COSC 2006 – DATA STRUCTURES I FA 2020 – PRACTICE QUIZ 1

Distributed: October 4, 2020

1. The two most important parts of an operation contract, which should be documented in code before the implementation of every function, describe the _____ and the _____.

2. The three fundamental principles of object-oriented design and programming are: (1)_____; (2) _____; and (3) _____.

3. The program component that uses a module or object is known as the module's or object's _____.

4. _____is an older procedural language that is used primarily for scientific and engineering applications.

5. Using the recursive definition of *n*-choose-*k*, i.e. $\binom{n}{k}$, compute $\binom{4}{2}$. Show all work.

6. Complete the code snippet:

```
char a[] = {'a', 'A', '8', …, '9', '#');
nchar = _____; // Number of characters in a
```

7. What is the purpose of `BagInterface` pointer in the function `f` below?

```
double f(BagInterface<ItemType>* somePtr) { . . .}
```

8. Which type of recursive algorithm is ***naturally*** and ***easily*** reformulated as an iterative solution? _____

9. Describe the difference between `++(*p)` and `*(++p)`.

10. Using the closed form equation that was proved by induction in class, compute 1 + 2 + 3 + … + *m*.

11. For 11(a), assume that initially `p` has address 1000, `q` has address 2000, and `x` has address 4000.

(a) What is the result after execution of the following lines of code?

Instruction

```
int *p;
int *q;
p = new int;
*p = 43;
q = p;
*p = 27;
```

	Address	Value
p		
q		
x		

(b) Assuming the results in part (a), what is the result after execution of the following?

Instruction

```
int x;
x = 5*(*p - *q)
+ (*q);
*p = 78;
*q = (*q) + 3 -
(*p);
q = &x;
x = (*p) + (*q);
```

	Address	Value
p		
q		
x		

(c) Assuming the results in parts (a) and (b), what is the result after execution of the following?

Instruction

```
q = new int;
x = x - (*p);
*q = 40;
p = &x;
*p = *q + x;
```

	Address	Value
p		
q		
x		

SOLUTIONS

1. The two most important parts of an operation contract, which should be documented in code before the implementation of every function, describe the **preconditions** and the **postconditions**.

2. The three fundamental principles of object-oriented design and programming are: (1) **encapsulation**; (2) **inheritance**; and (3) **polymorphism**.

3. The program component that uses a module or object is known as the module's or object's **client**.

4. **FORTRAN** is an older procedural language that is used primarily for scientific and engineering applications.

5. Using the recursive definition of *n*-choose-*k*, i.e. $\binom{n}{k}$, compute $\binom{4}{2}$. Show all work.

Recall: $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$

$$\begin{aligned}
\binom{4}{2} &= \binom{3}{2} + \binom{3}{1} \\
&= \binom{2}{2} + \binom{2}{1} + \binom{2}{1} + \binom{2}{0} \\
&= 1 + \binom{1}{1} + \binom{1}{0} + \binom{1}{1} + \binom{1}{0} + 1 \\
&= 1 + 1 + 1 + 1 + 1 + 1 + 1 \\
&= 6
\end{aligned}$$

6. Complete the code snippet:

```
char a[] = {'a', 'A', '8', …, '9', '#');
nchar = sizeof(a) / sizeof(*a); // Number of characters in a
```

7. What is the purpose of `BagInterface` pointer in the function `f` below?

```
double f(BagInterface<ItemType>* somePtr) { . . .}
```

See Sections 4.4 and 4.5 in the text. The most important point that should have strongly stood out is that `BagInterface` is, as its name indicates, an ***interface***, and it does ***not*** implement any methods. This pointer allows all "bag" ADTs whose interface is `BagInterface` to be used as a

parameter of `f`. For instance, with this notation, bags of type `ArrayBag` as well as those of type `LinkedBag` can be sent into `f`. This usage is explained fully in Sections 4.4 and 4.5.

8. Which type of recursive algorithm is ***naturally*** and ***easily*** reformulated as an iterative solution? **tail recursion**

9. Describe the difference between `++(*p)` and `*(++p)`.

`++(*p)` dereferences `p` (through the dereferencing operator *) and increments its value with the increment operator `++`; i.e. what is inside the memory address `p` is incremented.

`*(++p)` dereferences the incremented address `p`; i.e. `p` is an address which is incremented, and the contents of that new address is obtained through the dereferencing operator *.

10. Using the closed form equation that was proved by induction in class, compute 1 + 2 + 3 + … + m.

$$1 + 2 + 3 + \cdots + m = \sum_{i=1}^{m} i = \frac{m(m+1)}{2}$$

11. For 11(a), assume that initially `p` has address 1000, `q` has address 2000, and `x` has address 4000.

(a) What is the result after execution of the following lines of code?

Solution
The ***integer pointers*** `p` and `q` store addresses, and `*p` and `*q` ***dereference*** `p` and `q`, respectively. Prior to the statement `p = new int;`, `p` contains "garbage", but contains the address of a memory location afterwards. The same applies to `q`. However, the specific memory locations ***were not specified***. Therefore, any arbitrary address could be used, along with a brief explanation. The statement `q = p;` equates the values of the two variables, meaning that they both point to the same memory location. The integer variable `x` has not been allocated at this point, and is therefore undefined. At this point, both `p` and `q` point to the same memory location, and, because `*q == 52`, `*p == 52` also (but the question did not ask for `*p` and `*q` – just their values, which are addresses).

Instruction		Address	Value
`int *p;`	p	1000	88888 (arbitrary)
`int *q;`			
`p = new int;`	q	2000	88888 (same as p)
`*p = 43;`			
`q = p;`	x	Not allocated	Undefined
`*q = 52;`			

(b) Assuming the results in part (a), what is the result after execution of the following?

Solution

The integer variable x is now allocated. The pointer q does not change, but p is set to the address of x, which is 4000. After execution of `x = *p - 2*(*q);`, x ← 52 – 2(52) = -52. Subsequently, p ← 78, q ← 78 + 4 = 82, p ← (address of) x, and therefore *p = -52. The last line in this section evaluates as: x ← -52 + 82 = 30. This changes *p to 30 also since p == &x.

Instruction		Address	Value
`int x;`	p	1000	4000
`x = *p - 2*(*q);`			
`*p = 78;`	q	2000	8888 (from above)
`*q = (*p) + 4;`			
`p = &x;`	x	4000	30
`x = (*p) + (*q);`			

(c) Assuming the results in parts (a) and (b), what is the result after execution of the following?

Solution

Note that addresses of the variables do not change. The statement `q = new int;` simply sets q to point to a new memory location (the address of q, 2000, is unchanged). The pointer p still points to the address of x. Therefore, the statement `x = (*p) - x;` sets x to 0. The new memory location pointed to by q is updated to 40. However, the next statement sets q to the address of x. Therefore, `*q == 0`, and x is unchanged at `x == 0`, and at the end of execution, `*p == 0`, since 0 + 0 = 0.

Instruction		Address	Value
`q = new int;`	p	1000	4000
`x = (*p) - x;`			
`*q = 40;`	q	2000	4000
`q = &x;`			
`*p = *q + x;`	x	4000	0