Data Structures in C++

CMPE226- Data Structures

Recursion

Adapted from Textbook (D. S. MALIK), Lecture notes (C. TURHAN) and Lecture Notes (H. QU)

RECURSION

- Involves a function that calls itself to solve a ‘smaller’ version of the original problem.
- Example: The factorial problem (3!=6)

$$n! = \begin{cases} 1 \;\; if \; n = 0 \\ n * (n-1)! \quad if \;\; n > 0 \end{cases}$$

3! = 3 x 2!

3! = 3 x (2 x1!)

3!= 3 x (2 x (1 x 0!))

3!= 3 x (2 x (1 x 1))

3!= 3 x (2 x 1)

3!= 3 x 2

3!= 6

RECURSION

Every recursive process consists of

- **A base case,** stops recursion
- **A recursive case** that reduces the problem into one or more recursive calls.

Consider factorial problem:

Base case: 0!=1

Recursive case: $n * (n-1)! \quad if \;\; n > 0$

Exp.1 Recursive function implementing the Factorial

```
#include<iostream>
using namespace std;
int fact(int num){
        if(num==0)//base case
                return 1;
        else //recursive case
                return num*fact(num-1);
}
int main(){
        cout<<fact(3);
}
```

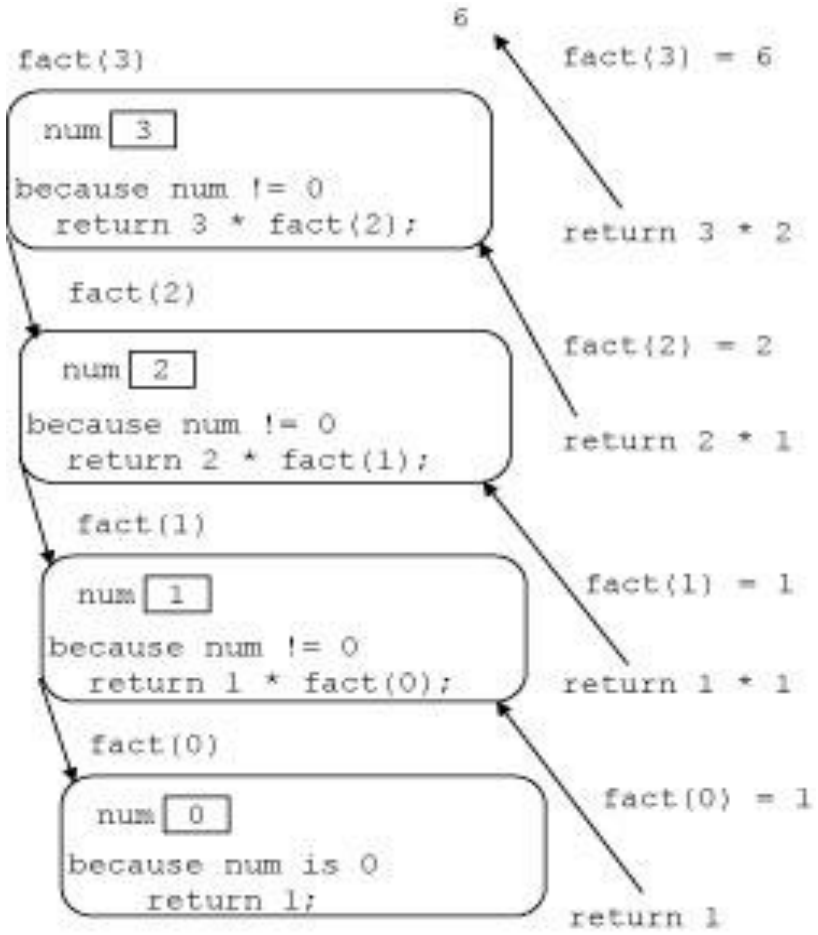
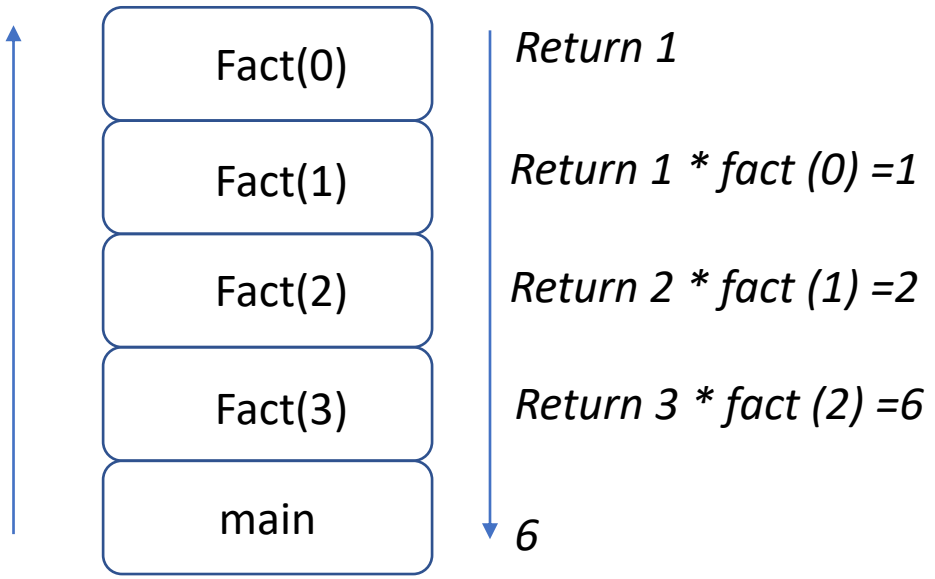


FIGURE 6-1 Execution of fact(3)

Runtime Stack (Call Stack)

- An area of memory set aside for programs / functions to use while they are running.
- It holds the activation record for each function call which includes all its parameters, local variables and return address.

```
#include<iostream>
using namespace std;
int fact(int num){
        if(num==0)//base case
                return 1;
        else //recursive case
                return num*fact(num-1);
}
int main(){
        cout<<fact(3);
}
```



Exp.2 Power operation in a recursive function

$2^3 = 8$ is equal to 2 x 2 x 2= 3 times

$a^n = ?$ Is equal to a x a … x a= n times

To write a recursive function first determine the base and general cases

Base case: if n=0 then $a^0 = 1$ *(or if n=1 then $a^1 = a$)*

General case: $a^n = a \times a^{n-1}$

So: $2^3 = 2 \times 2^2$

$2^3 = 2 \times (2 \times 2^1)$

$2^3 = 2 \times (2 \times (2 \times 2^0))$

$2^3 = 2 \times (2 \times (2 \times 1))$

$2^3 = 2 \times (2 \times 2)$

$2^3 = 2 \times 4 = 8$

```
#include<iostream>
using namespace std;
int power(int a,int n)//a is base, n is exponent
{
        if(n==0)//base case
                return 1;
        else //recursion case
                return a* power(a,n-1);
}
int main(){
        cout<<power(2,3);
}
```

Exp.2 Power operation in a recursive function

Rewrite the above function with following specifications:

Base case: $a^1 = a$

General case: $a^n = a \times a^{n-1}$

So: $2^3 = 2 \times 2^2$

$2^3 = 2 \times (2 \times 2^1)$

$2^3 = 2 \times (2 \times (2 \times 2^0))$

$2^3 = 2 \times (2 \times (2 \times 1))$

$2^3 = 2 \times (2 \times 2)$

$2^3 = 2 \times 4 = 8$

```
#include<iostream>
using namespace std;
int power(int a,int n)//a is base, n is exponent
{
        if(n==1)//base case
                return a;
        else //recursion case
                return a* power(a,n-1);
}
int main(){
        cout<<power(2,3);
}
```

Exp 3. Fibonacci Number Sequence:

The current number is the sum of two previous numbers: 1 1 2 3 5 8 ..
Output the n^{th} item in the series- if n=6 then fib(6) =8

Iterative Version

```
int  fib(int n){
  int i,twoback,oneback,curr;
  if ( n==1) {
    return 1;
  } else {
    twoback=0;
    oneback=1;
    for(i=2;i<=n;i++){
       curr=twoback+oneback;
       twoback=oneback;
       oneback=curr;
    }
  }
  return curr;
}
```

Recursive version

if n==0 result=0
if n==1 result=1
if n>=2 result=fib(n-2)+fib(n-1)

```
int  fib(int n){
  if ( n==0) {
    return 0;
  } else if ( n==1) {
    return 1;
  } else {
    return fib(n-2)+fib(n-1);
  }
}
```

NOTE: In general recursive programs are inefficient in terms of time and space compared to non-recursive versions. Yet recursive solutions are simpler and more natural

Exp 3. Fibonacci Number Sequence:

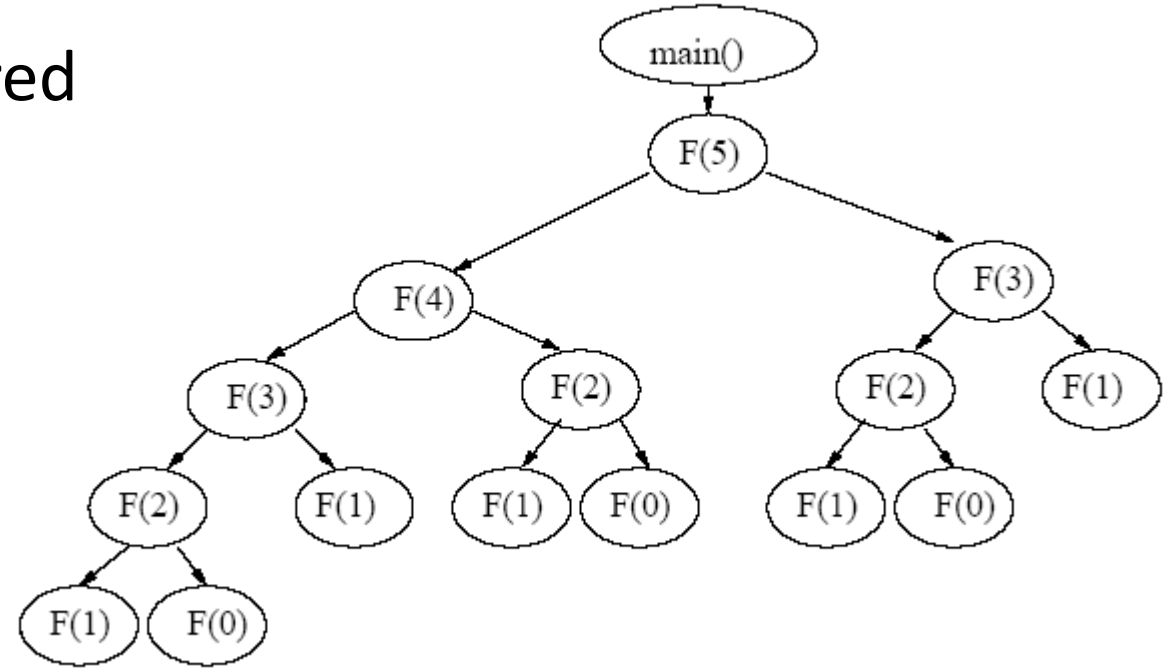
Fibonacci numbers (discussed in the lecture)

- How many calls to compute Fib(5)?

```
#include <iostream>

using namespace std;

int Fib(int n)
{
    // pre: n >= 0
    // post: return F(n)
    if (n == 0 || n == 1)
        return 1;
    return Fib(n-1) + Fib(n-2);
}

int main()
{
    cout << Fib(5) << endl;
    return 0;
}
```

Exp 3. Fibonacci Number Sequence:

- Answer: 15
- A Huge number of function invocations result in overhead and memory usage
- Iterative approach is preferred



Exp.4 Multiplying Two values with Addition

Write a recursive function called mult() that uses only addition to multiply 2 integers and return the result

e.g. mult (5, 3) will compute: 5 + 5 + 5 So mult(x,y) means to add x y times

NOTE: think all possible cases

What if y value is negative? Is it possible to add 5 for -3 times? No.

You should negate the result and convert y to a positive value –(5 x 3)

$5 * -3 = mult(5, -3) = -(5 + 5 + 5)$

$5 * 0 = 0$ Base case1 if y=0 result=0

$5 * 1 = 5$ Base case2 if y=1 result=x

Recursive case:

$5 * 3 = 5 + mult(5,2)$

$5 + mult(5,1)$

5

15

Exp.4 Multiplying Two values with Addition

Write a recursive function called mult() that uses only addition to multiply 2 integers and return the result

```
#include<iostream>

using namespace std;

int  mult(int x, int y){
        if (y<0)//if y is negative negate result, make y positive
                return -(mult(x,-y));
   else if(y==0)// Base case-1
        return 0;
   else if(y==1)// Base case-2
        return x;
   else //general case
        return x + mult(x,y-1);
}
```

Exp.5 Sum of numbers from 1 to n

Write a recursive function to sum integers from 1 to n.

e.g. n=5 the result: 1+2+3+4+5

Base case: n=1 result = 1

General case: n + func (n-1)

Func(5)

5 + Func(4)

4+ Func(3)

3+ Func(2)

2+ Func (1)

1

```
int sum(int n){
  if ( n==1){
     return 1;
  } else {
     return n + sum(n-1);
  }
}
```

Exp. 6 Trace the following program

```
#include<iostream>
using namespace std;
void displayMsg(int n){
        //if we print before recursive call worked as iterative- Output: 0 1 2 3 4
        cout<<n<<" ";
        if(n>0)
                displayMsg(n-1);//Recursive call
  //if we print after recursive call works in reverse order - Output: 4 3 2 1 0
  //burada ekrana yazilabilmesi icin butun cagrilan fonk. sonlandirilmasi gerekir
        cout<<n<<" ";
}
int main(){
        displayMsg(4);
}
```

Exp 7. Output the elements of a Linked List recursively in reverse order.

Base Case: List is empty: no action

General Case: List is nonempty

1. Print the tail
2. Print the element

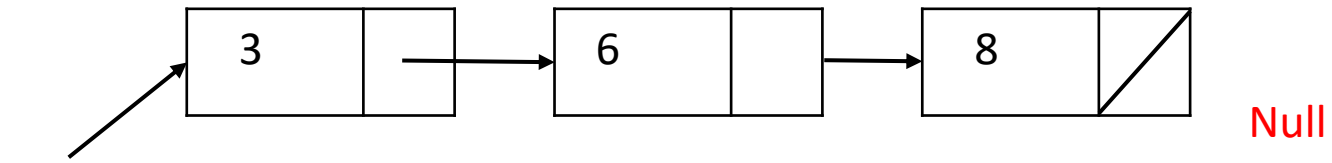
Function `reversePrint`

Given list pointer, prints list elements in reverse order

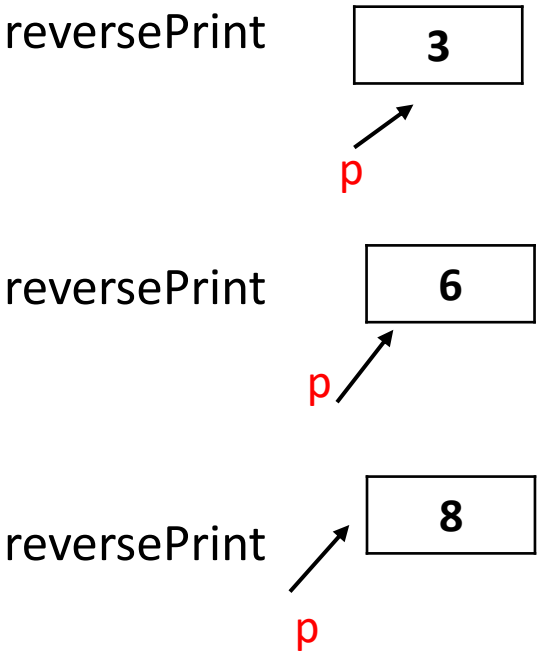
```
template <class T>
void reversePrint(node<T> *p){
  if(p!=NULL){
    reversePrint(p->link);//recursive call
    cout << p->info << endl;
  }
}

main(){
 reversePrint(head);
}
```

Exp 7. Output the elements of a Linked List recursively in reverse order.

```
template <class T>
void reversePrint(node<T> *p){
  if(p!=NULL){
    reversePrint(p->link);//recursive call
    cout << p->info << endl;
  }
}
main(){
 reversePrint(head);
}
```



Consider the LL below:
reversePrint(head);

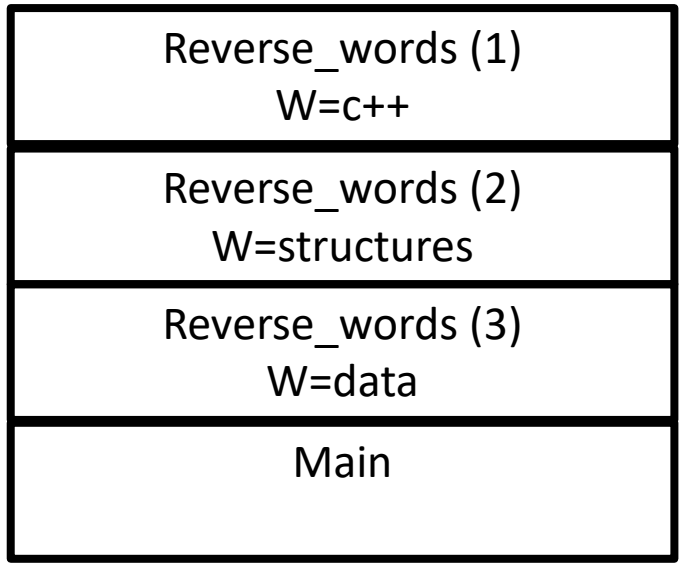


reversePrint (NULL)
Cout- 8
Cout- 6
Cout- 3

Exp 8. Write a recursive function to input n words & print them in reverse order :

Input : data structures c++
Output: c++ structures data

```
void reverse_words(int n){
  string word;
  if(n==1){
    cin >> word;
    cout << word << endl;
  }
  else {
    cin >> word;
    reverse_words(n-1);
    cout << word << endl;
  }
}
main(){
        reverse_words(3);
}
```



References

- CMPE226- Lecture Notes by Cigdem Turhan
- Data Structures Using C++, D.S. Malik, Thomson Course Technology, 2nd Edition.
- Lecture Slides by Huamin Qu, The Hong Kong University of Science and Technology (2005)