MA517M-Basic Programming Laboratory

Laboratory 7: Recursion and Structures

Panchatcharam Mariappan[1]

[1]Associate Professor
Department of Mathematics and Statistics
IIT Tirupati, Tirupati

September 29, 2025

Recursion

Recursion

- It is a process in which a function calls itself
- The function is called a recursive function.
- Example: Factorial Function

$$f(n) = n * f(n-1), f(1) = 1$$

- There should be at least two cases, one is the base case, and another one is the recurrence relation
- Write the base case in the if part and write the non-base case in the else part
- Most of the problems could be solved using if and else conditions; there are a few cases where you may need if..else if..else

Josephus Problem (Circle Elimination Game)

$$J(n,k) = \begin{cases} 0 & n = 1 \\ (J(n-1,k)+k) \bmod n & n > 1 \end{cases}$$

Mini N-Queens (4×4)

Place 4 queens on a 4×4 board so none attack each other.

Recursion

Factorial

```cpp
#include<iostream>
using namespace std;
unsigned long int factorial(unsigned long int n)
{
  if(n==1)
  return 1;
  else
  return n*factorial(n-1);
}
int main()
{
  unsigned long int n;
  cin>>n;
  cout<<n<<"! = "<<factorial(n)<<endl;
  return 0;
}
```

Direct and Indirect Recursion

1. When the body of a function calls itself, it is called **direct recursion**
2. When the body of a function A calls another function B, B calls another function C, and C calls function A, then it is called **indirect recursion** or mutual recursion.

Indirect Recursion: Write a C++ program to print numbers 1 to 10 in such a way that after dividing the number by 3,

- if the reminder is 0, add 2,
- if the reminder is 1, add 3,
- if the reminder is 2, subtract 1

Indirect Recursion

```
#include<iostream>
using namespace std;
int n=1;

void reminder0();
void reminder1();
void reminder2();

void reminder0()
{
  if(n<=10)
  {
    cout<<n+2<<endl;
    n++;
    reminder1();
  }
}

void reminder1()
{
  if(n<=10)
  {
    cout<<n+3<<endl;
    n++;
    reminder2();
  }
}
```

```

void reminder2()
{
  if(n<=10)
  {
    cout<<n-1<<endl;
    n++;
    reminder0();
  }
}


int main()
{
  reminder1();
  return 0;
}
```

Direct and Indirect Recursion

Direct Recursion	Indirect Recursion
Only one function called by itself	More than one function are called by the other function and number of time
Called by the same function	Called by other function
When function called next time, value of local variable will be stored	value will automatically lost when any other function is using the local variable
Engaged with memory location	For local variables, it is not
Dynamic	Determined by pointers that reference this memory Until the memory is freed

Direct and Indirect Recursion

- Generally slower than non-recursive programs
- Need to make a function call
- Program must save all its current state and retrieve it again later.
- Time Consuming
- Requires more memory to hold intermediate states in a stack.

`main` **function**

`main` function

- Remember that main is also a function.
- Can we pass arguments to main? Yes

`main` **function**

Let us reconsider the factorial problem

```
    #include<iostream>
using namespace std;
unsigned long int factorial(unsigned long int n)
{
  if(n==1)
  return 1;
  else
  return n*factorial(n-1);
}
int main(int argc, char **argv)
{
  cout<<argv[1]<<"! = "<<factorial(atoi(argv[1]))<<endl;
  return 0;
}
```

```
./a.out 5
```

`main` function

Let us reconsider the area of the rectangle problem

```
  #include<iostream>
using namespace std;
int main(int argc, char **argv)
{
  cout<<"Number of Arguments Passed = "<<argc<<endl;
  for(int i=0;i<argc;i++)
  cout<<"arguments["<<i<<"] = "<<argv[i]<<endl;
  float a=stof(argv[1]);
  float b=stof(argv[2]);
  cout<<"Area of the Rectangle is "<<a<<"x"<<b<<" = "<<a*b<<endl;
  return 0;
}
```

```
./a.out 5.2 6.5
```

Functions

- Beware of existing C++ functions. Use them appropriately to reduce your development time
- Give the name of the function appropriately
- Break functions into smaller tasks
- Distinguish between functions that return values and that don't
- Do not use the same name for function arguments and the parameters
- Collection of functions would be a good practice to debug and modify
- Try to fit the function in one line (don't use too many parameters)
- Number of arguments and parameters should agree
- function prototype with parameter name is valid. That is, both int `sum(int x , int y )` and int `sum(int , int )` are valid

Functions: Common Mistakes

- `double x,y` instead of `double x, double y` in the function parameter produces a compilation error
- Placing a semicolon after the function definition is a syntax error
- `double x` in the function parameters and using `double x;` (declaring again) inside the function would lead to a compilation error
- Defining a function inside another function
- Forgetting to return a value from a recursive function when it is needed
- Forgetting to write the base case in the recursive function

Structure

What is Structure?

- A structure is a user-defined data type in C++.
- It groups variables (members) of different types.
- Members can be of different data types.

Structure

```
struct structName{
  cType1 varName1;
  cType2 varName2;
  ....
  cTypeN varNameN;
  cType3 arrayName3 [arraySize];
  cType4 *ptrName4;
}[one or more structure variables];
```

Structure

```
struct Course
{
  char *code;
  char *name;
  float marks;
}

struct Students
{
  char *rollno;
  char *name;
  struct Course *Subject;
  char Grade;
  int ndaypresent;
  char attendcode;
};
```

- `struct`-keyword
- `Course`-structure name
- `code, name, marks`-members of structure

Structure

- Declaring a variable of a structure type:

```
Course A;
Students B;
```

- Assigning values to members:

```
A.code="MA517M";
B.name="Raja";
```

- Access members using dot operator (.).

```
   cout<<"Course Code  : "<<A.code<<endl;
   cout<<"Course Name  : "<<A.name<<endl;
   cout<<"Marks Scored : "<<A.Marks<<endl;
```

Structures with Function

- Pass Structure to functions

```
void Print(struct Course A)
{
   cout<<"Course Code   : "<<A.code<<endl;
   cout<<"Course Name   : "<<A.name<<endl;
   cout<<"Marks Scored  : "<<A.Marks<<endl;
}
```

- Structures can also have pointers:

```
Course *C=&A;
std::cout<<C->name;
```

-> Arrow Operator

- Structures can also have pointers:

```
Course *C=&A;
std::cout<<C->name;
```

- The arrow operator is used when you have a pointer to a structure or class object.
- It allows you to access members of the object that the pointer points to.

```
pointer_to_struct->member_name
```

- The arrow operator is just shorthand for this.

```
(*pointer_to_struct).member_name
```

- Use -> only with pointers.
- Use . with normal structure variables.
- Makes code cleaner and easier to read when working with pointers.

Why Structure?

- Groups related data together.
- Improves **code readability** and organization.
- Can be passed to functions or used with **pointers**.
- Basis for **advanced data types** like classes in C++.

Structure

```
1
2  #include<iostream>
3  #include<string>
4  using namespace std;
5
6  struct Course
7  {
8    string code;
9    string name;
10   float Marks;
11 };
12
13 struct Students
14 {
15   string rollno;
16   string name;
17   int ndayspresent;
18   char atcode;
19   int numcourses;
20   struct Course *Courses;
21   char grade;
22 };
```

```
1
2  void Print(struct Course A)
3  {
4    cout<<"Course Code  : "<<A.code<<endl;
5    cout<<"Course Name  : "<<A.name<<endl;
6    cout<<"Marks Scored : "<<A.Marks<<endl;
7  }
8  void Print(struct Students A)
9  {
10   cout<<"Student Details:"<<endl;
11   cout<<"Roll No: "<<A.rollno <<endl;
12   cout<<"Name: "<<A.name <<endl;
13   cout<<"Number of Days Present: "<<A.ndayspresent
           <<endl;
14   cout<<"Attendance Code: "<<A.atcode;
15   for(int i=0;i<A.numcourses;i++)
16   {
17     Print(A.Courses[i]);
18   }
19   cout<<"Grade : "<<A.grade<<endl;
20
21 }
```

Structure

```
int main()
{
  Course Course1={"MA517M", "Basic Programming Laboratory",0.0};
  Print(Course1);
  Students A;
  A.rollno="MA23M001";
  A.name="ABCDEF";
  A.ndayspresent=40;
  A.numcourses=6;
  A.Courses=new Course[A.numcourses];
  A.Courses[0]={"MA513L","Real Analysis",70};
  A.Courses[1]={"MA506L","Linear Algebra",65};
  A.Courses[2]={"MA516L","Basic Graph Theory",55};
  A.Courses[3]={"MA512L","Probability Theory",85};
  A.Courses[4]={"MA519L","Sampling Theory",75};
  A.Courses[4]={"MA518L","Groups and Rings",45};
  A.grade='S';
  A.atcode='G';
  Print(A);
  return 0;

}
```

Download the Code from here

Structure

```
#include<iostream>
struct Point
{
  double x,y,z;
}A,B;

void GetPoint(struct Point &node)
{
  cout<<"Enter the value of x: ";
  cin>>node.x;
  cout<<"Enter the value of y: ";
  cin>>node.y;
  cout<<"Enter the value of z: ";
  cin>>node.z;
}
void Distance()
{
  double dist;
  //Fill the code
}
```

```
void MidPoint()
{
  Point C;
  //Fill the code
  cout<<"The mid point of A and B is ("<<C.x<<",
      "<<C.y<<","<<C.z<<")"<<endl;
}

int main()
{
  cout<<"Enter the Vector A"<<endl;;
  GetPoint(A);
  cout<<"Enter the Vector B"<<endl;;
  GetPoint(B);
  Distance();
  MidPoint();
  return 0;
}
```

Download the Code from here

Structure

```
#include<iostream>
using namespace std;
#define SIZE 100
struct Vector
{
  int length;
  double values[SIZE];
};

int GetOperation()
{
  int op;
  cout<<"Enter the Operations for Vectors"
      <<endl;;
  cout<<"1. Addition: A + B"<<endl;;
  cout<<"2. Subtraction: A - B"<<endl;;
  cout<<"3. Scalar Multiplication: kA"<<
      endl;;
  cout<<"4. Dot Product: A . B"<<endl;;
  cout<<"5. Cross Product: A x B"<<endl;;
  cout<<"6. Quit"<<endl;;
  cin>>op;
  return op;
}
```

```
void Addition(struct Vector X, struct Vector Y
    )
{
  Vector Z;
  Z.length=X.length;

  for(int i=0;i<X.length;i++)
  {
    Z.values[i]=0;
    Z.values[i]=X.values[i]+Y.values[i];
    cout<<Z.values[i]<<endl;
  }
}

void Subtraction(struct Vector X, struct
     Vector Y)
{
  Vector Z;
  Z.length=X.length;

  for(int i=0;i<X.length;i++)
  {
    //Fill The Code
  }
}
```

Structure

```
int main()    {
    Vector A,B;
    int n;
    cout<<"Enter the dimension of the
          vector: ";
    cin>>n;
    A.length=n;
    B.length=n;
    cout<<"Enter the value of the first
          Vector"<<endl;;
    for(int i=0;i<n;i++)
    {
        cout<<"First Vec["<<i+1<<"]: ";
        cin>>A.values[i];
    }
    cout<<"Enter the value of the second
          Vector"<<endl;;
    for(int i=0;i<n;i++)
    {
        cout<<"First Vec["<<i+1<<"]: ";
        cin>>B.values[i];
    }
    int oper;
    int loop=0;
```

```
do    {
    oper=GetOperation();
    switch (oper)
    {
        case 1:
        Addition(A,B);          break;
        case 2:
        Subtraction(A,B);           break;
        case 3:
        ScalarMul(A);           break;
        case 4:
        DotProduct(A,B);          break;
        case 5:
        CrossProduct(A,B);          break;
        case 6:
        cout<<"Progaram Ends"<<endl;;
        loop=1;           break;
        default:
        cout<<"Invalid Input, Enter the
             correct input"<<endl;;
    }
}while(loop==0);
return 0;
}
```

Download the Code from here

Thanks

Doubts and Suggestions

panch.m@iittp.ac.in