1 My Dog Ate My Budget [ /42]

Alyssa P. Hacker is helping her friend Ben Bitdiddle manage expenses so he can survive on a meager graduate TA stipend. IMPORTANT: Read through all 4 pages of this problem before beginning your implementation. Here’s an example main.cpp showing how Ben will record and manage his expenses:

```cpp
#include "budget.h"
int main() {
    Budget my_budget;
    my_budget.addExpense("home", Expense("water", 20));
    my_budget.addExpense("food", Expense("groceries", 100));
    my_budget.addExpense("home", Expense("rent", 900));
    my_budget.addExpense("other", Expense("movie tickets", 14));
    my_budget.addExpense("home", Expense("electricity", 93));
    my_budget.addExpense("food", Expense("pizza", 7));
    my_budget.print();
}
```

And here’s the nicely formatted output that will be printed when the program is run:

Note how the categories are alphabetized and the costs are sorted.

```
category: food $ 107
groceries $ 100
pizza $ 7
category: home $ 1013
rent $ 900
electricity $ 93
water $ 20
category: other $ 14
movie tickets $ 14
TOTAL $ 1134
```

Before she had to leave, Alyssa wrote out the whole program on the whiteboard. After she left, Ben’s overly friendly (and very fluffy) dog, Muffit, erased a bunch of the code. Help Ben by filling in the blanks.

1.1 Expense Class Declaration (expense.h) [ /8]

sample solution: 11 line(s) of code
1.2 Expense Class Implementation (expense.cpp) [ /2]

```cpp
#include "expense.h"

void Expense::print() const {
    std::cout << std::left << std::setw(18) << name << " $"
               << std::right << std::setw(5) << cost << std::endl;
}

byCost() {
    return (a.getCost()>b.getCost()) || (a.getCost()==b.getCost() && a.getName()<b.getName());
}
```

1.3 BudgetCategory Class Declaration (budget_category.h) [ /10]

```cpp
#include "expense.h"
```

(sample solution: 13 line(s) of code)
#include "budget_category.h"

void BudgetCategory::print() const {
    std::cout << "category: " << std::left << std::setw(10) << name << "$"
    << std::right << std::setw(5) << getTotal() << std::endl;
    for (int i = 0; i < data.size(); i++) {
        data[i].print();
    }
}

addExpense() {

}

byName() {
    return a.getName() < b.getName();
}

1.5  Budget Class Declaration (budget.h) [ /5]

#include "budget_category.h"

sample solution: 3 line(s) of code

sample solution: 7 line(s) of code
1.6 Budget Class Implementation (budget.cpp)

```cpp
#include "budget.h"

void Budget::print() const {
    int total = 0;
    for (int i = 0; i < categories.size(); i++) {
        categories[i].print();
        total += categories[i].getTotal();
    }
    std::cout << std::left << std::setw(20) << "TOTAL" << "$"
              << std::right << std::setw(5) << total << std::endl;
}

addExpense( ) {
}
```

1.7 Budget Compilation

Assuming the current directory contains these 7 files, what UNIX/GNU Linux command(s) would you type to compile and run this program?
Write a function named `suggest_words_vertical` that takes four arguments: a partially completed crossword puzzle board and a dictionary of words (both represented as vectors of strings), and two integers that specify a row and column location in the board. Similarly to Homework 1, the board will use the ‘#’ character to represent a black square. The other squares will either have a lowercase letter ‘a’-‘z’ or a ‘?’ character which indicates the square needs to be filled in. Your function should return a vector of all legal words from the dictionary that could fit as the vertical word starting at the specified row and column. Note: Your function doesn’t need to worry about the horizontal words or error checking.

For example, given the board above, the call `suggest_words_vertical(board,dictionary,1,1)` returns `tide tike tile time tine tire` and `suggest_words_vertical(board,dictionary,0,3)` returns `shoe shop shot show slob slog slot slow smog snob snow soon soot spot stop stow`. 

*sample solution: 20 line(s) of code*
For each of the memory diagrams below, write a fragment of code that will produce the diagram.

Sample solution: 10 line(s) of code

Sample solution: 4 line(s) of code

Sample solution: 9 line(s) of code
Complete the code fragment below to implement a simplified notification system for a messaging platform similar to Twitter. The input file is formatted as multiple messages. Each message consists of the sender, the size of the message (number of words), and the words of the message. A sample of the data is shown above. Note that messages may wrap over multiple lines. After reading each message you will send notifications to everyone tagged with an '@' in the message body. Here is the expected output:

Sally->Fred: 'I just ate lunch with @Fred at Bella's and it was delicious.'
Fred->Sally: 'Mmmm... food... good! @Sally @Joe'
Fred->Joe: 'Mmmm... food... good! @Sally @Joe'
Joe->Fred: 'Hey @Fred why didn't you invite me?'
Fred->Sally: '@Sally next time let's call Joe!'
In this problem we will search a large ASCII Art canvas for matches to a target pattern. For example, given the ASCII Art Tic Tac Toe game on the right, we can search for the 'X' symbol on the board. Or the 'O' symbol. Or the height=5 column of '|' characters that make the vertical bar of the Tic Tac Toe board. You will write a function named `search_for_symbols` that takes in 2 arguments, `canvas` and `symbol`. Both inputs are STL vectors of STL strings. The function should return all locations of that pattern or symbol on the canvas. The return type will be an STL vector of positions where each position is an STL vector of integers. Note: Your function program should be general, and work with any size canvas and any size symbol.

In the example below we search for all of the 'O' symbols on the board:

```cpp
std::vector<std::string> symbol;
symbol.push_back(" *** ");
symbol.push_back("* * ");
symbol.push_back("* * ");
symbol.push_back("* * ");
symbol.push_back(" *** ");
std::vector<std::vector<int> > positions = search_for_symbols(tic_tac_toe_canvas,symbol);
for (int i = 0; i < positions.size(); i++) {
    assert (positions[i].size() == 2);
    std::cout << "An O is located at: " << positions[i][0] << "," << positions[i][1] << std::endl;
}
```

And this is the expected output:

```
An O is located at: 0,6
An O is located at: 12,12
```

5.1 Assumptions & Error Checking & Realistic Expectations [ /5]

You will implement the `search_for_symbols` function on the next page. You will not need to do any error checking in your function. But let's pause and consider what assumptions you are making about the input and output. Write 3-4 well-written and concise sentences. You may find it easier to write the function on the next page, then return to answer this part.
5.2 Now Implement the `search_for_symbols` Function

*sample solution: 22 line(s) of code*
6 Stack Sleuthing [ /15]

Carefully examine the contents of the stack after program execution shown on the right.
Your task for this problem is to complete the source code on the left so that it produces this stack.

```cpp
1 { 
2 
3 if ( < ) 
4 
5 else 
6 
7 } 
8 ) 
9 ( } 
10 std::cout << tmp << std::endl;
11 
12 
13 
14 return address main@line13 
function main 
answer: 42 
15 x: 8 
y: 3 
tmp: 42 
b: 3 
a: 8 
return address foo@line6 
function foo 
answer: 42 
16 a: 7 
return address foo@line6 
function foo 
answer: 42 
17 b: 5 
a: 7 
return address foo@line6 
function foo 
answer: 42 
18 b: 7 
a: 6
```
7 Common C++ Programming Errors

For each code fragment below, choose the letter that best describes the program error. *Hint: Each letter will be used exactly once.*

A) Accessing data beyond the array bounds  
B) Uninitialized memory  
C) Infinite loop  
D) Compile error: type mismatch  
E) Does not contain an error  
F) Memory leak  
G) Math error (incorrect answer)  
H) Syntax error

```
int* apple;
int banana[5] = {1, 2, 3, 4, 5};
apple = &banana[2];
*apple = 6;
```

```
std::vector<std::string> temperature;
temperature.push_back(43.5);
```

```
float* floating_pt_ptr = new float;
*floating_pt_ptr = 5.3;
floating_pt_ptr = NULL;
```

```
unsigned int x;
for (x = 10; x >= 0; x--) {
    std::cout << x << std::endl;
}
```

```
int balance = 100;
int withdrawal;
std::cin >> withdrawal;
if (withdrawal <= balance)
    balance -= withdrawal;
else
    std::cout << "failure\n";
```

```
int grades[4] = { 1, 2, 3, 4 };
std::cout << "grades" << grades[1]
<< " " << grades[2]
<< " " << grades[3]
<< " " << grades[4]
<< std::endl;
```
In this problem you will implement a simple class named RunTracker to keep track of an athlete's running workouts. IMPORTANT: Read through all 3 pages of this problem before beginning your portion of the implementation.

First we create a number of runners, and store their recent runs by distance (in miles):

```cpp
RunTracker george("George", "Smith");
RunTracker sally("Sally", "Williams");
RunTracker chris("Chris", "Jones");
george.addRun(6.7); sally.addRun(3.1); chris.addRun(2.5); george.addRun(5);
sally.addRun(13.1); chris.addRun(2); george.addRun(9.6); chris.addRun(26.2);
sally.addRun(13.1); george.addRun(11.7);
```

```cpp
std::vector<RunTracker> runners;
runners.push_back(george); runners.push_back(sally); runners.push_back(chris);
```

We'd like to count how many times each runner ran a specific distance (or more). For example:

```cpp
std::cout << "Runners who have completed the half marathon distance: " << std::endl;
for (unsigned int i = 0; i < runners.size(); i++) {
    int num = runners[i].numRunsAtLeast(13.1);
    if (num >= 1)
        std::cout << " " << runners[i].getName() << " (" << num << ")" << std::endl;
}
```

Should result in this output:

```
Runners who have completed the half marathon distance:
Williams, Sally (2)
Jones, Chris (1)
```

Finally, we'd like to print all runners, ordered by total distance completed. The following code:

```cpp
/* ONE LINE OF CODE OMITTED: YOU WILL FILL IN THIS LINE IN PART 1 */
for (unsigned int i = 0; i < runners.size(); i++) {
    std::cout << std::setw(18) << std::left << runners[i].getName() << " 
    << std::setw(4) << std::right << std::fixed << std::setprecision(1)
    << runners[i].totalDistance() << std::endl;
}
```

Should result in the following output:

```
Williams, Sally 29.3
Jones, Chris 30.7
Smith, George 33.0
```

8.1 Omitted Line of Code

What is the missing line of code above? It may depend on your answers on the next page(s).
8.2 RunTracker Class Declaration [ /14]

Using the sample code on the previous pages as your guide, write the class declaration for the RunTracker object. That is, write the header file (run_tracker.h) for this class. You don’t need to worry about the #include lines or other pre-processor directives. Focus on getting the member variable types and member and non-member function prototypes correct. Use const and call by reference where appropriate. Make sure you label what parts of the class are public and private. Save the implementation of all functions for the run_tracker.cpp file, which is the next part.

sample solution: 13 line(s) of code
8.3 RunTracker Class Implementation [ /16]

Now implement all of the functions prototyped in the run_tracker.h file, as they would appear in the corresponding run_tracker.cpp file.

*sample solution: 25 line(s) of code*
The code below is missing its variable declaration & initialization. Fill in these lines so the code compiles and runs without syntax errors, memory errors or leaks. Show your work by diagramming the memory.

```c
int main() {
    c[2][0] = 'R';
    (*a)[0] = NULL;
    (*b)[2][1] = 'P';
    c[1] = NULL;
    *(d[1]) = 'I';
    d[0] = NULL;

    // CREATE MEMORY
    // DIAGRAM

    delete [] (*a)[2];
    delete b;
    delete [] c;
    delete d[1];
}
```
In this problem you will implement a simple class named Family to keep track of the children and pets that are members of a family. All members of a family have a common last name. IMPORTANT: Read through all 4 pages of this problem before beginning your portion of the implementation.

Here’s a simple example using the Family class:

```cpp
Family king("King");
king addChild("Chris");
king addPet("Buddy");
king addChild("Sally");
std::cout << "The " << king.lastName() << " family has "
    << king.numChildren() << " children."
if (!king.isPet("Socks")) {
    std::cout << "The family does not have a pet named Socks."
}
king.print();
```

And here’s the output from this code:

```
The King family has 2 children.
The family does not have a pet named Socks.
King Family
    children: Chris Sally
    pets: Buddy
```

We’ll also parse data on the children and pets of multiple families from a file. For example if the input file family_input.txt contains:

```
child Alice Williams
child Ellen Davis
child Frank Jones
pet Garfield Davis
child Henry Williams
pet Mittens Brown
child Ryan Jones
pet Spot Jones
pet Tweety Davis
```

We will use the Family class to organize, sort, and print this output:

```
Jones Family
    children: Frank Ryan
    pets: Spot
Williams Family
    children: Alice Henry
Davis Family
    children: Ellen
    pets: Garfield Tweety
Brown Family
    pets: Mittens
```

Note that the children and pets are grouped by last name. The families with the most children are printed first. Families with the same number of children are ordered by last name.
10.1 Using the Family Class [15]

Complete this fragment of code to read the input file and produce the output on the previous page.

```cpp
std::string filename = "family_input.txt";
std::ifstream istr(filename);
if (!istr.good()) {
    std::cerr << "ERROR: could not open " << filename << std::endl;
    exit(1);
}

std::sort(families.begin(), families.end());
for (int i = 0; i < families.size(); i++) {
    families[i].print();
}
```

Sample solution: 19 line(s) of code
Using the sample code on the previous pages as your guide, write the class declaration for the Family object. That is, write the header file (family.h) for this class. You don’t need to worry about the #include lines or other pre-processor directives. Focus on getting the member variable types and member and non-member function prototypes correct. Use const and call by reference where appropriate. Make sure you label what parts of the class are public and private. Save the implementation of all functions for the family.cpp file, which is the next part.

sample solution: 15 line(s) of code
Now implement all of the functions prototyped in the `family.h` file, as they would appear in the corresponding `family.cpp` file. \textit{NOTE: You may omit the implementation of the print() function.}

\textit{sample solution: 19 line(s) of code}
Write a function named `print_square` that takes in a single argument, an STL string, and reformatst that text to fit in the *smallest square box*, surrounded by a border of stars. Unlike Homework 1, we won’t worry about fitting complete words or hyphenation. Just break the words when you get to the end of the row. A few sample calls to the function are shown below, and the output to `std::cout` of each call is shown on the right.

```cpp
print_square("Here is an example.");
print_square("the quick brown fox jumped over the lazy dogs");
print_square("Twinkle, twinkle, little star, how I wonder what you are. Up above the " +
  "world so high, like a diamond in the sky.");
```

**sample solution:** 18 line(s) of code
Write code to produce the memory structure shown in the diagram to the right.

Sample solution: 12 line(s) of code

Write code to print the current year to `std::cout` using ALL of the variables (a, b, c, and d).

Sample solution: 1 line(s) of code

Finally, write code to clean up the dynamically-allocated memory so we don’t have any leaks.

Sample solution: 3 line(s) of code
13 HasLetter [ /12]

For this problem you will write a function named HasLetter that accepts 3 arguments named words, letter, and selected. The function should examine the strings in the words vector, collecting all strings that contain the character letter in the selected vector.

If words contains these 8 words: dog bird cat fish turtle horse goat hedgehog
Then after executing this command:

    HasLetter(words,'r',selected);

The selected vector will contain: bird turtle horse

If we then execute:

    HasLetter(words,'o',selected);

Now the selected vector will contain: dog horse goat hedgehog

sample solution: 14 line(s) of code