Hochiminh City University of Technology
Computer Science and Engineering
[CO1027] - Fundamentals of C++ Programming

Recursive

Lecturer: Duc Dung Nguyen
Credits: 3

Outcomes

- Understand recursive algorithms
- Declare and implement recursive functions

Outline

- Recursion

Recursion

Recursion

- Problem solving methods
  - Principle: divide the big problem into smaller problems
- Recursivity is a property that function have to be called by themselves.
  - Principle: define the solution of big problem using the solution of smaller problems. A set of base solution must be defined

Recursion

- Factorial function: f(n) = n!
  - 0! = 1
  - f(n) = f(n - 1) * n
- Fibonacci sequence is defined as follows
  - F(1) = F(2) = 1
  - F(n) = F(n - 1) + F(n - 2)

Recursive termination

- A recursive termination is a condition that, when met, will cause the recursive function to stop calling itself.
- Factorial function: f(n) = n!
    - 0! = 1 (recursive termination)
    - f(n) = f(n - 1) * n

Example

```
#include<iostream>
using namespace std;

int factorial(int n);

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

int factorial(int n) {
   if (n == 0) return 1;
   return n * factorial(n - 1);
}
```

Example

```cpp
#include<iostream>
using namespace std;

int fibonacci(int n);

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

int fibonacci(int n) {
   if (n <= 2) return 1;
   return fibonacci(n - 1) + fibonacci(n - 2);
}
```

Indirect Recursion

```cpp
#include <iostream>
using namespace std;
int fa(int);
int fb(int);

int main() {
    int num = 5;
    cout << fa(num) << endl;
    return 0;
}

int fa(int n) {
    if (n <= 1) return 1;
    else return n * fb(n - 1);
}
int fb(int n) {
    if (n <= 1) return 1;
    else return n * fa(n - 1);
}
```

Recursion

- Type of recursions
  - Tail recursion: nothing has to be done after the call return
  - Head recursion: the first statement in function is a recursive call
  - Middle / multi-recursion
  - Mutual recursion: function X and Y are mutually-recursive if function X calls function Y, and function Y in turn call function X. This is called indirect recursion

Recursion vs. Iteration

Recursion vs. Iteration

- We can always solve a recursive problem iteratively!
- Iterative functions are almost always more efficient than their recursive counterparts.
- Why do we need recursion?
  - much simpler to write
  - much cleaner and easier to follow

When to choose recursion

- In general, recursion is a good choice when most of the following are true:
  - The recursive code is much simpler to implement.
  - The recursion depth can be limited (e.g. there’s no way to provide an input that will cause it to recurse down 100,000 levels).
  - The iterative version of the algorithm requires managing a stack of data.
  - This isn’t a performance-critical section of code.

Recursion

- More examples
  - Simple: print a string backward
  - Classic: Hanoi tower

Summarise

- Recursion technique