CSCI-1200 Data Structures
Test 2 — Practice Problems

Note: This packet contains selected practice problems from Test 2 from three previous years.
Your test will contain approximately one third to one half as many problems (totalling ∼100 pts).

1 Mock Interview Practice [28 pts]

Ben Bitdiddle is prepping for a coding interview and asks his Data Structures mentor Jenay for help. *Please read through the entire question before working on any of the subproblems.* Jenay suggests Ben tackle this problem from lecture: "`Write code to remove duplicates from a sequence of words.`"

Ben writes a function `remove_dups_1` that takes in an STL `vector` of STL `string`s and returns the number of words that were removed. For this input:

```
the quick brown fox jumped over the lazy brown dogs and also jumped over the lazy penguin
```

Calling Ben's first draft function returns '6' and the `vector` now contains:

```
also and brown dogs fox jumped lazy over penguin quick the
```

Jenay observes that while sorting the data makes the program run fast, unfortunately the likely intention was to preserve the order within the original data.

1.1 Quick but Flawed [14 pts]

Write code that matches the description of Ben's `remove_dups_1` function:

sample solution: 19 line(s) of code

If the input has n words and r words are removed, what is the Big O Notation of `remove_dups_1`?

1.2 Preserving the Sequence [14 pts]

For his second draft, `remove_dups_2`, Ben starts over from scratch. The function has the same prototype, but based on Jenay's feedback it now preserves the original order of the data. So for this input:

```
the quick brown fox jumped over the lazy brown dogs and also jumped over the lazy penguin
```

The function returns 6 and the `vector` now contains:

```
the quick brown fox jumped over lazy dogs and also penguin
```

Jenay's feedback about `remove_dups_2` is that he's using the `erase` function, which will negatively impact the performance of this code. Write code that matches the description of Ben's `remove_dups_2` function:

sample solution: 16 line(s) of code

If the input has n words and r words are removed, what is the Big O Notation of `remove_dups_2`?

Ben copies his code for `remove_dups_2` to a new function. The only difference for `remove_dups_3` is that he search-and-replaces '`vector`' with another STL container. The program produces the same answer but now runs faster. What's the replacement container? What is the Big O Notation for `remove_dups_3`?

2 Clown Car Data Structures [25 pts]

Write a function named `clowncar` that takes in two arguments, one of type `Vec` named `a` and the other of type `dslist` named `b`. *In lecture and lab we talked about the implementations of these two "homemade" versions of our favorite STL containers.* The function should swap the data stored in these two structures so that after the call `a` contains the sequence of values that was in `b` and `b` contains the data that was in `a`.

The `clowncar` function has been appropriately added as a friend function of both `Vec` and `dslist` so that it can directly access the private member variables of both classes. Your implementation of `clowncar` **SHOULD NOT call any other functions (including member functions of Vec or dslist) and it SHOULD NOT use iterators**. We want to see you directly edit the member variables and work with the dynamic memory. As a reminder, here are the private member variables of the relevant classes:

Vec

```
T* m_data;
size_type m_size;
size_type m_alloc;
```

Node

```
T value_;
Node<T>* next_;
Node<T>* prev_;
```

dslist

```
Node<T>* head_;
Node<T>* tail_;
unsigned int size_;
```

2.1 Drawing [9 pts]

First, make a detailed memory diagram of sample input to the function: `a` stores 2 even integers (6 & 8) and `b` stores 3 odd integers (15, 17, & 19). Next, neatly edit this diagram to show what will happen when you call `clowncar`. Instead of erasing, lightly cross out things that are changed (allowing us to legibly grade the diagram both before & after the call). Your diagram should match the code you write on the next page. Be sure to include any temporary variables, and all allocations and deallocations of memory.

2.2 Implementation [16 pts]

Now implement the `clowncar` function. Ensure your function does not have any memory errors or leaks.

sample solution: 29 line(s) of code

3 "Missing" `dslist` Iterator Operators [14 pts]

Louis B. Reasoner is working on a group project and his teammates are upset that the project code below doesn't compile. They claim something must be wrong with `dslist`!

```
dslist<std::string>::iterator itr = sentence.begin() + 1;
dslist<std::string>::iterator itr2 = itr + 4;
assert (!(itr2 < itr));
while (itr < itr2) {
  std::cout << *itr << " ";
  ++itr;
}
std::cout << std::endl;
```

Louis tries to explain that `dslist` is fine. That this code wouldn't work even if they switched to the STL `list` class. He suggests they modify the lines that do not compile to use these functions:

```
list_iterator<T>& operator++() { ptr_ = ptr_->next_; return *this;  }
bool operator!=(const list_iterator<T>& r) const { return ptr_ != r.ptr_; }
```

Unfortunately, Louis is unable to convince his teammates to change the project code, and with the deadline fast approaching Louis instead modifies the `dslist` implementation to make the project code above work. What two operator member functions does Louis add to the `list_iterator` class? Write these two functions as they would appear in the class declaration. *Note: You may break the course rule discouraging multiple line functions inside the class declaration.*

sample solution: 16 line(s) of code

4 Debugging Skillz [/ 14]

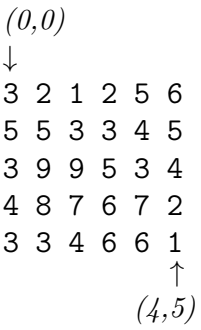
For each program bug description below, write the letter of the most appropriate debugging skill to use to solve the problem. Each letter should be used at most once.

A) get a backtrace
B) add a breakpoint
C) use step or next
D) add a watchpoint
E) examine different frames of the stack
F) reboot your computer
G) use Dr Memory or Valgrind to locate the leak
H) examine variable values in gdb or lldb

	A complex recursive function seems to be entering an infinite loop, despite what I think are perfect base cases.
	The program always gets the right answer, but when I test it with a complex input dataset that takes a long time to process, my whole computer slows down.
	I'm unsure where the program is crashing.
	I've got some tricky math formulas and I suspect I've got an order-of-operations error or a divide-by-zero error.
	I'm implementing software for a bank, and the value of a customer's bank account is changing in the middle of the month. Interest is only supposed to be added at the end of the month.

Select one of the letters you did not use above, and write 3-4 well-written sentences describing of a specific situation where this debugging skill would be useful. You are encouraged to describe a personal anecdote.

5 It's all Downhill from Here! [16 pts]



Write a *recursive* function named `downhill` that takes in 4 arguments: `grid`, `start`, `end`, and `path`. It searches the 2D `grid` of elevations, an STL `vector` of STL `vector`s of integers, for a path from a `start` location to an `end` location. Each step along the path can go up, down, left, or right, but each step must have a lower elevation value than the current position. If it finds a valid downhill path from start to end, the function stores the path (an STL `vector` of locations) and returns true, otherwise it returns false.

```
// A Location on the grid
class loc {
public:
  loc(int r, int c) :
    row(r), col(c) {}
  int row;
  int col;
};
```

For the example shown above right, if start is (3,1) and end is (0,2), then this is a valid downhill path:

```
(3,1) (3,2) (3,3) (2,3) (1,3) (0,3) (0,2)
```

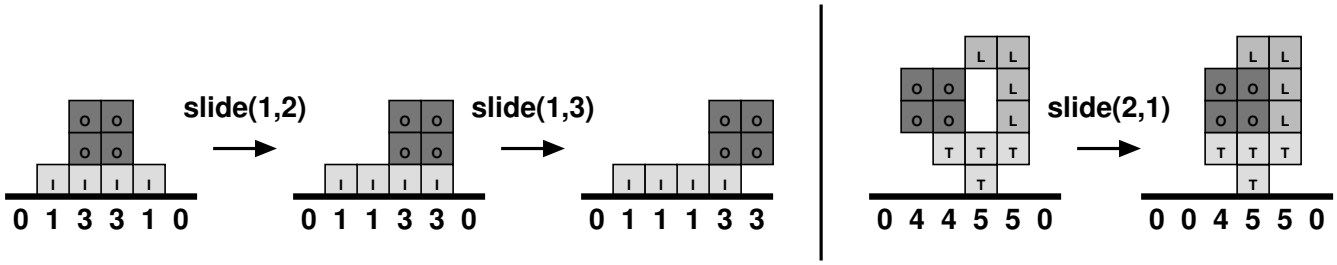
Note: There may be multiple valid downhill paths from start to end, and your function may choose any of these valid paths.

sample solution: 16 line(s) of code

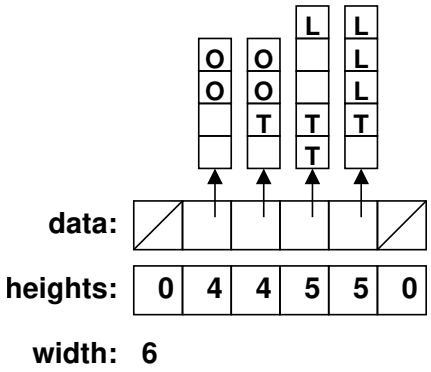
6 The Dynamic Tetris Slide [35 pts]

Our implementation of the Tetris game for Homework 3 only allowed pieces to drop vertically. The full game rules also allow pieces to move horizontally, which can be used by a skilled player to tuck in underneath an "overhang". In this problem we will extend our solution with the `slide` function that allows the square piece, the '`O`' piece, to *slide* one space to the right. *For this problem you don't need to worry about sliding any other piece shape, or about sliding to the left.*

Below are two example Tetris games showing how this function works.



The representation for the `Tetris` class consists of 3 private member variables: `data`, `heights`, and `width`. The memory layout for the 4th diagram above is shown to the right. Remember that we must maintain the arrays to be exactly as long as necessary to store the blocks on the board. The space character is used to represent empty air underneath a block.



The `slide` function takes in 2 integers, the row and column of the lower left corner block of the square '`O`' piece that we want to slide.

We will also implement the `can_slide` function which first tests whether a piece is able to slide to the right. It will return *false* if the 'O' piece at the specified row and column is already at the right edge of the board, e.g., calling `can_slide(1,4)` in the third image above returns false. It will return *false* if the 'O' piece at the specified row and column is blocked by another piece on the board, e.g., calling `can_slide(2,2)` in the 5th image above will return false.

6.1 Algorithm Analysis [5 pts]

Assume that the game board has width w, the height of the tallest column is h, and the number of blocks (total number of piece characters and space characters) is b. What is the Big O Notation for the running time of your `can_slide` and `slide` functions that you have implemented on the next two pages? Write two to three concise and well-written sentences justifying your answers.

```
can_slide:


slide:
```

6.2 `can_slide` Implementation [12 pts]

```
bool Tetris::can_slide(int row, int column) const {

   // First, let's do some error checking on input arguments
   // and the current board state.  This will help when we need
   // to debug this new function.  Write if/else statements
   // and/or assertions to verify your assumptions.
```

sample solution: 8 line(s) of code

```
    // Now, we can do the logic necessary to determine whether this piece
    // can slide to the right.
```

sample solution: 6 line(s) of code

```
}
```

6.3 `slide` Implementation [18 pts]

```
void Tetris::slide(int row, int column) {
    assert (can_slide(row,column) == true);
```

sample solution: 26 line(s) of code

```
}
```

7 Lightning Round [13 pts]

```
std::vector<std::string> a;
std::list<std::string> b;
// omitted: initialize both containers to hold n = a large number of  words

01  a.push_front("apple");
02  b.push_front("banana");
03  a.push_back("carrot");
04  b.push_back("date");
05  std::vector<std::string>::iterator itr_a = a.begin();
06  std::list<std::string>::iterator itr_b = b.begin();
07  itr_a = a.insert(itr_a,"eggplant");
08  itr_a += 5;
09  itr_a = a.erase(itr_a);
10  itr_b += 5;
11  itr_b = b.insert(itr_b,"eggplant");
12  ++itr_b;
13  itr_b = b.erase(itr_b);
14  a.sort();
15  b.sort();
16  std::sort(a.begin(),a.end());
17  std::sort(b.begin(),b.end());
```

Which lines result in a compilation error?

Which lines cause a segmentation fault?

Which lines have a memory leak?

Which lines run in $O(1)$ time?

Which lines run in $O(n)$ time?

Which lines run in $O(n \log n)$ time?

Which lines run in $O(n^2)$ time?

8 Button Up the Linked Grid [26 pts]

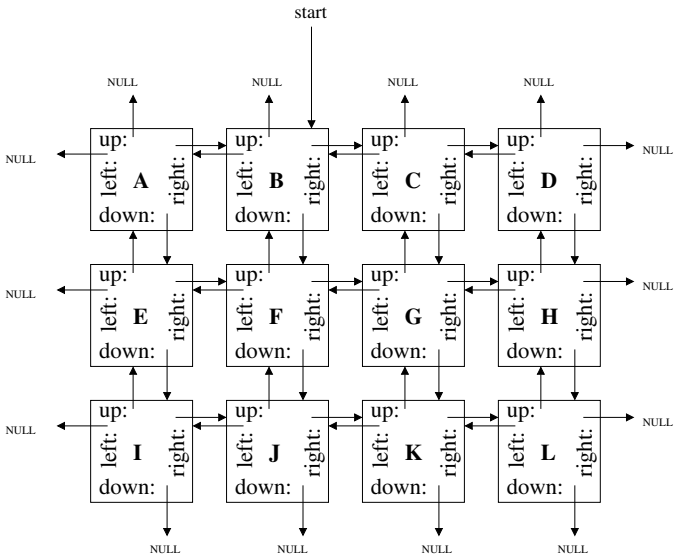
Alyssa P. Hacker and Ben Bitdiddle are working on a team project based on the linked grid of `Node`s data structure from Homework 5. Alyssa suggests they start with the `print_perimeter` function, which takes in a pointer to a `Node` named `start`, and walks around the edge of the grid in a *clockwise direction*. The function should print the value stored in every `Node` visited.

For example, `print_perimeter(start)` for the diagram shown on the right will print this sequence of values to the screen:

```
B C D H L K J I E A
```

Alyssa says it's ok to assume that the grid is at least two rows tall and at least two columns wide and that `start` definitely points to a `Node` somewhere on the edge/perimeter of the grid.

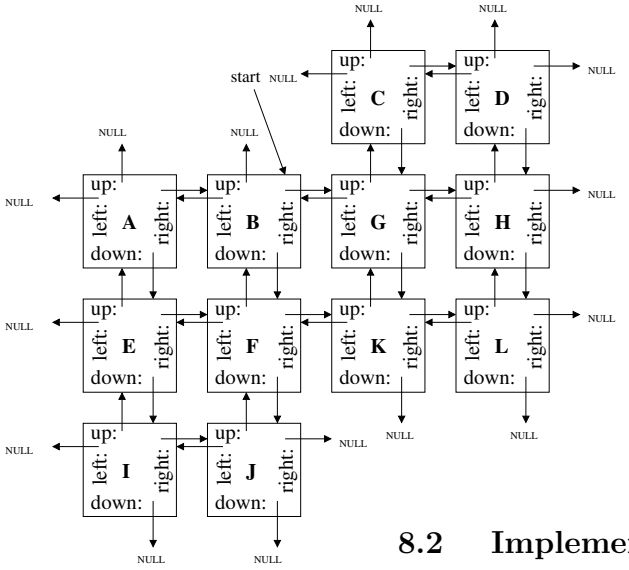
```
template <class T> class Node {
public:
  T value;
  Node<T> *up,*down,*left,*right;
};
```

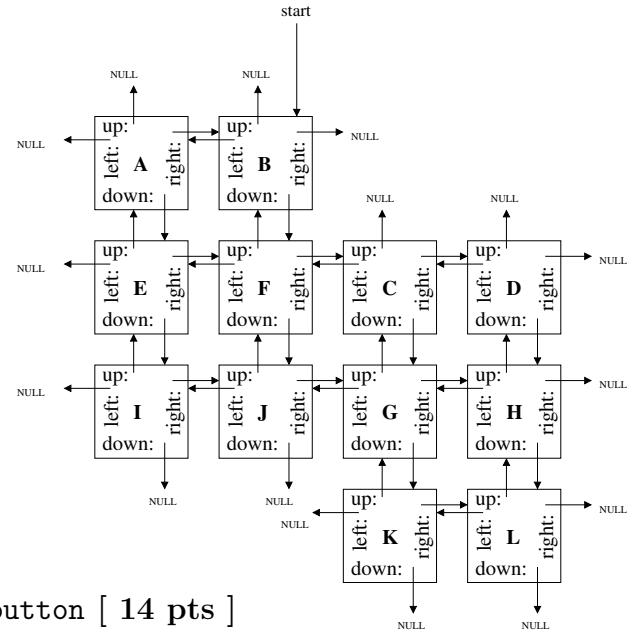


8.1 Implement `print_perimeter` [12 pts]

sample solution: 16 line(s) of code

Meanwhile, Ben is working on a function named `rebutton`, which takes in 2 arguments: `start`, a pointer to a `Node` on the top edge of the grid and a bool `shift_up`. The function makes a vertical cut to the right of `start` and reconnects the `Node`s on either side of the cut shifted up (below left) or shifted down (below right) one row. Ben claims that calling `rebutton(start,true)` followed by `rebutton(start,false)` will restore the original grid. And vice versa.

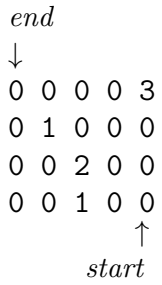




8.2 Implement `rebutton` [14 pts]

sample solution: 28 line(s) of code

9 Recursive Maximum Coin Path [23 pts]



Write a *recursive* function named `max_coin_path` that searches a 2D `grid` of "coins", an STL `vector` of STL `vector` of non-negative integers, for a path back to the origin (0,0). In walking from the start location (lower right corner of grid) to the origin (upper left corner), the path is only allowed to move up or left one grid space at a time. The goal is to find a path that *maximizes* the sum of the coins along the path. The function should return the maximum sum.

```
class Location {
public:
  Location(int r, int c)
    : row(r),col(c) {}
  int row;
  int col;
};
```

For the example shown above right, the path
`(3,4) (3,3) (3,2) (2,2) (2,1) (1,1) (1,0) (0,0)`
collects coins with values $1 + 2 + 1 = 4$, which is the maximum coin sum that can be achieved on this grid. The path achieving that sum should be stored in the second argument passed to the function, an STL `list` of `Location`s named `path`. Note: there are a few similar paths that have the same sum. Your function may return any of these optimal paths.

9.1 Usage [2 pts]

You will implement the `max_coin_path` on the next page. But first, complete the initial call to the `max_coin_path` function below. Assume `grid` has already been initialized; for example, with the data shown above. What additional information does your function need to get started?

```
std::list<Location> path;
int max_coin_sum = max_coin_path(grid,path,                    );
```

9.2 Algorithm Analysis [5 pts]

Assume that the grid width and height are w and h respectively, the number of non-zero coins in the grid is c, and the value of the maximum coin is m. What is the Big O Notation for the running time of your answer on the next page? Write three to four concise and well-written sentences justifying your answer.

9.3 Implementation [16 pts]

Now implement the `max_coin_path` function. Remember: it should be *recursive*.

sample solution: 27 line(s) of code

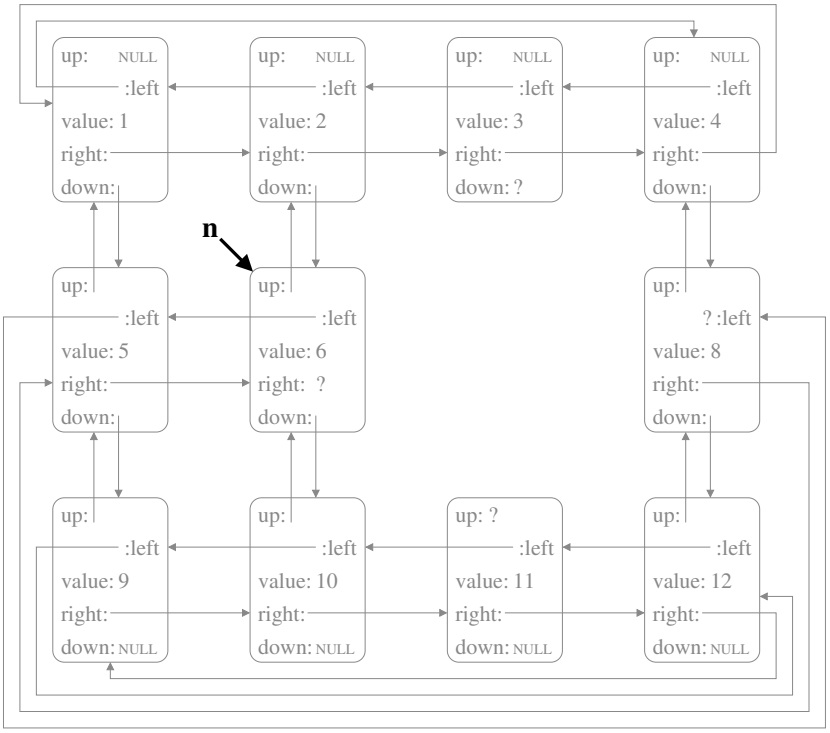
10 Linked Tube Repair [/ 33]

Alyssa P. Hacker is working on a modified linked list that is both two-dimensional and circular. A small sample with *height*=3 and *circumference*=4 is shown below. Each templated `Node` has pointers to its 4 neighbors. The top and bottom edges of the tube structure have `NULL` pointers. But the left and right edges wrap around, like a circularly linked list. This cylindrical *tube* structure may have any number of nodes for its height and its circumference.

```
template <class T>
class Node {
public:
  // REPRESENTATION
  T value;
  Node<T> *up;
  Node<T> *down;
  Node<T> *left;
  Node<T> *right;
};
```

10.1 Tube `repair` Diagram [/ 4]

First Alyssa wants to tackle the challenge of repairing a hole in the structure. Assume a single `Node` is missing from the structure, and we have a pointer `n` to the `Node` immediately to the left of the hole. Modify the diagram below to show all of the necessary edits for a call to `repair(n,7);`



10.2 Thinking about Tube `repair` Complexity [/ 3]

The `repair` function should have constant running time in most cases. Describe an example structure with a single missing `Node` that can be repaired, but *not* in constant time. Write 2-3 concise and well-written sentences. *You may want to complete the implementation on the next page before answering.*

10.3 Tube `repair` Implementation [/ 13]

Now, implement `repair`, which takes 2 arguments: a pointer to the `Node` immediately to the left of the hole and the value to be stored in the hole. You may assume a single `Node` is missing from the structure.

sample solution: 26 line(s) of code

10.4 Non-Iterative `destroy_tube` Implementation [/ 13]

Now write `destroy_tube` (and any necessary helper functions) to clean up the heap memory associated with this structure. The function should take a single argument, a pointer to any `Node` in the structure. You may assume the structure has no holes or other errors. You cannot use a `for` or `while` loop.

sample solution: 17 line(s) of code

11 Rehashing the `Vec` Assignment Operator [/ 15]

Complete the `Vec` assignment operator implementation below, while minimizing wasted heap memory. Assume the allocator is most efficient when all heap allocations are powers of two (1, 2, 4, 8, 16, etc.)

```
1   template <class T>
2   Vec<T>& Vec<T>::operator=(const Vec<T>& v) {
3     if (this != &v) {
4       delete                                                    ;
5       m_size =                                                  ;
6       m_alloc =                                                 ;
7       m_data =                                                  ;
8       for (int i = 0; i <                                       ; ++i) {
9         m_data[i] =                                             ;
10      }
11    }
12    return *this;
13  }
```

Add code below to perform a simple test of the assignment operator:

```
Vec<double> v;  v.push_back(3.14159);  v.push_back(6.02);  v.push_back(2.71828);
```

Is line 12 necessary? Continue your testing code above with a test that would break if line 12 was omitted.

What is the purpose of line 3? Write code for a test that would break if lines 3 and 10 were omitted.

12 Essay Revision: Embellishment [/ 14]

Write a function `embellish` that modifies its single argument, `sentence` (an STL `list` of STL `string`s), adding the word "`very`" in front of "`pretty`" and adding "`with a wet nose`" after "`grey puppy`". For example:

```
the pretty kitty sat next to a grey puppy in a pretty garden
```

Should become:

```
the very pretty kitty sat next to a grey puppy with a wet nose in a very pretty garden
```

sample solution: 20 line(s) of code

If there are w words in the input sentence, what is the worst case Big O Notation for this function? If we switched each STL `list` to STL `vector` in the above function, what is the Big O Notation?

STL `list`: STL `vector`:

13 Essay Revision: Redundant Phrases [/ 15]

Complete `redundant`, which takes a sentence and 2 phrases and replaces all occurrences of the first phrase with the second, shorter phrase. For example "`pouring down rain`" is replaced with "`pouring rain`":

```
it is pouring down rain so take an umbrella → it is pouring rain so take an umbrella
```

Or we can just eliminate the word "`that`" (the replacement phrase is empty):

```
I knew that there would be late nights when I decided that CS was the career for me
→ I knew there would be late nights when I decided CS was the career for me
```

```
typedef std::list<std::string> words;

void redundant(                  sentence,                  phrase,                  replace) {
```

sample solution: 19 line(s) of code

```
}
```

14 Don't Ignore Compilation Warnings! [/ 15]

Write a useful but buggy segment of code (or function) that will compile with no errors but will produce the indicated compilation warning. Put a star ★ next to the line of code that will trigger the warning. Write a concise and well-written sentence describing the intended vs. actual (buggy) behavior of the code.

`warning: comparison of integers of different signs: 'int' and 'unsigned int'`

`warning: control reaches / may reach end of non-void function`

`warning: variable is uninitialized when used here / in this function`

`warning: returning reference to local temporary object / reference to stack memory associated with a local variable returned`

`warning: expression result unused / expression has no effect`

`warning: unused variable / unused parameter`

15 Cyber Insecurity [/ 5]

Ben Bitdiddle wrote the following code fragment to manage his personal information.

```
1   std::ifstream istr("my_information.txt");
2   std::string s;
3   std::vector<std::string> data;
4   while (istr >> s) { data.push_back(s); }
5   std::vector<std::string>::iterator password = data.begin()+4;
6   data.push_back("credit_card:");
7   data.push_back("1234-5678-8765-4321");
8   data[4] = "qwerty";
9   std::cout << "my password is: " << *password << std::endl;
```

my_information.txt

```
name: Ben Bitdiddle
password: pa$$word
SSN: 123-45-6789
```

Write "True" in the box next to each *true* statement. Leave the boxes next to the *false* statements empty.

	Lines 2 & 3 will produce an "uninitialized read" error when run under `gdb` or `lldb`.
	Line 5 is not a valid way to initialize an iterator.
	Ben's credit card information is not saved back to the file.
	This program might behave differently if re-run on this computer or another computer.
	A memory debugger might detect an "unaddressable access of freed memory" error on Line 9.
	If we move lines 6 & 7 after line 9, this code fragment will run without memory errors.
	This code contains memory leaks that can be detected by Dr. Memory or Valgrind.
	These password choices disqualify Ben from any job in computer security.