CISP 400 - Lab Assignment #3 (part 1)
Due Wednesday, September 9, 2018 at 11:59 PM

We are going to be writing a program that accepts information about purchased real estate. It will take the input from the user and then repeats that information to the user. However, we are going to write this program using structs, enums, and memory allocated with pointers. We are also going to be using constants, and we'll be storing some data in some string objects as well.

#1 – Enum named Property

Define an enum, named Property. The enum Property should contain the following enumerators: Lakeside, Downtown, Suburb, Country, Parkside. Lakeside should equal 0 and the others should successively increase in value, 1 each, e.g. Downtown should equal 1, etc.

#2 – Constant EXIT_OPTION and constant MAX_NAME_LENGTH

Define a constant int, named EXIT_OPTION, which has an option that is the value of Parkside + 1. You can just enter this as a number, rather than trying to do a calculation off of Parkside.

Define a second constant named MAX_NAME_LENGTH and set it to 20.

#3 – Struct OwnedPropertyStruct

Define a struct, named OwnedPropertyStruct. It should contain an integer named 'transactionNumber', a string (an instance of the C++ string CLASS, not a char, char pointer, or char array) named 'owner', and a Property enumeration variable named 'which';

#4 – Type OwnedProperty

Use typedef to define a new type, OwnedProperty, from OwnedPropertyStruct. Remember to look up typedef in the lecture slides if you have trouble.

#5 – Function TranslateProperty

Write a function named TranslateProperty. It should have the following prototype:

```
string TranslateProperty(Property translationData);
```

It should return – **NOT** print out, return – a descriptive string for each enumerator in Property, and return a string that says Error if the variable does not match any enumerator in Property. This last condition should never happen, but it's always good to leave some error handling code in this sort of place, just in case something highly unexpected happens.

#6 – Function userMenu()

Write a function named userMenu() that returns an int. It should present a user a choice of several numbers, one each for the enumerators in Property. The options shown should match the int value of the enumerator.

For instance, for Lakeside, it should present:

```
0 - Lakeside
```

And for Downtown, it should present:

```
1 - Downtown
```

And so forth. It should also list the constant, EXIT_OPTION, as a possible option.

Have the user select between these. Each option should be on a newline in the menu.

The function should repeat until the user enters a valid number, so if a user enters 55 (which is invalid in this program) it should inform them that this is an invalid choice and ask them to select an option again.

This function should return the user's choice. Remember, you are asking for an **int**, not anything of type char or string. You are also not looking to return anything of the enum type Property, we'll be translating that elsewhere.

#7 – Function printOwnedProperty

Write a function named printOwnedProperty(OwnedProperty *printableData). It should print out the contents of the instance of OwnedProperty that the pointer parameter is pointing to, labeling each field appropriately. So if you print out the 'owner' field, it should print out something like:

```
Owner - Anne
```

if the owner field contains the string "Anne". Remember - you are passing a POINTER! So use the -> operator to access the data.

In order to print out the 'which' field in the OwnedProperty struct, use the translateProperty function we defined earlier.

#8 – Function handlePropertyData

Write a function named handlePropertyData. The prototype is:

void handlePropertyData(int userOption);

This function is a complex one as it’s going to do the data handling, much as the name suggests. If this was a real program it would probably need a more descriptive name. Fortunately, it isn’t, so we’ll just focus on making it do what we want it to.

It should take the user's choice (from the main menu) as an integer parameter. It should use static_cast to translate userOption into a variable of the enumerator type, Property. We'll call that variable purchasedProperty.

There should be, as a variable, a pointer of type OwnedProperty, we'll call that purchaseData.

This function should allocate an instance of OwnedProperty. Remember we ONLY NEED ONE, so don't use the square brackets to allocate an array or multiples of it.

Next, we set the 'which' field in the OwnedProperty pointed to by purchaseData to the user's choice. Remember we used static_cast to turn that into a Property enum? Use that casted variable here.

Now, ask the user for the transaction number and the owner’s name. Make sure that the name entered for the owner is no more than MAX_NAME_LENGTH long. If it is too long then print an error message and ask the user to try to enter another name, and continue asking until you get a valid name.

Both the name and the transaction number should be stored using the purchaseData pointer.

purchaseData should then be passed to printOwnedProperty.

Now, we are done with purchaseData, so deallocate the memory.

Then we can exit the function.

#9 – Tie it all together

Finally, to tie it all together, write a main() function. This function should:

- Use a loop to continue running until the user indicates they wish to quit.
- Get the user’s choice of options.
- check to see if the option choice is EXIT_OPTION. If so, then we can exit the loop and end the program.
- If not, then the main program should call handlePropertyData, with the user's option as a parameter.
- At this point, if we do not want to continue the program, end the loop and exit.

TURN IN:

Turn in the .cpp file you have written by the due date.