The University of Michigan
Electrical Engineering & Computer Science
EECS 281: Data Structures and Algorithms
Fall 2024

# Lab 3: Complexity Analysis, Amortization, and Strings

**Instructions:**
Work on this document with your group, then enter the answers on the canvas quiz.

---

**Note:**
Be prepared before you meet with your lab group, and read this document so that you know what you must submit for full credit. You can even start it ahead of time and then ask questions during any lab section for help completing it.

You MUST include the following assignment identifier at the top of every file you submit to the autograder as a comment. This includes all source files, header files, and your Makefile (if there is one). If there is no autograder assignment, you may ignore this.

**Project Identifier:** 5AE7C079A8BF493DDDB6EF76D42136D183D8D7A8

The Autograder has changed from previous semesters. If you are taking this class for the second time, you may need to alter the code that you have from the last time you took the class.

---

## 1 Logistics

1. What is the due date of lab 2 quiz and AG?
   - A. 9/21/2024
   - B. 9/23/2024
   - C. 9/19/2024
   - D. 9/25/2024
2. What is the due date of lab 3 handwritten?
   - A. 9/24/2024
   - B. 9/27/2024
   - C. 9/23/2024
   - D. 9/26/2024
3. What is the due date of lab 3 quiz and AG?
   - A. 9/28/2024
   - B. 9/26/2024
   - C. 10/04/2024
   - D. 9/30/2024
4. What is the due date of Project 2?
   - A. 10/6/2024
   - B. 10/10/2024
   - C. 10/12/2024
   - D. 10/14/2024

## 2 Recurrence Relations and Complexity Analysis

5. What is the best approach for deducing the following relation, given $T(1) = 1$?

$$T(n) = 8T(\frac{n}{2}) + 3n$$

   A. The Substitution Method
   B. The Master Theorem

6. What is the best approach for deducing the following relation, given $T(1) = 1$?

$$T(n) = 2T(n-1) + 2$$

   A. The Substitution Method
   B. The Master Theorem

7. Given the function below, calculate the recurrence relation. Assume bar($n$) runs in $\Theta(n)$ time.

```
void foo(int n){
    if (n == 1)
        return;

    foo(n / 4);

    int k = n * n;

    for (int i = 0; i < k; ++i)
        for (int j = 0; j < n; ++j)
            bar(n);

    for (int i = 0; i < n; ++i)
        foo(n / 2);

    bar(k);
} // foo()
```

   A. $T(n) = T(\frac{n}{4}) + n^3 \log n + T(\frac{n}{2}) + n^2$
   B. $T(n) = T(\frac{n}{4}) + n^3 \log n + nT(\frac{n}{2}) + n^2$
   C. $T(n) = T(\frac{n}{4}) + n^4 + nT(\frac{n}{2}) + n^2$
   D. $T(n) = T(\frac{n}{4}) + n^4 \log n + nT(\frac{n}{2}) + n^2$

8. Which ordering lists $\Theta(n^{100}), \Theta(100^n), \Theta(n!), \Theta(\log(n!))$ in order of increasing complexity? *Hint*: $\Theta(\log(n!))$ can be simplified to a complexity class that you've likely seen before. Think about any mathematical identities that may help you perform this simplification.
   - A. $\Theta(n^{100}), \Theta(100^n), \Theta(\log(n!)), \Theta(n!)$
   - B. $\Theta(100^n), \Theta(n^{100}), \Theta(\log(n!)), \Theta(n!)$
   - C. $\Theta(\log(n!)), \Theta(n^{100}), \Theta(n!), \Theta(100^n)$
   - D. $\Theta(\log(n!)), \Theta(n^{100}), \Theta(100^n), \Theta(n!)$
   - E. $\Theta(\log(n!)), \Theta(100^n), \Theta(n^{100}), \Theta(n!)$

9. Solve the following recurrence relation:

$$T(n) = \begin{cases} 281, & \text{if } n = 1 \\ T(n-1) + 370, & \text{if } n > 1 \end{cases}$$

   - A. 281n + 370
   - B. 281n + 651
   - C. 370n + 281
   - D. 370n - 89
   - E. 370n - 459

10. Consider the following recurrence relation:

$$T(n) = \begin{cases} c_0, & \text{if } n = 1 \\ \sqrt{2}T(\frac{n}{2}) + c_1, & \text{if } n > 1 \end{cases}$$

What is the tightest complexity class that you can attribute to this recurrence relation?
   - A. $T(n) = \Theta(\sqrt{n})$
   - B. $T(n) = O(\sqrt{n})$
   - C. $T(n) = \Omega(\log_2 n)$
   - D. $T(n) = \Theta(n \log_2 n)$
   - E. $T(n) = \Theta(n^2)$

11. What is the worst-case complexity of this function?

```
void foo(vector &v) {
    int n = static_cast(v.size());
    int rt = static_cast(floor(sqrt(n)));

    for (int i = 0; i < rt; ++i) {
        for (int j = 0; j < rt * rt; j += rt) {
            cout << v[j + i] << " ";
        }
        cout << endl;
    }
}
```

A. $\Theta(\sqrt{n})$
B. $\Theta(n)$
C. $\Theta(n\sqrt{n})$
D. $n \log n$
E. $n^2$

12. What is the worst-case time complexity of this function?

```
void potato(int n) {
    for (int i = 1; i < n; i *= 2) {
        for (int j = 1; j < n; ++j) {
            for (int k = 1; k < j; ++k) {
                cout << "I'm a smart potato! o(^-^)o \n";
}
```

A. $\Theta(n^2)$
B. $\Theta(n^2 \log n)$
C. $\Theta(n \log^2 n)$
D. $\Theta(n^2 \log^2 n)$
E. $\Theta(n^3)$

## 3 Vectors and Strings

13. What is the worst-case time complexity of the `push_back` operation discussed in the lab slides, given a vector of size $n$?

    A. $\Theta(1)$

    B. $\Theta(\log n)$

    C. $\Theta(n)$

    D. $\Theta(n^2)$

14. What is the amortized time complexity of the `push_back` operation discussed in the lab slides, given a vector of size $n$?

    A. $\Theta(1)$

    B. $\Theta(\log n)$

    C. $\Theta(n)$

    D. $\Theta(n^2)$

15. Consider the following snippet of code:

    ```
    char str1[] = "BING";
    cout << strlen(str1);
    cout << sizeof(str1);

    string str2("BING");
    cout << str2.length();
    cout << str2.size();
    ```

    What is the output?

    A. 4444

    B. 4455

    C. 4544

    D. 5544

    E. None of the above is correct.

16. Consider the following snippet of code:

    ```
    const char *s = "hello";
    char ss[20];
    int length = strlen(s);
    for (int i = 0; i < length; ++i)
        ss[i] = s[length - i];
    printf("%s", ss);
    ```

    What is the output?

    A. hello

    B. olleh

    C. olle

    D. segmentation fault

    E. no output is printed

## 4 Handwritten Problem

This problem is to be submitted independently. We recommend trying it on your own, checking your answer with a group and discussing solutions, and then submitting it in lab. These will be graded on completion, not by correctness. However, we want to see that you were thinking about the problem. Please implement your solution on the lab assignment template, and you may test the solution using `anagram.h`. The starter files can be found on Canvas.

17. Write a function that takes in two strings and returns whether they are anagrams of each other (words that contain the same letters). The only characters will be spaces and lowercase letters. Do this in $\Theta(n)$ time.

    - Example 1: Given s1 = “anagram” and s2 = “nagaram” , return true.
    - Example 2: Given s1 = “i love eecs” and s2 = “i scole ve e” , return true.
    - Example 3: Given s1 = “anagrams” and s2 = “anagrams anagrams” , return false.
    - Example 4: Given s1 = “cats” and s2 = “cat” , return false.

```
// check if two strings are anagrams
bool isAnagram(const string &s1, const string &s2);
```

## 5 Coding Assignment

You can work on these problems by yourself or with your group, but a solution must be submitted to the autograder for each individual.

18. **String Library Implementation**

The following specs contains every detail about the problem - if you want a quick TL;DR summary, see `appendix A`.

In lab and lecture, you learned about C++ strings and the C++ string library. The string class provides many helpful member functions that simplify the usage of string objects (such as find, insert, erase, just to name a few!). While you do not have to fully understand how a string object works under the hood, knowing these string operations will be helpful for future projects and interviews. In this lab assignment, you will implement five of these operations within a `String` class that we have provided for you. You will be given three submissions to the autograder per day (but we will give you the autograder's test cases; this will be covered later).

Download the starter files from either Canvas or Github (https://github.com/eecs281staff/l3-string-library). Please complete the assignment in the `String.cpp` file. **Do not include any libraries beyond the ones given to you or modify anything outside the parts labeled with TODO**. Doing so may cause your program to fail on the AG. To clone the directory on to your local machine, go to your terminal and type the following command:
`git clone https://github.com/eecs281staff/l03-string-library.git`

You will be implementing a portion of this String class so that its behavior emulates that of a standard C++ string. Before you begin, there are a few things you should know. Please read the following items carefully:

- On lines 162 and 163 of the original `String.h` file, you will see two member variables: `cstr` and `sz`. `cstr` is a c-string member variable that stores the underlying contents of the String object. Recall that c-strings are arrays of characters that are terminated by a null-character, or sentinel ('\0'). The variable `sz` stores the size of the String object, not including the null-terminating character. **Make sure that both variables are updated correctly after each operation.**
- The null-terminating character ('\0') is defined as the member variable `a_null_byte` (on line 29 of the `StringVerifier.cpp` file). You may compare a character with `a_null_byte` or assign a character to `a_null_byte`. Remember that this character must be found at the end of the c-string representing the contents of the String object, as it denotes the end of the c-string.
- Strings have a constant value called `npos` that represents the largest possible value for an element of type `size_t` (or 18,446,744,073,709,551,615). This is defined on line 28 of the `StringVerifier.cpp` file. This constant is often returned by find operations if the target is not found in the string.
- You won't have to worry about illegal operations (e.g. indexing out of bounds, etc.) or growing the c-string array if an operation causes its contents to exceed its capacity, although it is important to note that such situations can exist and must be accounted for.
- For `String` operations that require returning a reference to a `String`, you should return `*this` in the function implementation. this is a pointer to the current `String`, so `*this` dereferences the pointer and returns a reference to the current String.

You may notice that the starter file we gave you supports many string operations. **For this lab assignment, you will only be responsible for five of these functions.**

- `erase`
- `insert`
- `replace`
- `find_first_of`
- `find_last_of`

Other than the information presented above, you may skip over everything that doesn't have TODO attached to it. However, feel free to look over these other functions if you are curious! It is recommended that you review the "Arrays and Containers" lecture before starting this assignment. Go to the first TODO in the starter file (search the document for "TODO #1," which should be located on line 186 of the `String.cpp` file if the original file was not modified).

**TODO #1: Erase Function**

The `erase()` function is defined as follows:

```
String& String::erase(size_t pos, size_t len = npos);
```

This function can be used to remove a range of characters within a `String`. When `erase()` is called using two parameters, `pos` and `len`, this function erases the portion of the `String` that begins at index `pos` and spans `len` characters, or until the end of the `String`, whichever comes first. If `len` is not defined, it defaults to a value of `npos`, and the `String` is erased from `pos` to the end. A reference to the modified `String` is then returned. For example:

```
String str = "darden";
str.erase(3, 2);
```

erasing would begin at index 3 of `str`, and 2 characters would get erased. In this case, the second 'd' (the character at index 3) and the following 'e' would be erased, and the contents of `str` after the call to erase would be "darn". You may assume that the starting position `pos` will always be valid, and `len` will be greater than 0. A valid `pos` can take on any value in the inclusive range 0 ≤ `pos` ≤ `sz`.

Make sure that the final c-string you end up with has a sentinel at the end. This can be done by assigning a character in the c-string with `a_null_byte`, a member variable that has a value of '\0':

```
cstr[last_index] = a_null_byte;
```

where `last_index` is the index one past the last character of the modified `String`.

One thing to note is that the underlying c-string that is used to store the contents of the String is an array of characters, which stores elements **contiguously** in memory. Thus, once you erase a portion of the array, all the elements after the point of erasure must be shifted over so that all the characters can **remain contiguous in memory**.

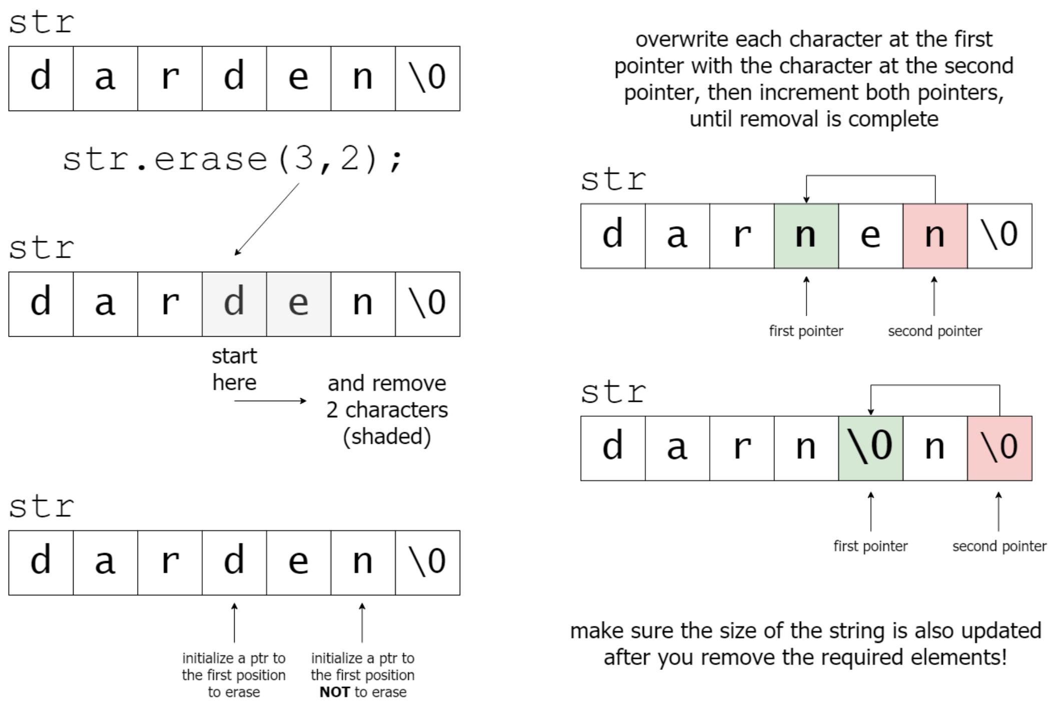
To accomplish this, you may use the following algorithm:

1. First, declare an index that references the index of the first character to be erased. This can be done by accessing the element at position `pos` of the c-string `cstr`.
2. Next, declare an index that references the index of the first character NOT to be erased. This can be done by accessing the element at position `pos` + `len` of the c-string `cstr`.

3. After declaring both indices, overwrite each character at the first index with the character at the second index, then increment both pointers, until removal is complete. Removal is complete when the second index reaches the end of the string. Make sure the last index of the final modified string is set to the sentinel character, `a_null_byte` (or '\0').
4. Adjust the value of `sz` so that it reflects the new size of the modified `String`.

This process is shown visually on the next page. Note that the following figure uses the word "pointer" instead of "index" to represent positions in the c-string array. **However, this does not mean that you must use pointers to implement the erase operation.** It is more than possible to use indices to keep track of the two positions needed to perform the erase. In other words, instead of keeping track of a pointer to the address of `cstr[2]`, for example, you could just keep track of the element's integer index, or 2.

The `erase` function, visualized: In the above diagram, after `erase` is complete, the contents


str
d a r d e n \0
str.erase(3,2);
str
d a r d e n \0
start here
and remove 2 characters (shaded)
str
d a r d e n \0
initialize a ptr to the first position to erase
initialize a ptr to the first position NOT to erase
overwrite each character at the first pointer with the character at the second pointer, then increment both pointers, until removal is complete
str
d a r n e n \0
first pointer
second pointer
str
d a r n \0 n \0
first pointer
second pointer
make sure the size of the string is also updated after you remove the required elements!

of the c-string are 'd','a','r','n','\0','n','\0'. Are we done here, or do we have to remove the last 'n' and sentinel character at the end? Think about what the purpose of the sentinel character is in a c-string. Do the data elements after this character matter? No, it does not! This is because the sentinel tells the program that it has reached the end of the string, so all characters after this sentinel character are ignored.

An alternative solution for those who want a challenge: `erase` can also be implemented using the copy-swap method. This would require initiating a `String` object with the updated contents (after the necessary characters are removed) and swapping it with the current `String` (using the `String` swap function we defined for you on line 80). To use the copy-swap method to erase characters from a string, you will need to use the `String substr` function, which is implemented for you on line 165. Note that this copy-swap implementation is **completely optional**, and the algorithm listed on the previous page is sufficient for getting full credit on the `erase` function.

## TODO #2: Insert Function

Next, we will look at the `insert()` function, which can be used to insert characters into a `String`. The `insert()` function is defined as follows:

```
String& String::insert(size_t pos, const String& str);
```

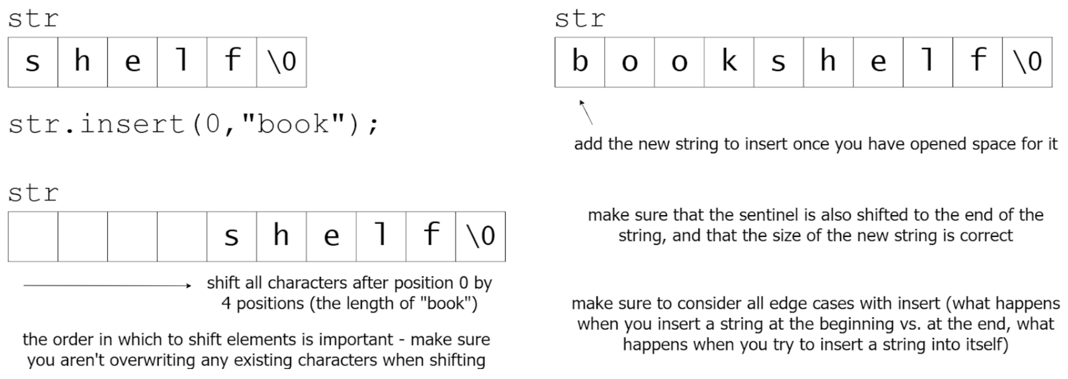
When `insert()` is called using two parameters, `pos` and `str`, this function inserts the contents of `str` **BEFORE** the character at position `pos`. A reference to the modified `String` is returned. For example:

```
String str = "shelf";
str.insert(0, "book");
```

the string "book" would be added before position 0 of `str`, resulting in a final `str` value of "bookshelf". You may assume that the starting position `pos` will always be valid.

Since insertion may cause the underlying `c-string` to exceed its capacity, a check needs to be done to ensure that there is enough memory allocated to hold the new `String` once everything has been inserted. This check has been provided for you; you do not need to worry about it, but do know that it exists.

How would you begin implementing this `insert` operation? Recall how an array works when you try to insert an element - all other elements after the insertion point must be shifted. This is true here as well. First, you must shift all characters after the insertion point by a certain distance. Then, you would insert the contents of the new `String` into the slots you have freed up for insertion. Use the figure below as a reference.


str
s h e l f \0
str.insert(0,"book");
str
s h e l f \0
shift all characters after position 0 by 4 positions (the length of "book")
the order in which to shift elements is important - make sure you aren't overwriting any existing characters when shifting
str
b o o k s h e l f \0
add the new string to insert once you have opened space for it
make sure that the sentinel is also shifted to the end of the string, and that the size of the new string is correct
make sure to consider all edge cases with insert (what happens when you insert a string at the beginning vs. at the end, what happens when you try to insert a string into itself)

Once again, make sure that your new `c-string` includes the sentinel at the very end, and that the size is properly updated after the `insert` is complete.

For those who want to explore copy-swap (optional): `insert` can also be implemented using the copy-swap method. This would require initiating a `String` object with the updated contents (after the necessary elements are added) and swapping it with the current `String` (using the `String` swap function we defined for you on line 80). Like `erase`, you would need to use the `substr` function, which has been implemented for you. As an additional hint, the new `String` has three "sections" to it, the portion before the point of insertion, the portion after the point of insertion, and the new `String` that is added, so you will need to combine all three sections before you initiate the copy-swap.

One note that should be made is that you may come across a few test cases where a string is inserted into itself. In these cases, it would not be safe to begin modification of `cstr` immediately, since the insertion may depend on the original value of the `String`. There are several ways to bypass this. One way is to check if the address of the `String` to insert (`&str`) is equal to the address of the current `String (this)`, and to insert differently if they are. Another method is to simply implement `insert` in a way that guarantees that `str` always retains its original value throughout the life of the function (e.g. making a copy of `str` and using the copy to do the `insert`).

### TODO #3: Replace Function

Next, you will implement the `replace()` function. The `replace()` function is defined as follows:

```
String& String::replace(size_t pos, size_t len, const String& str);
```

When `replace()` is called using three parameters, `pos`, `len`, and `str`, this function replaces the portion of the `String` that begins at character `pos` and spans `len` characters with the contents of `str`. For example:

```
String str = "EECS 281 is hard";
str.replace(12, 4, "fun");
```

the substring of length 4 starting at position 12 of `str` ("`hard`") would be replaced with the string "`fun`". The final contents of `str` after the call to replace would be "`EECS 281 is fun`". You may assume that `pos` is valid. If the value of `len` exceeds the end of the `String`, replace as many characters as possible.

The implementation of `replace` should not take more than 10 lines of code! This is because you have already implemented `erase` and `insert`; a call to `replace` is simply a combination of these two operations.

If you aren't sure that your `erase` or `insert` are fully functional, an alternative (but longer) approach would be to implement the function from scratch. Shift all the characters after the segment to replace, and move the new `String` into the space you opened up. If `str` is shorter than `len`, the replaced section will be shorter than it was before, and if `str` is longer than `len`, the replaced section will be longer than it was before.

Like with `insert`, you will have to deal with self-replacement. To solve this, use an approach similar to the one you used to overcome self-insertion; either check for the case where `&str == this` (the `String` being passed in is the same as the `String` you are modifying) or implement your function in a way that guarantees that the value of `str` stays valid throughout the entire `replace` call.

### TODO #4: Find-First-Of Function

Now, we will discuss functions that can be used to find specific characters in a `String` object. The `find_first_of()` function is defined as follows:

```
size_t String::find_first_of(const String& str, size_t pos = 0);
```

This function checks if ANY characters of `str` can be found in the `String`. In other words, given two parameters `str` and `pos`, this function searches the `String` for the first character that matches ANY of the characters in `str`, starting from position `pos` of the `String`. Characters before `pos` are ignored. It is enough for a single character of `str` to match for the search to be successful. If a match is found, the function returns a `size_t` that represents the position of the first character that matches. Otherwise, the function returns `npos`. If `pos` exceeds the length of the string, the function will never find a match. If `pos` is not specified, the value of `pos` is assumed to be 0.

As long as a single character in `str` can be found in the `String`, the search is successful and returns the position of the match, as demonstrated in the examples below:

```
String str = "EECS 281 is fun";
size_t found = 0;
```

```
found = str.find_first_of("281", 0);
// found is 5 since the char '2' can be found at index 5

found = str.find_first_of("280", 0);
// found is 5 since the char '2' can be found at index 5

found = str.find_first_of("281", 6);
// found is 6 since the char '8' can be found at index 6
```

The implementation of `find_first_of()` only checks for the existence of one character match rather than an entire `String` match.

## TODO #5: Find-Last-Of Function

The `find_last_of()` function is defined as follows:

```
size_t String::find_last_of(const String& str, size_t pos = npos);
```

The `find_last_of()` function is very similar to the `find_first_of()` function. However, this function looks for the last character in a `String` that matches any of the characters in `str`, rather than the first. The search begins at position `pos` of the `String` and moves toward position 0, searching for a match along the way. Characters after position `pos` are ignored. If a match is found, the function returns a `size_t` that represents the position of the last character that matches. Otherwise, the function returns `npos`. If `pos` exceeds the length of the `String`, the entire `String` is searched. For example:

```
String str = "EECS 281 is fun";
size_t found = 0;
found = str.find_last_of("Eggs", 14);
// found is 10 since an 's' can be found at index 10
```

Unlike `find_first_of()`, where `pos` defaults to zero, it is possible for `find_last_of()` to take in a value of `pos` that exceeds the size of the `String`. Make sure you take this into consideration.

One thing you have to watch out for when implementing `find_last_of()` is the concept of overflow. Overflow occurs when the value of a data type exceeds the maximum or goes below the minimum value that a data type can hold. When this happens, the value "wraps around" - that is, if you increment a data type past its maximum value, the new value wraps around to the minimum value, and vice versa. Consider the following code:

```
int main() {
   int counter = 10, val = 2147483643;
   while (--counter >= 0) {
     cout << val << "\n";
     ++val;
   }
}
```

The largest possible value of an `int` is 2,147,483,647, so incrementing 2,147,483,647 does not give you 2,147,483,648. Instead, `val` would wrap around to the minimum possible value an `int` can hold (-2,147,483,648), so incrementing 2,147,483,647 by one instead gives you -2,147,483,648.

The above code prints the following. Notice that overflow occurs when 2,147,483,647 is incremented.

```
2147483643
2147483644
2147483645
2147483646
```

```
2147483647
-2147483648
-2147483647
-2147483646
-2147483645
-2147483644
```

The same thing applies to values of type `size_t`. A `size_t` is unsigned, so the minimum value a `size_t` can take on is 0. If you attempt to decrement a `size_t` below 0, the value of `size_t` ends up wrapping around to `npos`, which is the largest value a `size_t` can take on! As a result, this for loop will never exit:

```
for (size_t i = std::min(pos, sz - 1); i >= 0; --i) { /* do stuff */ }
```

This is because the expression `i >= 0` is **always true** for a `size_t`. You will need to find another way to break out of the loop when implementing the function.

### Testing Your Code

To test your code, you may use the provided `StringVerifier.cpp` test file, which includes all the tests that will be used to judge your implementation on the autograder. To begin testing, run `make` using the Makefile we provide you in the starter files. This will generate an executable file called `strlib`, which can be used to test your solution.

### Submitting to the Autograder

You may work with a partner on this lab. To submit to the autograder, create a `.tar.gz` file containing just `String.cpp`, as shown below. **Do not modify the `String.cpp` file name, or your submission won't run!** Make sure the capitalization of your command matches.

```
tar -czvf lab3.tar.gz String.cpp
```

If you are using the Makefile we provide you, you can type `make fullsubmit`, which will also run the command above.

If you are working with a partner, **both partners must submit to the autograder**. Only students who submit code to the autograder will receive points. It's perfectly fine for both partners to submit identical code, as long as the code was written by both of the partners. You will be able to make three submissions to the autograder per day, up until the due date. Make sure the assignment identifier is on all code files you submit. The autograder will be very lenient with time and memory; the score you see is the score you will get, even if you have some test cases that are blue.

The test cases in the provided starter files are separated on the autograder, so if you are failing only 1 of the insert tests, you can get points for the other ones you are passing. The first letter of the test case denotes the operation that is being tested, and the number represents the test number (the same as in your starter files). Thus, test case E05 is the 5th erase test in `StringVerifier.cpp`.

**Project Identifier:** 5AE7C079A8BF493DDDB6EF76D42136D183D8D7A8

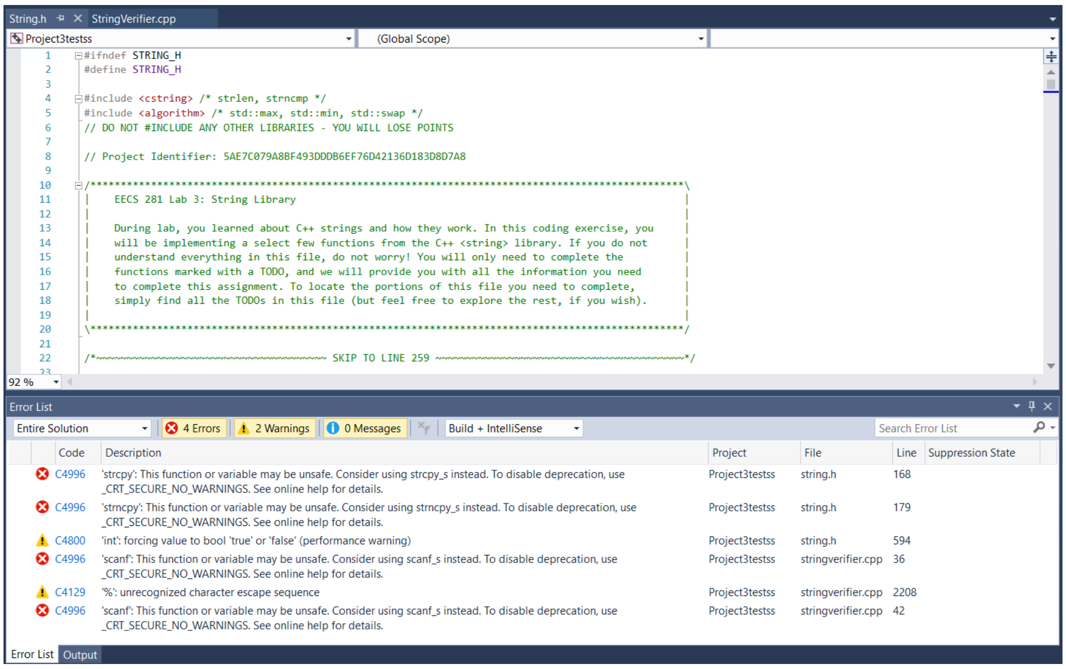
## Appendix A: Summary of Coding Exercise

Here is a quick summary of what you should do, for those who do not want to read the whole thing (however, the above instructions contain useful hints on how to implement each of the required functions):

- Retrieve the Lab 3 starter files from the GitHub or Canvas.
- Open the `String.cpp` file, which you will be implementing this lab assignment in. The `cstr` and `sz` member variables of the `String` class represent the underlying `c-string` that stores the `String`'s data and its size, respectively. The value `npos` represents the largest value a `size_t` can hold.
- Implement the `insert`, `erase`, `replace`, `find_first_of`, and `find_last_of` operations, testing with the provided test files along the way. Submit to the autograder frequently, and don't wait until the last day!

## Appendix B: Compiling on Visual Studio

If you are using Visual Studio, you might encounter the following compiler errors when trying to run your code:



```
#ifndef STRING_H
#define STRING_H

#include <cstring> /* strlen, strncmp */
#include <algorithm> /* std::max, std::min, std::swap */
// DO NOT #INCLUDE ANY OTHER LIBRARIES - YOU WILL LOSE POINTS

// Project Identifier: 5AE7C079A8BF493DDDB6EF76D42136D183D8D7A8

/*****************************************************************************************\
|    EECS 281 Lab 3: String Library                                                       |
|                                                                                         |
|    During lab, you learned about C++ strings and how they work. In this coding exercise, you |
|    will be implementing a select few functions from the C++ <string> library. If you do not |
|    understand everything in this file, do not worry! You will only need to complete the   |
|    functions marked with a TODO, and we will provide you with all the information you need |
|    to complete this assignment. To locate the portions of this file you need to complete, |
|    simply find all the TODOs in this file (but feel free to explore the rest, if you wish). |
|                                                                                         |
\*****************************************************************************************/

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ SKIP TO LINE 259 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
```

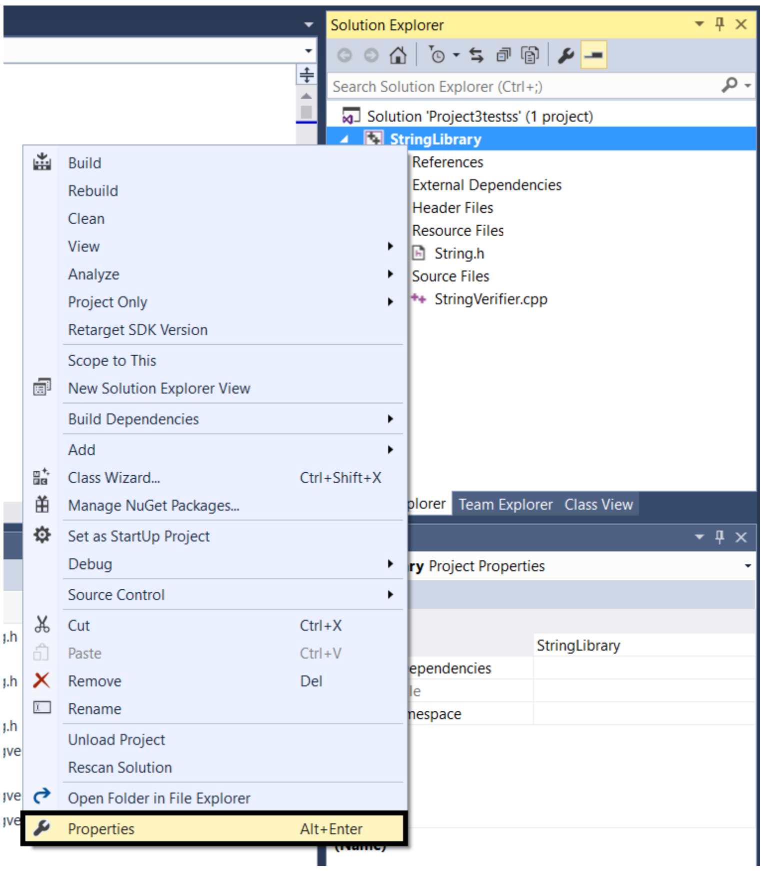
| | Code | Description | Project | File | Line | Suppression State |
|---|---|---|---|---|---|---|
| ✖ | C4996 | 'strcpy': This function or variable may be unsafe. Consider using strcpy_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details. | Project3testss | string.h | 168 | |
| ✖ | C4996 | 'strncpy': This function or variable may be unsafe. Consider using strncpy_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details. | Project3testss | string.h | 179 | |
| ⚠ | C4800 | 'int': forcing value to bool 'true' or 'false' (performance warning) | Project3testss | string.h | 594 | |
| ✖ | C4996 | 'scanf': This function or variable may be unsafe. Consider using scanf_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details. | Project3testss | stringverifier.cpp | 36 | |
| ⚠ | C4129 | '%': unrecognized character escape sequence | Project3testss | stringverifier.cpp | 2208 | |
| ✖ | C4996 | 'scanf': This function or variable may be unsafe. Consider using scanf_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details. | Project3testss | stringverifier.cpp | 42 | |

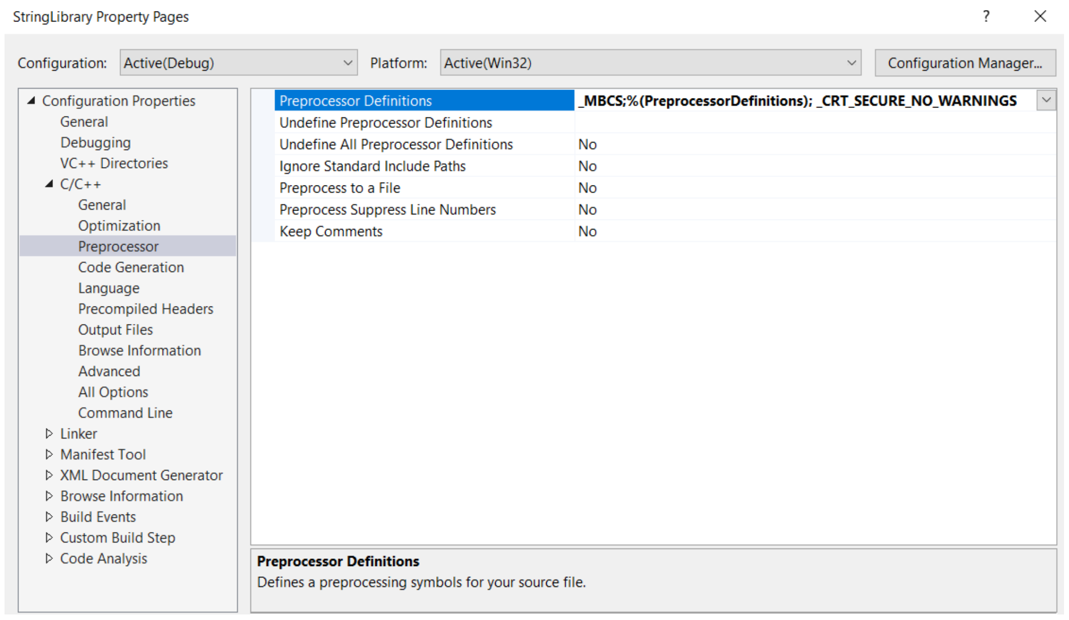
```
Error C4996 'strcpy': This function or variable may be unsafe.
Consider using strcpy_s instead. To disable deprecation, use
    _CRT_SECURE_NO_WARNINGS. See online help for details.

Error C4996 'strncpy': This function or variable may be unsafe.
Consider using strncpy_s instead. To disable deprecation, use
    _CRT_SECURE_NO_WARNINGS. See online help for details.
```

This is not an issue with the code we gave you, but rather your Visual Studio settings. To fix this, you must modify your preprocessor settings so that c-string functions can be used. Right click the project name in the solution explorer, click Properties, go to C/C++ > Preprocessor, and add a semicolon and "`_CRT_SECURE_NO_WARNINGS`" to the Preprocessor Definitions line (as shown below).

After adding this to the preprocessor definitions, your code should compile normally.





Configuration: Active(Debug) Platform: Active(Win32)

| | |
|---|---|
| Preprocessor Definitions | _MBCS;%(PreprocessorDefinitions); _CRT_SECURE_NO_WARNINGS |
| Undefine Preprocessor Definitions | |
| Undefine All Preprocessor Definitions | No |
| Ignore Standard Include Paths | No |
| Preprocess to a File | No |
| Preprocess Suppress Line Numbers | No |
| Keep Comments | No |

**Preprocessor Definitions**
Defines a preprocessing symbols for your source file.