
6. sizeof()

Program Example:

```
#include <iostream>

using namespace std;


int main()

{

    // Post increment

    int a = 1;

    cout << "a value: " << a << endl;

    int b = a++;

    cout << "b value after a++ : " << b << endl;

    cout << "a value after a++ : " << a << endl;


    // Pre increment
```

prayers remember me

```
    a = 1;
    cout << "a value:" << a << endl;
    b = ++a;
    cout << "b value after ++a : " << b << endl;
    cout << "a value after ++a : "<< a << endl;

    // Post decrement
    a = 5;
    cout << "a value before decrement: " << a << endl;
    b = a--;
    cout << "b value after a-- : " << b << endl;
    cout << "a value after a-- : " << a << endl;

    // Pre decrement
    a = 5;
    cout << "a value: "<< a<<endl;
    b = --a;
    cout << "b value after --a : " << b << endl;
    cout << "a value after --a : " << a << endl;
    return 0;
}
```

prayers remember me

Lecture No. 33

Both C and C++ have a set of rules for converting one type to another. These rules are used in the following situations:

- When assigning a value. For example, if you assign an integer to a variable of type long, the compiler converts the integer to a long.
- When performing an arithmetic operation. For example, if you add an integer and a floating-point value, the compiler converts the integer to a float before it performs the addition.

- When passing an argument to a function; for example, if you pass an integer to a function that expects a long.
- When returning a value from a function; for example, if you return a float from a function that has double as its return type.

"In all of these situations, the compiler performs the conversion implicitly. The conversion explicit is possible by using a cast expression."

Can we do conversion with objects of our own classes?

The answer is yes. If we go to the basic definition of a class it is nothing but a user defined data type. As class is a user defined data type, we can also define conversion on it.

Initializing Strings

Let's define a String class. We will define it our self, without taking the built in String class of C. We know that a string is nothing but an array of characters. So we define our String class, with a data member buffer, it is a pointer to character and is written as *buf i.e. a pointer to character (array).

Initialization of strings can be written in the main program as under:

String s1 ("This is a test");

Thus, an object of String has been created and initialized. The string "This is a test" has been placed in its buffer.

What happens if we have another String object, let's say s2, and want to write s2 = s1 ; Here we know that the buffer is nothing but a pointer to a memory location. If it is an array of characters, the name of the array is nothing but a pointer to the start of the memory location.

Strcpy:

The strcpy () function in C++ copies a character string from source to destination. The strcpy () function takes two arguments: dest and src

Program Example:

```
#include <iostream>

#include <cstring>

#include <cstdlib>

using namespace std;

// class definition
```

```
class String
{
 private :
 char *buf ;
 public:
 // constructors
 String();
 String( const char *s )
 {
 buf = new char [ 30 ];
 strcpy (buf,s);
 }
 // display the string
 void display ( )
 {
 cout << buf << endl ;
 }
 // getting the length of the string
 int length ()const
 {
 return strlen(buf);
 }
 // overloading assignment operator
 void operator = ( const String &other );
};
```

prayers remember me

```
// ----------- Assignment operator
void String::operator = ( const String &other )
{
 int length ;
 length = other.length();
 delete buf;
 buf = new char [length + 1];
 strcpy( buf, other.buf );
}
//the main program that uses the new String class with its assignment operator:
main()
{
 String myString( "here's my string" );
 cout << "My string is = " ;
 myString.display();
 cout << '\n';
 String yourString( "here's your string" );
 cout << "Your string is = " ;
 yourString.display();
 cout << '\n';
 yourString = myString;
 cout << "After assignment, your string is = " ;
 yourString.display();
 cout << '\n';
```

prayers remember me

}

This Pointer:

Whenever an object calls a member function, the function implicitly gets a pointer from the calling object. That pointer is known as this pointer. 'This' is a key word.

We cannot use it as a variable name. 'This' pointer is present in the function, referring to the calling object.

For example

If we have to refer a member, let's say buf, of our String class, we can write it simply as: buf ; That means the buf of calling object is being considered.

We can also write it as this->buf; i.e. the data member of the object pointed by this pointer is being called.

These (buf and this->buf) are exactly the same. We can also write it in a third way as: (*this).buf;.

Assignment:

The process of giving a value to a pre-existing object. Also see copy constructor and initialization.

Self-Assignment:

Suppose, we have an integer 'i. In the program, somewhere, we write i = i ; It's a do nothing line which does nothing. It is not an error too. Now think about the String object, we have a string s that has initialized to a string, say, 'This is a test'. And then we write s = s ; The behavior of equal operator that we have defined for the String object is that it, at first deletes the buffer of the calling object. While writing s = s ; the assignment operator frees the buffer of s. Later, it tries to take the buffer of the object on right hand side, which already has been deleted and trying to allocate space and assign it to s.

This phenomenon is known as Self-Assignment.

Suppose we have the address of an object in a pointer. We write:

String s, *sptr ;

Now sptr is a pointer to a String while s is a String object.

In the code of a program we can write sptr = &s ;

This statement assigns the address of s to the pointer sptr.

prayers remember me

Now somewhere in the program we write s = *sptr ;

That means we assign to s the object being pointed by sptr. As sptr has the address of s , earlier assigned to it. This has the same effect as s = s ;.

The buffer of s will be deleted and the assignment will not be done. Thus, the program will become unpredictable

Program Example:

```
#include <bits/stdc++.h>

using namespace std;


// Tie Class

class Time {

    int hrs, mins;


public:

    // Constructor

    Time(int, int);


    // Casting operator

    operator int();


    // Destructor

    ~Time()

    {

        cout << "Destructor is called."

             << endl;
```

prayers remember me

```
    }
};

// Function that assigns value to the
// member variable of the class
Time::Time(int a, int b)
{
    hrs = a;
    mins = b;
}

// int() operator is used for Data
// conversion of class to primitive
Time::operator int()
{
    cout << "Conversion of Class"
         << " Type to Primitive Type"
         << endl;

    return (hrs * 60 + mins);
}

// Function perfomrs type conversion
// from the Time class type object
// to int data type
```

prayers remember me

```
void TypeConversion(int hour, int mins)
{
    int duration;

    // Create Time Class  object
    Time t(hour, mins);

    // Conversion OR duration = (int)t
    duration = t;
    cout << "Total Minutes are "
         << duration << endl;

    // Conversion from Class type to
    // Primitive type
    cout << "2nd method operator"
         << " overloading " << endl;

    duration = t.operator int();

    cout << "Total Minutes are "
         << duration << endl;

    return;
}
```

prayers remember me

```
// Driver Code
int main()
{
    // Input value
    int hour, mins;
    hour = 2;
    mins = 20;

    // Function call to illustrate
    // type conversion
    Type Conversion(hour, mins);
    return 0;
}
```

Lecture No. 34

Array of Object:

A class is a user-defined data type. Objects are instances of classes the way int variables are instances of ints. The declaration of arrays of user-defined data types is identical to the array of primitive data types. New is called before the constructor, and that delete is called after the destructor.

A global array called pool that can store all the Name objects expected.

Name of Class Size of array with object name;

Program Example:

prayers remember me

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

class Student {
  public:
  string name;
  int rollno;
  Student() {}
  Student(string x, int y) {
    name = x;
    rollno = y;
  }

  void printDetails() {
    cout << name << " - " << "Roll no." <<rollno<<endl;
  }
};

int main() {
  //declare array with specific size
  Student students[] = {Student("Haseeb", 5), Student("Abdullah", 4), Student("Abbas", 2)};

  for(int i=0; i < 3; i++) {
    students[i].printDetails();
  }
```

prayers remember me

}

Dynamic Array of Objects:

- A dynamic array is quite similar to a regular array, but its size is modifiable during program runtime.
- Dynamic Array elements occupy a contiguous block of memory.
- Once an array has been created, its size cannot be changed. However, a dynamic array is different.
- A dynamic array can expand its size even after it has been filled.

Syntax:

pointer_variable = new data_type;

1. String *text;
2. text = new String [5];

In line 1, we have declared a pointer text of String type.

In line 2, we are creating an array of 5 objects of String type. This statement allocates space for each object of the array, calls the parameter-less constructor for each object and starting address of the first object is
Assigned to the pointer text.

Program Example:

```
#include<iostream>

using namespace std;

int main() {

        int x, n;

        cout << "Enter the number of items:" << "\n";

        cin >>n;

        int *arr;

            arr = new int(n);

        cout << "Enter " << n << " items" << endl;
```

prayers remember

```
        for (x = 0; x < n; x++) {
                cin >> arr[x];
        }
        cout << "You entered: ";
        for (x = 0; x < n; x++) {
                cout << arr[x] << " ";
        }
        return 0;
}
```

Delete Operator:

There is a difference in delete arr and delete [] arr. The first one will only delete the pointer which contains the address for first object of class arr. But the second one will delete whole array "arr".

Overloading [ ] Operator to Create Arrays

We know that if we overload operators new and delete for a class, those overloaded operators are called whenever we create an object of that class. However, when we create an array of those class objects, the global operator new ( ) is called to allocate enough storage for the array all at once, and the global operator delete ( ) is called to enough storage for the array all at once, and the global operator delete ( ) is called to release that storage.

We can control the allocation of arrays of objects by overloading the special array versions of operator new[ ] and operator delete[ ] for the class.

Lecture No. 35

Streams

A C++ stream is a flow of data into or out of a program, such as the data written to cout or read from cin. cin is an example of an istream. ostream is a general purpose output stream.

Streams are nothing but an ordered sequence of bytes. They allow data to move from one part of the computer to another which may be the screen or key board from and to, or from memory or files on disc and so on. Byte stream is used to connect the source and the destination.

prayers remember me

"Every stream has an associated source and a destination"

```
#include<iostream>
using namespace std;
int main()
{
    float i,j,c;
    cin>>i;
    cin>>j;
    c = i/j;
    cout<<c;
}
```

- cin.get():

```
#include <iostream>
using namespace std
int main()
{
char name[25];
cin.get(name, 25);
cout << name;
return 0;
}
```

- cout.put():

```
#include <iostream>
using namespace std;
int main()
```

prayers remember me

```
{
    char gfg[] = "Welcom";
    char ch = 'e';

    // Print first 6 characters
    cout.write(gfg, 6);

    // Print the character ch
    cout.put(ch);
    return 0;
}
```

- cin.getline():

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

//macro definitions for maximum length of variables
#define MAX_NAME_LENGTH     50
#define MAX_ADDRESS_LENGTH  100
#define MAX_ABOUT_LENGTH    200

using namespace std;

int main()
{
```

prayers remember me

```
    char
name[MAX_NAME_LENGTH],address[MAX_ADDRESS_LENGTH],about[MAX_ABOUT_
LENGTH];

    cout << "Enter name: ";
    cin.getline(name,MAX_NAME_LENGTH);

    cout << "Enter address: ";
    cin.getline(address,MAX_ADDRESS_LENGTH);

    cout << "Enter about yourself (press # to complete): ";
    cin.getline(about,MAX_ABOUT_LENGTH,'#');    //# is a delimiter

    cout << "\nEntered details are:";
    cout << "\nName: "  << name << endl;
    cout << "Address: " << address << endl;
    cout << "About: " << about << endl;

    return 0;
}
```

- cin.read() - cout.write():

```
#include <iostream>
using namespace std;
int main()
{
 char buffer[80];
```

prayers remember me

```
cout << " \n Enter a sentence: \n" ;

cin.read(buffer, 3);


cout << " The sentence entered was: \n";

cout.write(buffer, cin.gcount());

cout << endl;

return 0;

}
```

Buffered input/output

Input/output costs and the I/O devices (keyboard, monitor and disc etc) are slower as compared to the speed of the microprocessor and the memory being used. To overcome this speed difference, we use the mechanism, called buffered input/output.

Predefined Stream Objects:

Object Meaning

- cin Standard input
- cout Standard output
- cerr Standard error with unbuffered output.
- clog Standard error with buffered output
- caux Auxiliary (DOS only)
- cprn Printer(DOS only)

All these functions (getline, get, read, unget and peek) are implemented as member functions of the input class.

Methods of streams

We can write something like:

cin.getline(char *buffer, int buff_size, char delimiter = '\n')

Examples using streams

A simple example showing the use of getline function.

prayers remember me

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

int main () {
   char data[100];

   // open a file in write mode.
   ofstream outfile;
   outfile.open("afile.dat");

   cout << "Writing to the file" << endl;
   cout << "Enter your name: ";
   cin.getline(data, 100);

   // write inputted data into the file.
   outfile << data << endl;

   cout << "Enter your age: ";
   cin >> data;
   cin.ignore();

   // again write inputted data into the file.
   outfile << data << endl;
```

prayers remember me

```
    // close the opened file.
    outfile.close();

    // open a file in read mode.
    ifstream infile;
    infile.open("afile.dat");

    cout << "Reading from the file" << endl;
    infile >> data;

    // write the data at the screen.
    cout << data << endl;

    // again read the data from the file and display it.
    infile >> data;
    cout << data << endl;

    // close the opened file.
    infile.close();

    return 0;
}
```

Lecture No. 36

prayers remember me

The manipulators are like something that can be inserted into stream, effecting a change in the behavior.

Non parameterized Manipulators do not take argument to control the formatting of input/output whereas parameterized manipulators take argument for formatting.

Stream Manipulation:

- *cin.eof()*
- *cin.bad()*
- *cin.clear()*

setw():

```
#include <iostream>

#include <iomanip>

using namespace std;

int main ()

{

int num = 453;

cout<<setw(10)<<num ;

return 0;

}
```

setfill():

```
#include <iostream>

#include <iomanip>

using namespace std;

int main ()

{

int num = 453;

cout<< setfill ('A')<< setw(10);

cout << 77;
```

prayers remember me

```
return 0;
}
```

setprecision():

```
#include <iostream>
#include <iomanip>
using namespace std;
int main ()
{
float num = 45.32423;
cout<< setprecision(4) << num;
return 0;
}
```

setbase()

```
#include <iostream>
#include <iomanip>
using namespace std;
int main ()
{
int a = 15;
cout<< setbase(16) << a;
}
```

Scientific Representation:

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

prayers remember me

```
int main()
{
    double x = 2343.23;

    cout.precision(5);

    cout << "without scientific flag: "
         << x << endl;

    // Using scientific()
    cout << "with scientific flag: "
         << scientific << x << endl;
    return 0;
}
```

Lecture No. 37

Overloading Insertion and Extraction Operators

Extraction operator's (>>) Insertion operator (<<)

- If we overload insertion (<<) and extraction (>>) operators then the user of our class, does not need to know the specific names of the functions to input and display our objects.
- Stream insertion (<<) and extraction operators (>>) are always implemented as non-member functions.
- Operator << returns a value of type ostream & and operator >> returns a value of type istream & to support cascaded operations.
- The first parameter to operator << is an ostream & object. cout is an example of an ostream object. Similarly first parameter to operator >> is an istream & object. cin is an example of an istream object. These first parameters are always passed by reference. The compiler won't allow you to do otherwise.

prayers remember me

- For operator >>, the second parameter must also be passed by reference.
- The second parameter to operator << is an object of the class that we are overloading the operator for. Similar is the case for operator >>.

Program Example:

```
#include <iostream>

using namespace std;


class Distance {

  private:

    int feet;            // 0 to infinite

    int inches;          // 0 to 12


  public:

    // required constructors

    Distance() {

      feet = 0;

      inches = 0;

    }

    Distance(int f, int i) {

      feet = f;

      inches = i;

    }

    friend ostream &operator<<( ostream &output, const Distance &D ) {

      output << "F : " << D.feet << " I : " << D.inches;

      return output;

    }
```

prayers remember me

```
    friend istream &operator>>( istream  &input, Distance &D ) {
      input >> D.feet >> D.inches;
      return input;
    }
};

int main() {
  Distance D1(11, 10), D2(5, 11), D3;

  cout << "Enter the value of object : " << endl;
  cin >> D3;
  cout << "First Distance : " << D1 << endl;
  cout << "Second Distance :" << D2 << endl;
  cout << "Third Distance :" << D3 << endl;

  return 0;
}
```

Lecture No. 38

User Defined Manipulators:

A manipulator created by the user for their benefit in a particular program is known as user defined manipulator.

SYNTAX:

ostream& manipulator_name (ostream& os)

Program Example:

prayers remember me

```
#include <iostream>
#include <cstdlib>
using namespace std;
// Gives System Beep
ostream & bell ( ostream & output ) // Manipulator
{
 return output << '\a' ;
}
// Gives Tab
ostream & tab ( ostream & output ) // Manipulator
{
 return output << '\t' ;
}
// Takes the cursor to next line
ostream & endLine ( ostream & output ) // Manipulator
{
 return output << '\n';
}
int main()
{
 cout << "Virtual " << tab << "University" << bell << endLine <<"A"; // Use of Mainpulator
}
```

Static Keyword:

Static Variable in Function:

prayers remember me

- When a variable is declared as static, space for it gets allocated for the lifetime of the program.
- Even if the function is called multiple times, space for the static variable is allocated only once and the value of variable is carried in the next function call.

Program Example:

```
#include <iostream>

#include <cstdlib>

using namespace std;

// Gives System Beep

ostream & bell ( ostream & output ) // Manipulator

{

 return output << '\a' ;

}

// Gives Tab

ostream & tab ( ostream & output ) // Manipulator

{

 return output << '\t' ;

}

// Takes the cursor to next line

ostream & endLine ( ostream & output ) // Manipulator

{

 return output << '\n';

}

int main()

{

 cout << "Virtual " << tab << "University" << bell << endLine <<"A"; // Use of Mainpulator
```

prayers remember me

```
    }
```

Static Object in Class:

As the variables declared as static are initialized only once because they are allocated space in separate static storage so, the static variables in class are shared by the objects.

Program Example:

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

class GfG
{
public:
    static int i;

    GfG()
    {
        // Do nothing
    };
};

int GfG::i = 1;

int main()
{
    GfG obj,obj1;
    // prints value of i
```

prayers remember me

```
   cout << obj.i<<"\n"<<obj1.i;
}
```

Static Data members of Class:

Literally speaking, the word ‘Static’ means the stationary condition of things. Stationary for object or class? Here it will be stationary for the class.

That means that static data will be created once and initialized once for that class.

Program Example:

```
#include <iostream>
#include<string.h>


using namespace std;
class Student {
  private:
  int rollNo;
  char name[10];
  int marks;
  public:
  static int objectCount;
  Student() {
    objectCount++;
  }

  void getdata() {
    cout << "Enter roll number: "<<endl;
    cin >> rollNo;
```

prayers remember me

```
        cout << "Enter name: "<<endl;
        cin >> name;
        cout << "Enter marks: "<<endl;
        cin >> marks;
    }

    void putdata() {
        cout<<"Roll Number = "<< rollNo <<endl;
        cout<<"Name = "<< name <<endl;
        cout<<"Marks = "<< marks <<endl;
        cout<<endl;
    }
};
int Student::objectCount = 0;
int main(void) {
    Student s1;
    s1.getdata();
    s1.putdata();
    Student s2;
    s2.getdata();
    s2.putdata();
    cout << "Total objects created = " << Student::objectCount << endl;
    return 0;
}
```

prayers remember me

C++ Strings

Operator	Meaning
=	Assignment
+	Concatenation
+=	Concatenation assignment
==	Equality
!=	Inequality
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
>>	Reads
<<	Prints

HEADER FILES	*TYPES (Full Forms)*
iostream.h	Include all input and output streams
conio.h	All console input and output functions
math.h	Include all Mathematical functions
stdlib.h	Include all standard library functions
string.h	All string manipulation functions
ctype.h	All character manipulating function
iomanip.h	Include all input and output manipulators
ftype.h	Include all file type functions

Precedence level goes down the table		
Operator name	Associativity	Operators
Scope resolution (included in C++)	left to right	::
Primary	left to right	() [ ] . -> **dynamic_cast typeid**
Unary	right to left	++ -- + - ! ~ & * **(type_name) sizeof new delete**
Pointer to Member(C++)	left to right	*. ->
Multiplicative	left to right	* / %
Additive	left to right	+ -
Bitwise Shift	left to right	<< >>
Relational	left to right	< > <= >=
Equality	left to right	== !=
Bitwise AND	left to right	&
Bitwise Exclusive OR	left to right	^
Bitwise Inclusive OR	left to right	|
Logical AND	left to right	&&
Logical OR	left to right	||
Conditional	right to left	? :
Assignment	right to left	= += -= *= /= <<= >>= %= &= ^= |=
Comma	right to left	,

prayers

strlen - Finds out the length of a string
strlwr - It converts a string to lowercase
strupr - It converts a string to uppercase
strcat - It appends one string at the end of another
strncat - It appends first n characters of a string at the end of another.
strcpy - Use it for Copying a string into another
strncpy - It copies first n characters of one string into another
strcmp - It compares two strings
strncmp - It compares first n characters of two strings
strcmpi - It compares two strings without regard to case ("i" denotes that this function ignores case)
stricmp - It compares two strings without regard to case (identical to strcmpi)
strnicmp - It compares first n characters of two strings, Its not case sensitive
strdup - Used for Duplicating a string
strchr - Finds out first occurrence of a given character in a string
strrchr - Finds out last occurrence of a given character in a string
strstr - Finds first occurrence of a given string in another string
strset - It sets all characters of string to a given character
strnset - It sets first n characters of a string to a given character
strrev - It Reverses a string

Lecture No. 39

Pointers:

Pointers are symbolic representation of addresses. They basically point towards start of a memory chunk in a memory location.

SYNTAX:

```
datatype *var_name;

int *ptr;
```

Program Example:

```
#include <bits/stdc++.h>

using namespace std;

void arr()

{
```

```
    //Declare an array
    int val[3] = { 5, 10, 20 };

    //declare pointer variable
    int *ptr;

    //Assign the address of val[0] to ptr
    ptr = val ;
    cout << "Elements of the array are: ";
    cout << ptr[0] << " " << ptr[1] << " " << ptr[2];
}
int main()
{
 arr();
}
```

References

Another name for an object Access to an object via a reference is like manipulating the object itself. References are typically implemented as pointers in the underlying generated code.

What is calling by value?

The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function.

Call by Value

The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function.

What is calling by reference?

The call by reference method of passing arguments to a function copies the address of an argument into the formal parameter. It means the changes made to the parameter affect the passed argument.

The use of call by reference is also important for the sake of efficiency

Dynamic Memory Allocation

Dynamic memory allocation refers to managing system memory at runtime.

prayers remember me

- malloc()
- calloc()
- realloc()
- free()

Assignment and Initialization

Initialization gives a variable an initial value at the point when it is created.

Assignment a variable a value at the some point after the variable is created.

Copy Constructor

A copy constructor is a constructor that creates an object using another object of the same java class.

Usage:

We want to copy a complex object that has several fields. We want to make a deep copy of an existing object.

```
#include <iostream>

using namespace std;

class Line {

  public:
    int getLength( void );
    Line( int len );             // simple constructor
    Line( const Line &obj);  // copy constructor
    ~Line();                     // destructor

  private:
    int *ptr;
};

// Member functions definitions including constructor
Line::Line(int len) {
  cout << "Normal constructor allocating ptr" << endl;
```

prayers remember me

```
   // allocate memory for the pointer;
   ptr = new int;
   *ptr = len;
}

Line::Line(const Line &obj) {
   cout << "Copy constructor allocating ptr." << endl;
   ptr = new int;
   *ptr = *obj.ptr; // copy the value
}

Line::~Line(void) {
   cout << "Freeing memory!" << endl;
   delete ptr;
}

int Line::getLength( void ) {
   return *ptr;
}

void display(Line obj) {
   cout << "Length of line : " << obj.getLength() <<endl;
}

// Main function for the program
int main() {
   Line line(10);

   display(line);
```

prayers remember me

```
  return 0;
}
```

Rules for Using Dynamic Memory Allocation

Use dynamic in the following situations:

- When you need a lot of memory. ...
- When the memory must live after the function returns. ...
- When you're building a structure (like array, or graph) of size that is unknown (i.e. may get big), dynamically changes or is too hard to precalculate.

Lecture No. 40

Objects as Class Members

A class is a user defined data type and it can be used inside other classes in the same way as native data types are used. Thus we can create classes that contain objects of other classes as data members.

Program Example:

```
// Program to illustrate the working of
// public and private in C++ Class
#include <iostream>
using namespace std;

class Room {

  private:
   double length;
   double breadth;
   double height;
```

prayers remember

```
  public:

    // function to initialize private variables
    void initData(double len, double brth, double hgt) {
        length = len;
        breadth = brth;
        height = hgt;
    }

    double calculateArea() {
        return length * breadth;
    }

    double calculateVolume() {
        return length * breadth * height;
    }
};
int main() {
    // create object of Room class
    Room room1;
    // pass the values of private variables as arguments
    room1.initData(42.5, 30.8, 19.2);
    cout << "Area of Room =  " << room1.calculateArea() << endl;
    cout << "Volume of Room =  " << room1.calculateVolume() << endl;
```

prayers remember me

```
    return 0;

}
```

Template Functions and Objects

The format for declaring function templates with type parameters is:

```
template <class identifier> function_declaration;
template <typename identifier> function_declaration;
```

Lecture No. 41

Template Functions

Function templates are special functions that can operate with generic types. This allows us to create a function template whose functionality can be adapted to more than one type or class without repeating the entire code for each type. In C++ this can be achieved using template parameters.

Program Example:

```
// If two characters are passed to function template, character with larger ASCII value is displayed.

#include <iostream>

using namespace std;


// template function

template <class T>

T Large(T n1, T n2)

{

        return (n1 > n2) ? n1 : n2;

}
```

prayers remember me

```
int main()
{
	int i1, i2;
	float f1, f2;
	char c1, c2;

	cout << "Enter two integers:\n";
	cin >> i1 >> i2;
	cout << Large(i1, i2) <<" is larger." << endl;

	cout << "\nEnter two floating-point numbers:\n";
	cin >> f1 >> f2;
	cout << Large(f1, f2) <<" is larger." << endl;

	cout << "\nEnter two characters:\n";
	cin >> c1 >> c2;
	cout << Large(c1, c2) << " has larger ASCII value.";

	return 0;
}
```

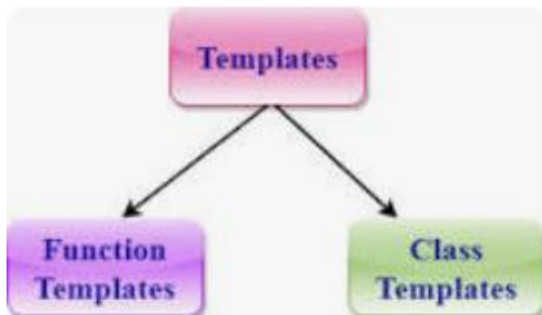
Lecture No. 42

Class Template:

We can define a template for a class. For example, a class template can be created for the array

prayers remember me

class that can accept the array of various types such as int array, float array or double array.



Program Example:

```
#include <iostream>

using namespace std;


// One function works for all data types. This would work

// even for user defined types if operator '>' is overloaded

template <typename T>

T myMax(T x, T y)

{

return (x > y)? x: y;

}


int main()

{

cout << myMax<int>(3, 7) << endl; // Call myMax for int

cout << myMax<double>(3.0, 7.0) << endl; // call myMax for double

cout << myMax<char>('g', 'e') << endl; // call myMax for char
```

prayers remember me

```
return 0;

}
```

What is class templates and Non-type parameters?

Class template can make use of another kind of template parameter known as non-type parameter. Non-type parameter is a special type of parameter that does not substitute for a type, but is instead replaced by a value.

What is templates and friend function?

A template function instantiated with one set of template arguments may friend to one template class instantiated with the same set of template arguments.

```
// C++ program to demonstrate the working of friend function


#include <iostream>

using namespace std;


class Distance {

    private:

        int meter;


        // friend function

        friend int addFive(Distance);


    public:

        Distance() : meter(0) {}


};
```

prayers remember me

```
// friend function definition
int addFive(Distance d) {

    //accessing private members from the friend function
    d.meter += 5;
    return d.meter;
}

int main() {
    Distance D;
    cout << "Distance: " << addFive(D);
    return 0;
}
```

Advantages and Disadvantages of Templates

The Template Method pattern provides you with the following advantages:

- As we saw earlier in the chapter, there is no code duplication.
- Code reuse happens with the Template Method pattern as it uses inheritance and not composition. Only a few methods need to be overridden.
- Flexibility lets subclasses decide how to implement steps in an algorithm.

The disadvantages of Template Method patterns are as follows:

- Debugging and understanding the sequence of flow in the Template Method pattern can be confusing at times. You may end up implementing a method that shouldn't be implemented or not implementing an abstract method at all. Documentation and strict error handling has to be done by the.

Standard Template Library (STL)

The Standard Template Library, or STL, is a C++ library of container classes, algorithms, and iterators; it provides many of the basic algorithms and data structures of computer science. The STL is a generic library, meaning that its components are heavily parameterized: almost every component in the STL is a template.

Lecture No. 43

What is programming Exercise- matrices and design recipe and program analysis?

Mathematics is a good domain to develop different classes and programs. E.g solutions for complex numbers, Matrices and Quadratic equations can be sought for developing our own classes.

Design recipe is a roadmap for defining function, which programmers use to make sure code they write does what they want it to do.
Program analysis is the process of looking at an organization's intended social and behavior change communication program and then identifying enabling.

What is a problem analysis?

A matrix is nothing but a two-dimensional array of numbers. It is normally represented in rows and columns. A matrix is represented as:

What is Matrix class and matrix constructor?

A matrix can have a number of rows and columns and each cell of a matrix can be represented by their combination A matrix constructor is a special 'member function having the same name as that of its class which is used to initialize some valid values to the data members of an object.

What is destructor of Matrix class?

A destructor is called for a class object. Destructor are usually used to de-allocate memory and do other cleanup for a class object and its class members when the object is destroyed.

Lecture No. 44&45

What is array and loop and variables?

An array is a type of data structure. We use an array to store multiple values of the same data type A loop is a sequence of instructions that is continually repeated until a certain condition is reached. A variable is a name given to a memory location.

prayers remember me

What is addition operator function and minus operator function and multiplication operator function?

Addition assignment operator (+=) adds the value of the right operand to a variable and assigns the result to the variable. Minus assignment operator minus the value right operand from the left operand and assigns the result to the left operand. Multiplication assignment operator multiplies the right operand with the left operand and assigns the result to the left operand.

What is an assignment operator?

Assignment operators are used to assigning value to a variable.

What is input function and transpose function?

A utility function transforms an outcome into a numerical value and measure the worth of an outcome. Two types of utility function: 1. Input function 2. Transpose function Transpose function returns a vertical range of cells as a horizontal range or vice versa. Input function is used to ask the user of the program a question, and then wait for a typed response

Most useful operator function in C++ for mcqs Operator Meaning

- =___________Assignment
- +___________Concatenation
- +=__________Concatenation assignment
- ==_________Equality
- !=____________Inequality
- <=_____________Greater than or equal to
- >___________less than
- >=___________ less than or equal to
- >>__________ Reads
- >>__________Prints

Header files	full forms
iostream.h	Include all input and output.streams
conio.h	All console input and output functions
math.h	include all mathematical functions
stdllb.h	include all standard library functions
string.h	All string manipulation functions
ctype.h	all character manipulating function

prayers remember me

iomanip.h — include all input and output manipulators

ftype.h — include all file type function ________

prayers remember me