Programming Techniques

Topic 7 – Recursion

3/2024

Agenda

- ☐ The Nature of Recursion

- ☐ Tracing a Recursive Function

- ☐ Work through Examples of Recursion

Recursion

☐ Recursion is repetition (by self-reference)

■ it is caused when a function calls/invokes itself.

■ Such a process will repeat forever unless terminated by some control structure.

Recursion

□ So far, we have learned about control structures that allow C++ to iterate a set of statements a number of times.

□ In addition to iteration, C++ can repeat an action by having a function call itself.

■ This is called recursion. In some cases it is more suitable than iteration.

Recursion

☐ While recursion is very powerful

- ■ and will allow us to at times simply solve complex problems

- ■ it should <u>not</u> be used if iteration can be used to solve the problem in a maintainable way (i.e., if it isn't too difficult to solve using iteration)

- ■ so, think about the problem. Can loops do the trick instead of recursion?

Recursion

- Why select iteration versus recursion?
  - Efficiency!
  - Every time we call a function a stack frame is pushed onto the program stack and a jump is made to the corresponding function
  - This is done in addition to evaluating a control structure (such as the condition for an if statement) to determine when to stop the recursive calls.
  - With iteration all we need is to check the control structure (such as the conditional expression for the while, do-while, or for)

Recursion

☐ Let's look at a very simple example;

- ■ in this case we can see that by using recursion we can make some difficult problems very trivial...

- ■ many of these problems would be very difficult to solve if you only were able to use iteration.

- ■ *trace through the following problem in class...showing how the stack frame works*

Recursion

☐ What is the purpose of the following?

```cpp
void strange() {
    int t;
    cin >> t;
    if (t != 0){
        strange();
        cout << t << " ";
    }
}

int main(){
    cout <<"Pls input numbers(0-stop)" << endl;
    strange();
    cout << endl;
    return 0;
}
```

Recursion

☐ This program writes the reverse of what was entered at the keyboard, no matter how many numbers were entered!

■ Try to write an equally simple program just using the iterative statements we know about; it would be difficult to make it behave the same without limiting the number of numbers that can be entered or using up a lot of memory with a huge array of numbers!

■ Notice, with recursion, we didn't have to even use an array or list to store data...

Recursion

☐ What happens to this "power" if we had swapped the cout statement with the recursive call in the previous example?

■ It would have simply read and echoed what was typed in.

■ Recursion would be overkill; iteration should be used instead.

Recursion

☐ When a recursive call is encountered, execution of the current function is temporarily stopped.

☐ This is because the result of the recursive call must be known before it can proceed.

☐ So, it saves all of the information it needs in order to continue executing that function later (i.e., all current values of all local variables and the location where it stopped).

☐ Then, when the recursive call is completed, the computer returns and completes execution of the function.

Recursion

☐ In order for your recursive calls to be useful, they must be designed so that your program will ultimately terminate.

☐ As with iteration or looping, there is danger of creating a recursive function that is an infinite loop!

☐ We need to be careful to prevent infinite repetition.

☐ Therefore, when designing a recursive function

  ◼ one of the first steps should be to determine what the <u>stopping condition</u> should be

Recursion

☐ The best way to do this is to use

   ■ an if statement to determine if a recursive call should be made -- depending on the value of some conditional expression.

☐ Eventually, every recursive set of calls should reach a point that does not require recursion (i.e., this will stop recursion).

☐ Recursion should not be used if it makes your algorithm harder to understand or if it results in excessive demands on storage or execution time.

Recursion

- Therefore, there are 3 requirements when using recursion:
  - Every recursive function must contain a control structure that prevents further recursion when a certain state is reached.
  - That state must be able to be reached each time you run the program.
  - When that state is reached, the function must have completed its computation and (if the function returns a value) return the appropriate value for each recursive call. *don't forget to have the function "use" the returned value...if there is one!*

Recursion

☐ In class, walk through the following:

```
int factorial(int n) {
    if (n < 2)
        return 1;
    else
        return (n * factorial(n-1));
}
```

Recursion

☐ In class, walk through the following:

```
int factorial(int n) {
    if (n < 2)
        return 1;
    else
        return (n * factorial(n-1));
}
```

☐ Compare and contrast with the iterative version. Which is better? Why?

Recursion

- ☐ If you request nesting or recursion that goes beyond what your system can handle...you will get an error when you try to execute your program...such as "stack overflow".

- ☐ This simply means that you've tried to make too many function calls - recursively.

- ☐ If you get this error, one clue would be to look to see if you have infinite recursion.

  - ◼ This situation will cause you to exceed the size of your stack -- no matter how large your stack is!

Examples of Recursion

- Two meaningful examples of recursion are the
    - towers of hanoi problem
    - binary search
- Let's discuss each of these and examine:
    - the process they go thru
    - see how recursion helps solve the problem
    - look at the implementation details (of the binary search)
    - discuss the benefits and drawbacks of recursion for these algorithms

Homework

☐ Given a singly linked list, implement a function to print the values of the nodes out in reversed order (i.e. print backward from the last to the first)

  ■ Implement it in 2 ways, one without recursion and one with recursion