RAMAKRISHNA MISSION
VIVEKANANDA CENTENARY COLLEGE, RAHARA

Roll Number: 715 Exam Roll number: 2022111243

Registration Number: A01-1112-117-014-2021 of 2021-2022

Semester : 1st Semester

Paper Code: CMSA CC-1

Date of Examination: 18.01.2022

B.Sc. End-Semester Examination

1. All bitwise operators available in C are:
   - bitwise AND (&)
   - bitwise OR (|)
   - bitwise XOR (^)
   - bitwise NOT (~)
   - left shift operator (<<)
   - right shift operator (>>)

The properties of recursive function are:
- There should be a base condition in the recursive function, for which the function does not recurse and may or may not return a value.
- At each function call, other than the base condition, the function should call itself recursively in a manner such that it progressively moves closer to the base condition.

A recursive function is much better than a non-recursive function as it helps to shorten the amount of code written and simplify the implementation of a problem, thus reducing chances of errors or bugs in the code.

// C function to calculate m^n

```
int power (int m, int n)
{
    if (n==0)
        return 1;

    int ans = 1;
    for (int i=0; i<n ; i++)
        ans *= m;

    return ans;
}
```

3. The main role of a constructor is initialization of the data members of the class when an object of that particular class is created.

The name of the constructor is always same as the class name to differentiate it from other functions of the class, so that the compiler knows which function to call when an object of that class is created.

A copy constructor of a class is a constructor which initializes the data members of a particular object of a class based on another object of the same class, essentially "copying" the data of that object into the current object of the class.

For example:

```
class ExampleClass
{
private:
    int a, b;
```

```
public:
    // Normal parameterized constructor
    ExampleClass (int A, int B)
    {
        a = A;
        b = B;
    }

    // copy constructor
    ExampleClass (ExampleClass* obj)
    {
        a = obj->a;
        b = obj->b;
    }
};
```

```
/* C++ program to perform constructor overloading and function overloading */
#include <iostream>
using namespace std;

class MyClass
{
    ~~int a, b;~~
private:
    int a, b;
public:
    // Constructor overloading
    MyClass (int A)
    {
        a = A;
        b = 0;
    }
```

```
    MyClass (int A, int B)
    {
        a = A;
        b = B;
    }

    // function overloading
    int sum ()
    {
        return a+b;
    }

    int sum (int x)
    {
        return a+b+x;
    }
};

int main()
{
    MyClass obj1 (5);
    MyClass obj2 (5,7);

    int ans1 = obj1.sum (12);
    int ans2 = obj2.sum ();
    return 0;
}
```

4. /* C program to check whether an inputted character is a capital letter, a small case letter, a digit or a special symbol using conditional statements */

```
#include <stdio.h>
int main()
{
    char c;
    printf ("Enter a character:\n");
    scanf ("%c", &c);

    if ((int)c >= (int)'A' && (int)c <= (int)'Z')
    {
        printf ("Capital letter.\n");
    }
    else if ((int)c >= (int)'a' && (int)c <= (int)'z')
    {
        printf ("Small letter.\n");
    }
    else if ((int)c >= (int)'0' && (int)c <= (int)'9')
    {
        printf ("Digit.\n");
    }
    else
    {
        printf ("Special character\n");
    }
    return 0;
}
```

The advantages of structure over union is that in a structure, multiple data members of the structure can be accessed simultaneously, while in union, multiple data members of the union cannot be accessed simultaneously.

The disadvantages of structure over union is that structure occupies more memory than a union as it occupies a total memory space of sum of all individual data members, while union occupies less space as it occupies memory space of only the largest data member of the union.

The memory size of a pointer variable is the size of an integer variable in C. It depends upon the size of an int variable in the particular system. (For example, it is 2 bytes in a 16-bit system, and 4 bytes in a 32-bit system).

The break statement is used in a loop to immediately stop the execution of the loop and return the control flow from the loop. It is also used to break out of a certain case in a switch statement in C.

The continue statement is used in a loop to skip the execution of the code after the continue statement for the current iteration and return the control flow to the loop control variable for the next iteration of the loop.

5.

Call by value

- It does not modify the original values of the parameters passed.
- It creates a copy of the parameters and works on the copies created.

Example:

```
#include <stdio.h>
void add (int a, int b)
{
    a = a+b;  // does not modify the
              // actual value of a passed
              // as parameter.
}
int main ()
{
  int a = 5, b=7;
  add (a,b);
  printf ("%d", a); // prints 5
  return 0;
}
```

Call by reference.

- It modifies the original value of the parameters passed.
- It does not create a copy of the parameters passed and ~~instead~~ instead modifies the original parameters themselves.

Example:

```
#include <stdio.h>
void add (int* a, int*b)
{  *a = *a + *b; // modifies actual
                 // value of a
}
int main ()
{  int a = 5, b=7
   add (&a,&b);
   printf ("%d", a); // prints 12
   return 0;
}
```

A function helps to group the code into various independent parts, which can be reusable, thereby reducing the amount of code written and improving the reusability and readability of the code.

A function can return an array using a pointer variable of the ~~specified~~ required datatype.

Example:

```
int* func()
{
    int* a = (int*) malloc (sizeof(int) * 5);
    for (int i=0; i< 5; i++)
        a[i] = i;
    return a;
}
```

```
// function to calculate the length of a string without using
// inbuilt function
int LengthOfString (char S[100]) // assuming length of string < 100 (it can be any value
                                 //                which fits into memory )
{
    int len = 0;
    for (len = 0; len < 100; len++)
    {
        if (S[len] == '\0')   // returns value of len when it reaches end of string
            return len;
    }
}
```

7. /* c++ code to showcase function overriding and dynamic binding */

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

class BaseClass
{
public:
    virtual void func1 ()
    {
        cout<< " BaseClass func1" << endl;
    }
    void func2()
    {
        cout << " BaseClass func2" << endl;
    }
};
class DerivedClass : public BaseClass
{
public:
    void func1()
    {   cout<< "Derived Class func1 " << endl;
    }
    void func2()
    {
        cout << " Derived Class func2" << endl;
    }
};
```

```
int main()
{
    ~~BaseClass * bf =~~

    BaseClass baseobj;
    DerivedClass derivedobj;

    // ~~fun~~ dynamic binding
    BaseClass * bptr = & baseobj;
     bptr -> func1();  // prints BaseClass func1

    BaseClass * bptr2 = & derivedobj;
     bptr2 -> func1(); // prints DerivedClass func1


    // function overriding.
    baseobj.func2(); // prints BaseClass func2
    derivedobj.func2(); // prints DerivedClass func2
    derivedobj.BaseClass::func2(); // prints BaseClass func2

   return 0;
}
```

A friend class is used to permit access of data members of its friend, regardless of the access specifier assigned to it.

Example:

```
#include <iostream>
class A
{
 ~~public~~
 private:
    int k;
 public:
    friend class B;
    void print();
};
```

```
class B
{
public :
    A aObj = A();
    aObj.k = 5; // does not throw error while accessing private member of class
};
```

If a function is declared as a friend to a class, it reduces the data hiding capabilities of the class.

~~It reduce~~ A friend function can breach access specifiers of the class by being able to access the private members of the class too. Thus, it reduces the data hiding capabilities of a class.