Recursive

SCSJ2013 Data Structures & Algorithms

Nor Bahiah Hj Ahmad & Dayang Norhayati A. Jawawi

Faculty of Computing

Objectives

- Identify problem solving characterestics using recursive.
- Trace the implementation of recursive function.
- Write recursive function in solving a problem

Introduction

- Repetitive algorithm - sequence of operations is executed repeatedly until certain condition is achieved.
- Implemented using loop :

  `while`, `for` or `do..while`.

- C++ allow programmers to implement recursive to replace loops.

- Not all programming language allow recursive implement, e.g. Basic language.

Introduction

- Recursive is a repetitive process in which an algorithm calls itself.
- A recursive procedure is mathematically more elegant than one using loops.
- Sometimes procedures can become straightforward and simple using recursion as compared to loop solution procedure.

Introduction

- Advantage : Recursive is a powerful problem solving approach, since problem solving can be expressed in an easier and neat approach.

- Drawback : Execution running time for recursive function is not efficient compared to loop, since every time a recursive function calls itself, it requires multiple memory to store the internal address of the function.

Recursive solution

- **Not all** problem can be solved using recursive.
- Recursive solve problem by:

1. breaking the problem into the same smaller instances of problem,
2. solve each smallest problem and
3. combine back the solutions.

Understanding recursion

Every recursive definition has 2 parts:

BASE CASE(S)

case(s) so simple

RECURSIVE CASE(S)

break the problem to smaller sub-problems

combine into a solution to the larger problem

Rules for Designing Recursive Algorithm

1. Determine the base case
2. Determine the general case
3. Combine the base case and general case

Designing Recursive Algorithm

- **Recursive algorithm.**

```
if (terminal case is reached)// base case
<solve the problem>

else                            // general case
< reduce the size of the problem and
   call recursive function >
```

Base case and general case is combined

Classic examples

- Multiplying numbers
- Find Factorial value.

Multiply 2 numbers using Addition Method

- Multiplication of 2 numbers can be achieved by using addition method.
- Example :

  To multiply 8 x 3, the result can also be achieved by adding value 8, 3 times as follows:

  8 + 8 + 8 = 24

Implementation of `Multiply()` using loop

```
int Multiply(int M,int N)
{  for (int i=1,i<=N,i++)
       result += M;
   return result;
}//end Multiply()
```

Implementation of recursive function: Multiply()

```
int Multiply (int M,int N)
{
   if (N==1)
      return M;
   else
      return M + Multiply(M,N-1);
}//end Multiply()
```

Recursive algorithm

3 important factors for recursive implementation:

- There's a condition where the function will stop calling itself.
- Each recursive function call, must return to the called function.
- Variable used as condition to stop the recursive call must change towards terminal case.

Tracing Recursive Implementation for `Multiply()`.

Step 1: Get the multiplication of 2 numbers.

Problem: Multiply (8,3);

Step 2: Run Multiply() function.

Sub problem1: int Multiplyint M, int N)

Value of M =8 and N =3.

Since , N ≠ 1, Multiply() will be called and the parameter value is reduced

```
return 8 + Multiply(8,3-1)
```

Step 3: Run Multiply() function.

Sub problem2: int Multiply(int M, int N)

Value of M =8 and N =2.

Since , N ≠ 1, Multiply() will be called and the parameter value is reduced

```
return 8 + Multiply(8,2-1)
```

Step 4: Run Multiply() function..

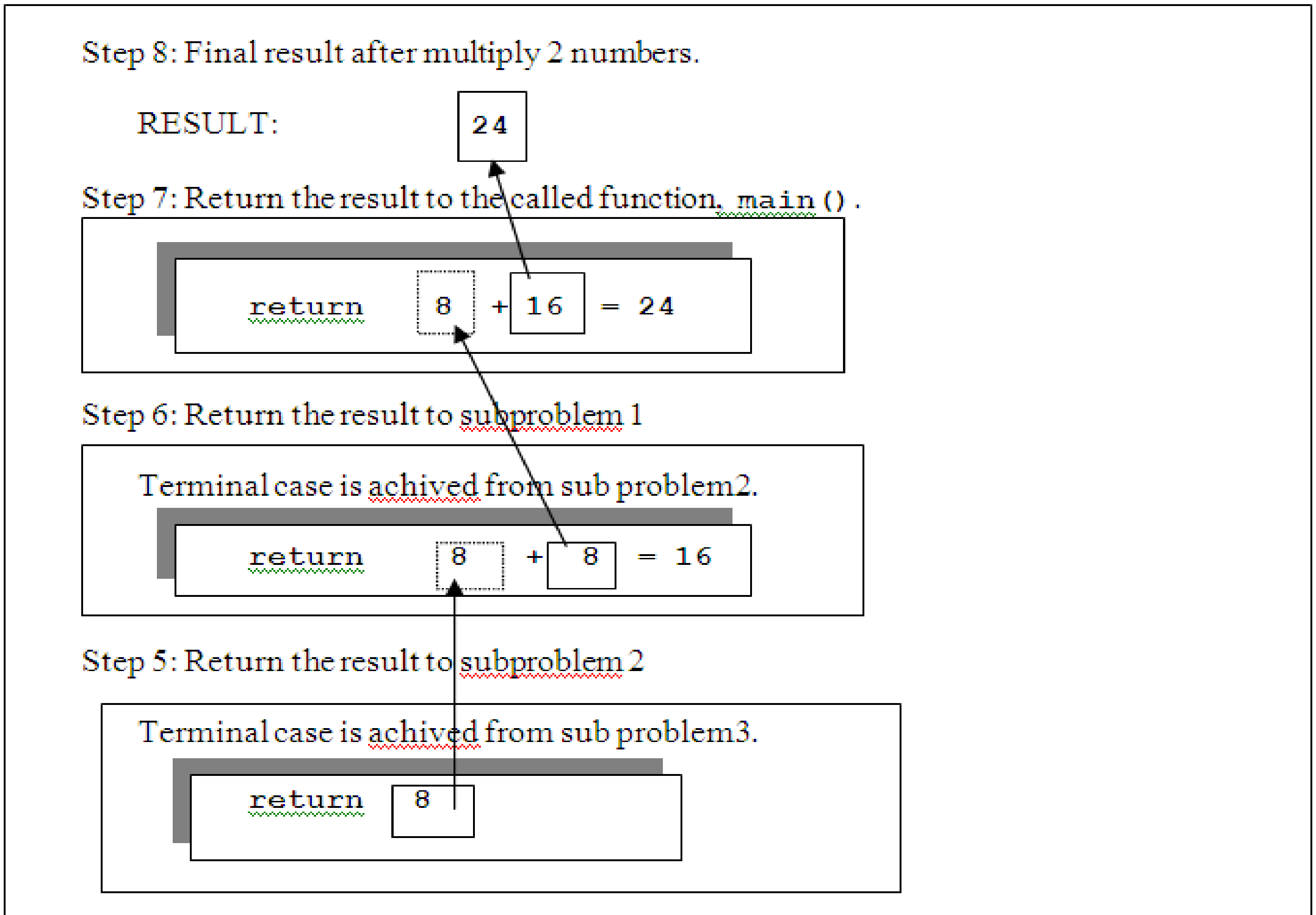
Sub problem3: : int Multiply(int M, int N)

Value of M =8 and N =1.

When N=1, terminal case is achieved.

```
return 8
```

Returning the `Multiply()` result to the called function



Factorial Problem

- Problem : Get Factorial value for a positive integer number.
- Solution : The factorial value can be achieved as follows:

  0! is equal to 1

  1! is equal to 1 x *0! = 1 x 1 = 1*

  2! is equal to 2 x *1! = 2 x 1 x 1 = 2*

  3! is equal to 3 x *2! = 3 x 2 x 1 x 1 = 6*

  4! is equal to 4 x *3! = 4 x 3 x 2 x 1 x 1 = 24*

  N! is equal to N x (N-1)! For every N>0

Factorial function

```
int Factorial (int N )
{ /*start Factorial*/
if (N==0)
   return 1;
else
   return N * Factorial (N-1);
} /*end Factorial
```

Execution of `Factorial(3)`

STEP 1: Get factorial 3.

Problem: `Factorial(3);`

STEP 2: Run `Factorial()`.

Sub problem 1: `int Factorial (int N)`

Value for `N=3`.

Since `N` ≠ `0`, `Factorial()` is called by reducing the parameter value.

```
return N * Factorial(3-1);
```

STEP 3: Run `Factorial()`.

Sub problem 2: `int Factorial (int N)`

Value for `N =2`.

Since `N` ≠ `0`, `Factorial()` is called by reducing the parameter value.

```
return N * Factorial(2-1);
```

STEP 4: Run `Factorial()`..

Sub problem 3: `int Factorial (int N)`

Value for `N =1`.

Since `N` ≠ `0`, `Factorial()` is called by reducing the parameter value.

```
return N * Factorial(1-1);
```

Terminal case for `Factorial(3)`

STEP 5: Run `Factorial()`..

Sub problem 4: `int Factorial (int N)`

Value for `N =1`.

Since `N` = 0, terminal case is achieved.

`return` 1

Execution of `Factorial(3)`

Return value for Factorial(3)

STEP 10: Final result for `Factorial(3)`.

RESULT: 6

STEP 9: Return the result to the called function, `main()`.

Terminal case is achieved for Sub problem 1.

```
return 3 * 2 = 6
```

STEP 8: Return the result to Sub problem 1

Terminal case is achieved for Sub problem 2.

```
return 2 * 1 = 2
```

STEP 7: Return the result to Sub problem 2.

Terminal case is achieved for Sub problem 3.

```
return 1 * 1 = 1
```

STEP 6: Return the result to Sub problem 3.

Terminal case is achieved for Sub problem 4.

```
return 1
```

Infinite Recursive

- Impossible termination condition

- How to avoid infinite recursion:
  - must have at least 1 base case
  - each recursive call must get closer to a base case

Infinite Recursive : Example

```
#include <stdio.h>
#include <conio.h>
void printIntegesr(int n);
main()
{   int number;
    cout<<"\nEnter an integer value :";
    cin >> number;
    printIntegers(number);
}
void printIntegers (int nom)
{   cout << "\Value : " << nom;
    printIntegers (nom);
}
```

1. No condition satatement to stop the recursive call.
2. Terminal case variable does not change.

Conclusion and Summary

- Recursive is a repetitive process in which an algorithm calls itself.
- Problem that can be solved by breaking the problem into smaller instances of problem, solve and combine.
- Every recursive definition has 2 parts:
  - BASE CASE: case that can be solved directly
  - RECURSIVE CASE: use recursion to solve *smaller* sub-problems & combine into a solution to the larger problem

Thank You

http://comp.utm.my/