1. **Literature Survey:**

The Hangman Game is prepared by using different kind of concepts. We have gone through different kind of concepts regarded to OOP, C++. Different kinds of concepts used in this project are as:

**Concepts Used in Project:**

Our project is completed using different kind of concepts. Some of them are as:

* **Classes:**

The building block of C++ that leads to Object Oriented programming is a Class. It is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class.

Syntax:

class *class\_name*

{

………

};

* **Constructor:**

A constructor is a member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object (instance of class) create. It is special member function of the class.

Syntax:

class *class\_name*

{

*class\_name( ) // function known as constructor*

*{*

*..............*

*}*

}

* **Primitive Built-in Types:**

C++ offers the programmer a rich assortment of built-in as well as user defined data types. One of the primitive built-in types for a function we use is bool. bool stands for Boolean,

*Boolean:* Boolean is a primitive built-in types which returns TRUE or FALSE if it used in the form of type of the function.

* **Function:**

A function is a set of statements that take inputs, do some specific computation and produces output.

Syntax:

*data\_type* funtion( *arguments list* )

{

    .......................

}

* **Inheritance:**

Inheritance is defined as derive quality and characteristics from parents or ancestors. Inheritance in Object Oriented Programming can be described as a process of creating new classes from existing classes.

Syntax:

class derived :*access specifier* base

{

………..

……….

};

* **Array:**

Array is a collection of similar kind of data member. Array can be the collection of any kind of data. For example: integer, character, floating point value, string, etc.

Syntax:

*data\_type*array\_name [size] = {……. *array elelments*……..};

* **Loops:**

Loop is the way of performing any kind of task several times. Loops in programming comes into use when we need to repeatedly execute a block of statement. Syntax of while loop is given below:

Syntax:

initialization expression;

while (test\_expression)

{

………….

update\_expression;

}

* **If Statements:**

The ability to control the flow of your program, letting it make decisions on what code to execute, is valuable to the programmer. This allows you to control if a program enters a section of code or not based on whether a given condition is true or false.

Syntax:

if (TRUE)

{

// *execute the code if trues*

}

else

{

*// when if case fails*

}

* **Switch Statement:**

Switch statement is the case of if statement when there will be more cases. In C++ we will be using switch case for long if statements.

Syntax:

switch ()

{

case 1: *// code for first case*

break;

case 2: *// code for second case*

break;

……..

……..

default :*// code for none of the case matched above*

}

* **Randomisation:**

The process of selecting an element randomly from a group of elements is known as Randomisation. C++ uses rand() function to randomize the elements. rand() function is used in C to generate random numbers. If we generate a sequence of random number with rand() function, it will create the same sequence again and again every time program runs.

Syntax:

int rand(void):

*returns a pseudo-random number in the range of 0 to RAND\_MAX.*

Elements can even be randomised in such a way that the format of randomisation is unknown. This can be performed by using the srand() function. The srand() function sets the starting point for producing a series of pseudo-random integers. If srand() is not called, the rand() seed is set as if srand(1) were called at program start. Any other value for seed sets the generator to a different starting point.

Syntax:

void srand( unsigned seed ):

*Seeds the pseudo-random number generator used by rand()*

*with the value seed.*

* **Transform:**

Transform applies an operation sequentially to the unary elements or binary elements ranges and stores the result in the range that begins at *result*.

*Unary Operation*: Applies a unary operator on input to convert into output

Syntax:

transform (*input*.begin, *input*.end, *output*.begin, *unary\_operation*)

*Binary Operation*: Applies a binary operator on input to convert into output

Syntax:

transform (*input*.begin1, *input*.end1, *input*.begin2, *output*.begin, *binary\_operation*)

Here *input* is the iterator kind of input.