Jaimen Govender PA1 COS214

u20464348

27/07/2022

**Question 1:**

* 1. a is created on the stack, because int a is a local variables and local variables are created and stored at the top of the stack.

b is created on the heap, the new operator requests for memory allocation to be dynamically created on the heap.

c is created on the stack; the char array is stored on the stack because the array is a fixed size during compile time.

d is created on the stack; the integer array created on the stack and is stored on the stack because the array is a fixed size during compile time.

e is created on the stack, because the const integer pointer in on the stack. It is being initialized and created on the stack

f is created on the stack, the initialization is done statically that is why it is created on the stack.

g is created on the stack, because char g is a local variables and local variables are created and stored at the top of the stack.

h is created on the stack, because const integers variables are local it is stored on the stack

n is created on the stack, because const long variables are local it is stored on the stack.

C[10] is created on the stack, the char array is stored on the stack because the array is a fixed size during compile time.

* 1. Assigning the value NULL to variable h will not work because NULL is a special value as it is not a valid pointer value. Null can only be assigned to reference types; NULL cannot be assigned to primitive variables.
  2. Firstly, change the array from char to integer (char c[10]; to int c[10];),

Then, change char g = 2 to char g = ‘2’ as the first one initialized an integer to a char

Lastly, change c[10] = \*&\*e; to c[10] = 522;

**Question 2:**

**2.1)** When a derived class is created from Class A, the constructor for the base class(or superclass 🡺 ClassA) will run first, then the constructor of the derived class.

**2.2)** ClassA destructor is called after the derived classes destructor is called.

**2.3)** ClassC constructor is called after ClassA constructor.

**2.4)** ClassA’s constructor is called first, then ClassB’s constructor is called

**2.5)** First, Class B destructor called,

Then Class A destructor called.

**Task 3:**

**3.2)** This worked, because every parameter was satisfied and arithmetic operations on integer values are allowed.

**3.3)** This worked, because every parameter was satisfied and arithmetic operations on double values are allowed.

**3.4)** This does work when one chains methods. By creating another object or variable to store parts of the equation, the summation of strings is possible.

**3.5)** This is did not work, because multiplication cannot be performed on strings.

**Task 4:**

**4.1)** 1st cout statement, the output is “15 15” this is because \*ptr\_a points to the address of 15 and in the cout, the address is dereferenced (cout << \*ptr\_a) that is why the output of a is 15. B equals 15 because ptr\_b = ptr\_a.

2nd cout statement, the output is “15 4”, the value of ptr\_b changes because ptr\_b initializes a new int that points to the address where value for is. In the cout, the address is dereferenced (cout << \*ptr\_b) that is why the output of B is 4. A remains the same.

3rd cout statement, the output is “15 15” this is because \*ptr\_b = \*ptr\_a, so the address that B points to is overwritten(changes) to the same location that ptr\_a points to and ptr\_a still points to 15 which means ptr\_b also points to the address where 15 is.

4th cout statement, the output is “15 15”, ptr\_a is deleted and reinitialized to equal ptr\_b which points to 15. This means that ptr\_a also points to 15. Ptr\_a’s and ptr\_b’s value are being dereferenced in the cout, that is why the output is 15

5th cout statement, the output is “0x555f864332e0 15”, this is because ptr\_c equals to the address of ptr\_a so in the cout when “\*ptr\_c”, the memory address of ptr\_a is being dereferenced and not the value ptr\_a points to, hence the 0x555f864332e0 address output. The 15 output comes from the double dereferenced, which gets the value of ptr\_a.

**Task 6:**

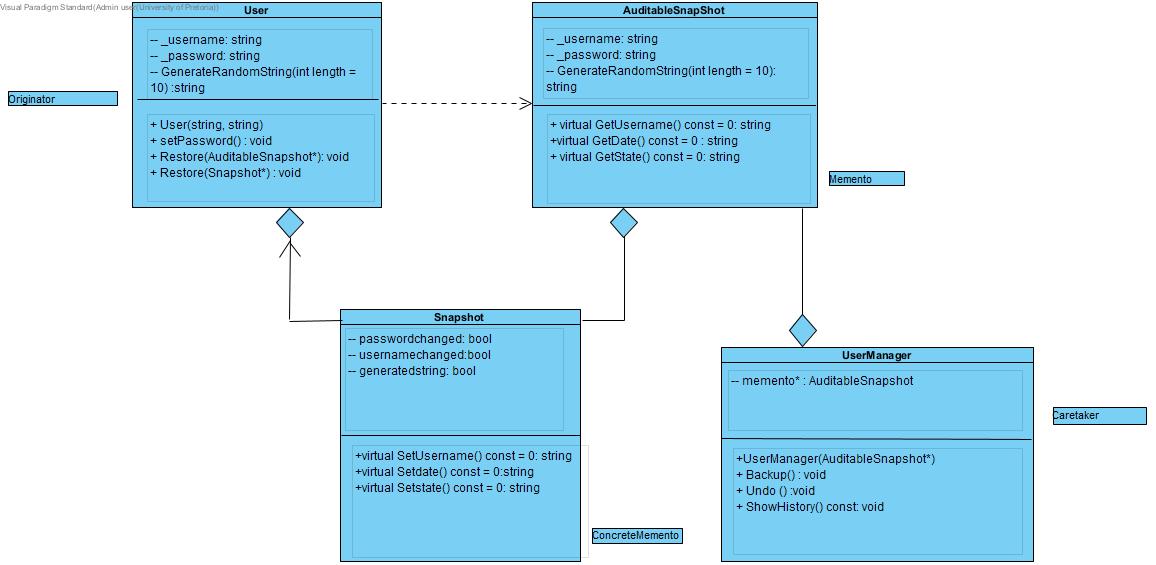
**6.1)** AuditableSnapshot.cpp

**6.2)** Snapshot.cpp

**6.3)** User.cpp

**6.4)** UserManager.cpp

**6.7)**

****