COSC1076 | Advanced Programming Techniques

Revision Questions | Week 03 | Answers

1. To create a user defined type, which may serve as a synonym for:
   - a more complex type
   - the purpose of abstraction.
2. *The type name is your choice*

```
typedef double* DblPtr;
```

3. (a) It is the deconstructor on the `Example` class, that is called when the object is deallocated, and it should clean-up all memory that has been allocated by the class.
   (b)
   (c)
   (d)
   (e) Snippet for questions:

```
Example ex(8)
Example *exPtr = &ex;
ex.noParams();
ex->someParams(1);
```

4. Pointers. The name of a variable serves as a pointer to the first element of the array.
5. Interpretations are:
   (a) Pointer to a pointer to a double
   (b) 2D array
6. (a) The memory for `array` has not been allocated.
   (b) Method 1:

```
double makeSpace[10];
double* array = makeSpace;
array[0] = 1.2;
```

   Method 2:

```
double* array = new double[10];
array[0] = 1.2;
```

7. The header File should contain:

```
// Including length is optional since it may be needed in multiple files
#define LENGTH          5

int foo(int x);
class Example {
public:
    Example(int value);
    ~Example();

    void noParams();
};
```

The code file should contain

```
#include <iostream>

#define EXIT_SUCCESS    0

// Uncomment this if length should only be local to this file
//#define LENGTH          5

using std::cout;
using std::endl;

int void main(void) {
    Example ex(1);
    ex.noParams();

    return EXIT_SUCCESS;
}

int foo(int x) {
   int y = 3;
   return x + y;
}

Example::Example(int value) {
}

Example::~Example() {
}

void noParams() {
}
```

8. Types:
   (a) Loaded program code storage
   (b) Program Call Stack
   (c) Heap
9. Purposes:
   (a) Store the currently executing program in memory so the computer can run the program
   (b) Automatic (program) manage memory for local variable and function calls
   (c) User managed dynamically allocated memory for transient data
10. The complete call-stack is below. Stacks are drawn from bottom-to-top.

After line 9:

`b`	8
`a`	7

After line 10:

`b`	8
`a`	10

After line 18 (during call to `fnA`):

`tmp`	0
`value`	`&b`
return addr. to `main`	0x00...
`b`	8
`a`	10

After line 24 (during call to `fnB`):

`calc`	0
`dbl`	10
return addr. to `fnA`	0x00...
return value of `fnB`	*unknown*
`tmp`	0
`value`	`&b`
return addr. to `main`	0x00...
`b`	8
`a`	10

After line 25 (during call to `fnB`):

`calc`	20
`dbl`	10
return addr. to `fnA`	0x00...
return value of `fnB`	*unknown*
`tmp`	0
`value`	`&b`
return addr. to `main`	0x00...
`b`	8
`a`	10

At line 19 (after return from `fnB`):

~~`calc`~~	~~20~~
~~`dbl`~~	~~10~~
~~return addr. to `fnA`~~	~~0x00...~~
return value of `fnB`	20
`tmp`	20
`value`	`&b`
return addr. to `main`	0x00...
`b`	8
`a`	10

After line 19 (after return from `fnB`):

~~`calc`~~	~~20~~
~~`dbl`~~	~~10~~
~~return addr. to `fnA`~~	~~0x00...~~
~~return value of `fnB`~~	~~20~~
`tmp`	20
`value`	`&b`
return addr. to `main`	0x00...
`b`	8
`a`	10

After line 12 (after return from `fnA`):

~~`calc`~~	~~20~~
~~`dbl`~~	~~10~~
~~return addr. to `fnA`~~	~~0x00...~~
~~return value of `fnB`~~	~~20~~
~~`tmp`~~	~~20~~
~~`value`~~	~~`&b`~~
~~return addr. to `main`~~	~~0x00...~~
`b`	8
`a`	20

11. Code snippet:

```
    double * dbl = new double(0);
    char* string = new char[12];
    string = "hello world";
    delete dbl;
    delete[] string;
    Example *ex = new Example(2);
    delete ex;
```

12. Look at the below program that implements a class. What is the error in this code?
    (a) The deconstructor does not clean-up (that is, delete) the memory that was allocated for the private variable `ptr`.
    (b) The deconstructor should be:

```
ContainsError::~ContainsError() {
    delete ptr;
}
```