CSI 3334 Data Structures
Homework 0

Yufan Xu

January 14, 2018

Do your own work for this assignment; do not work with others. Consult the book and your professor for help if you need it. Use good grammar, correct spelling, and complete sentences.

1. Define what is meant by the run-time stack, and the run-time heap. Explain how each of them are used.

   Solution:

   — The run-time stack is used for static memory allocation whereas the run-time heap is used for dynamic memory allocation. The run-time stack and the run-time heap are two different kinds of computer memory structures that stores different variables.

   — Variables that being allocated on the stack are directly stored into the memory and it is fast to access this part of the memory; the run-time stack is also directly controlled by the CPU; the run-time stack only stores local variables; the run-time stack also has limited and fixed size.

   — The run-time heap is globally accessible, it has no limited or fixed size until the computer ran out of memory; the run-time heap can be resized; the computer will use virtual memory from hard drives when the memory is full. The access of the run-time heap is slower than the run-time stack; the run-time heap is also fragmented; programmers can use 'new' to allocate and 'delete' to free the memories on heap

2. Using proof by induction, prove that the sum of integers 1, 2, ..., n is equal to n(n+1)/2.

   Solution:

   Base Case:
   Let $n = 1$;
   LHS: $n = 1$;

RHS: $n(n+1)/2 = 1 * (1+1)/2 = 1$;
So LHS = RHS is true.

Inductive Hypothesis:
Let $n = k$;
Assume $1 + 2 + ... + k = k(k+1)/2$ is true.

Inductive Step:
Let $n = k + 1$;
Now prove $1 + 2 + ... + k + (k+1) = [(k+1) * (k+2)]/2$;
LHS: $k(k+1)/2 + (k+1) = [k(k+1) + (2k+2)]/2 = (k^2 + 3k + 2)/2$;
RHS: $[(k+1) * (k+2)]/2 = (k^2 + 3k + 2)/2$;
So LHS = RHS is true.
Which means $1 + 2 + ... + k + k + 1 = [(k+1) * (k+2)]/2$;
So the sum of $1, 2, ..., n = n(n+1)/2$ is true.

3. Name and define the four fundamental rules of recursion.

Solution:

— Base Case: the cases that can be solved without recursion. A base case can stop the recursive function at some point, a recursive function can be executed infinite times if it does not have a base case.

— Making Progress: for each of the recursive function calls must be making progress towards the base case.

— Design Rule: a good recursive function should be well-designed in order to make things workout, it is important that all the recursive function calls work.

— Compound Interest Rule: while using the recursive function calls, the user should never duplicate work by solving the same instance of a problem in separate recursive calls.

After listing them, design a recursive function that follows the four rules and explain how it follows each of them.

Solution:

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

— This recursive function calculates the first 'n' numbers of fibonacci numbers. This function clearly follows the four fundamental rules of recursion. First, it has two base cases in order to terminate the recursive calls and get the result. Second, the function is making progress to the base cases after each recursive call. It also follows the design rule and compound interest rule because in general, the recursive function works perfectly fine as we expected, as well as the user may not duplicate work using this recursive function.

4. What are the two things that define an Abstract Data Type (ADT)?

   Solution:

   — An Abstract Data Type can be defined by a set of values and a set of operations, which means in a case where you need to define a class, member variables and member functions of that class need to be created.

5. What are the "big five" functions that any C++ class must have if it allocates its own memory (on the heap)? Explain why are they needed.

   Solution:

   — Destructor: the destructor is called whenever an object goes out of scope. It is needed because the program needs to free the allocated memory while needed.

   — Copy Constructor: it creates a copy an existing object and assign to another, it is needed for the Left Values in the class.

   — Move Constructor: it creates a copy an existing object and assign to another, it is needed for the Right Values in the class.

   — Copy Assignment Operator =: whenever the user uses = to assign an existing value to another, the operator = is called, it is needed for the Left Values in the class.

— Move Assignment Operator =: whenever the user uses = to assign an existing value to another, the operator = is called, it is needed for the Right Values in the class.

6. Prove that $\sum_{i=1}^{N} i^3 = (\sum_{i=1}^{N} i)^2$

   Solution:

   I decided to use proof by induction.
   Base Case:
   Let $N = 1$;
   LHS: $\sum_{i=1}^{1} i^3 = 1^3 = 1$;
   RHS: $(\sum_{i=1}^{1} i)^2 = (1)^2 = 1$;
   So LHS = RHS is true.

   Inductive Hypothesis:
   Let i = k;
   Assume $\sum_{k=1}^{N} k^3 = (\sum_{k=1}^{N} k)^2$ is true;

   Inductive Step:
   Let i = k + 1;
   Now prove $\sum_{k=1}^{N}(k+1)^3 = [\sum_{k=1}^{N}(k+1)]^2$ is true;
   Because we know that $1+2+3+...+n = n(n+1)/2$;
   So$1^3+2^3+3^3+...+n^3 = [n(n+1)/2]^2$;
   Hence,
   LHS: $[k(k+1)/2]^2 + (k+1)^3 = [k^2(k+1)^2 + 4(k+1)^3]/4 = [(k+1)(k+2)/2]^2$;
   Whereas the
   RHS:$[(k+1)(k+2)/2]^2$;
   So LHS = RHS is true;
   Hence $\sum_{k=1}^{N}(k+1)^3 = [\sum_{k=1}^{N}(k+1)]^2$ is true;
   Then we can get the conclusion that $\sum_{i=1}^{N} i^3 = (\sum_{i=1}^{N} i)^2$ is true.