Recursion

CSCE 110

From James Tam's material
What Is Recursion?

“the determination of a succession of elements (as numbers or functions) by operation on one or more preceding elements according to a rule or formula involving a finite number of steps”
(Merriam-Webster online)
Factorial

• $4! = 4 \times 3 \times 2 \times 1$
• More formally
• $n! = n \times (n-1) \times (n-2) \times \ldots \times 1$
Divide and Conquer Algorithms

• Fractals (self similarity)
  http://www.pbs.org/wgbh/nova/fractals/scal-flash.html
• Binary Search
Recursion In Programming

“a computer programming technique involving the use of a procedure, subroutine, function, or algorithm that calls itself one or more times until a specified condition is met at which time the rest of each repetition is processed from the last one called to the first”

(Merriam-Webster online)
Direct Call

```
module
proc;
begin
  proc ();
end;
```

J. Michael Moore
Indirect Call
Indirect Call

\[ \text{m}^1 \quad \text{m}^2 \quad \text{m}^3 \quad \ldots \quad \text{m}^n \]
procedure proc1;
begin
  :
  proc2;
end;

procedure proc2;
begin
  :
  proc3;
end;

procedure proc3;
begin
  :
  proc1;
end;
An Issue With Indirect Recursion

Example Scenario:

Which one should be defined first?
Procedure Proc1 First?

```
procedure proc1;
begin
  :
  proc2;
  :
end;

procedure proc2;
begin
  :
  proc1;
  :
end;
```

What is proc2?
Procedure Proc2 First?

procedure proc2;
begin
  : proc1;
  : end;

procedure proc1;
begin
  : proc2;
  : end;

What is proc1?
Solution: Use A Dummy Definition

A "placeholder" for the compiler (definition comes later)

Example problem

```pascal
procedure proc1;
begin
  :
  proc2;
  :
end;

procedure proc2;
begin
  :
  proc1;
  :
end;
```

J. Michael Moore
Solution: Use A Dummy Definition

A "placeholder" for the compiler (definition comes later)
Example problem

```pascal
procedure proc2; FORWARD;
procedure proc1;
begin
  :
  proc2;
  :
end;

procedure proc2;
begin
  :
  proc1;
  :
end;
```

J. Michael Moore
Requirements For Recursion

1) Base case
2) Recursive step
Example Program

program factorialProgram (input, output);

function factorial (num :integer):integer;
begin
  if (num = 1) then
    factorial := 1
  else
    factorial := num*factorial(num - 1);
end;

begin
  var number, nFactorial :integer;

  write('Enter the number for factorial :');
  readln(number);
  nFactorial := factorial(number);
  writeln(number, '! = ', nFactorial);
end.
User Types in 3

\[ n! = \text{factorial}(n) \]

factorial (3)
if (3 = 1) then
    factorial := 1
else
    factorial := 3 * factorial(3 - 1);

factorial (2)
if (2 = 1) then
    factorial := 1
else
    factorial := 2 * factorial(2 - 1);

factorial (1)
if (1 = 1) then
    factorial := 1
else
    factorial := 1 * factorial(1 - 1);
When To Use Recursion

• When a problem can be divided into steps.
• The result of one step can be used in a previous step.
• There is scenario when you can stop sub-dividing the problem into steps and return to previous steps.
• All of the results together solve the problem.
When To Consider Alternatives To Recursion

- When a loop will solve the problem just as well
- Types of recursion:
  - Tail recursion
    - A recursive call is the last statement in the recursive module.
    - This form of recursion can easily be replaced with a loop.
  - Non-tail recursion
    - A statement which is not a recursive call to the module comprises the last statement in the recursive module.
    - This form of recursion is very difficult to replace with a loop.
Drawbacks Of Recursion

Function/procedure calls can be costly

- Uses up memory
- Uses up time
Benefits Of Using Recursion

• Simpler solution that’s more elegant (for some problems)
• Easier to visualize solutions (for some people and certain classes of problems – typically require either: non-tail recursion to be implemented or some form of “backtracking”)

J. Michael Moore
Common Pitfalls When Using Recursion

• These three pitfalls can result in a segmentation fault occurring
  • No base case
  • No progress towards the base case
  • Using up too many resources (e.g., variable declarations) for each function call
No Base Case

function factorial (num : integer): integer;
begin
    factorial := num * factorial (num - 1);
end;
No Base Case

function factorial (num : integer): integer;
begin
    factorial := num * factorial (num - 1);
end;

When does it stop???
No Progress Towards The Base Case

function factorial (num : integer): integer;
begin
  if (num = 1) then
    factorial := 1
  else
    factorial := num * factorial (num -1);
end;
Using Up Too Many Resources

procedure proc;
var
    arr : array [1..1000000] of char;
begin
    proc;
    proc;
end;
Undergraduate Definition Of Recursion

Word: re·cur·sion

Pronunciation: ri-'kər-zhən

Definition: See recursion