Lab 2

Due Feb 6 by 12:30pm **Points** 100 **Submitting** a file upload **File Types** s
Available Jan 23 at 1:55pm - Feb 6 at 12:30pm 14 days

This assignment was locked Feb 6 at 12:30pm.

Purpose

In this lab, you will gain experience seeing how classes are actually implemented in assembly. You will also gain experience in how functions are "overloaded", since labels must be unique in assembly to avoid ambiguity.

Assignment

You will be implementing the functions of a simple C++ "string" class in assembly called "mystring" (declared below). You must use the exact same names as I have below, including the class. This is due to the "name mangling" that C++ uses. You do not have to figure out the mangled names yourself, please scroll down in this assignment. I have added a table to translate the C++ to Assembly member function names.

mystring Class

```
class mystring {
        char *mString;
        int mStringLength;
public:
        //Destructor. You must free whatever was allocated in mString and reset
        //the mString pointer and mStringLength to 0.
        ~mystring();
        //Default constructor sets mString to an empty string
        //and mStringLength to 0. Do NOT let mString be a NULL pointer, ever!
        mystring();
        //Constructor that takes a C-style string pointer. You must allocate
        //memory for mString and COPY src into it. Then determine the length
        //and store that into mStringLength.
        mystring(const char *src);
        //Constructor that takes a mystring. You must copy mString and mStringLength.
        //DO NOT just copy the pointer. You must allocate a new pointer and
        //copy the string.
        mystring(const mystring &rhs);

        //Assignment of a C-style string. This must function just like
        //mystring(const char *src) except that in this function, you must
        //free the old mString pointer. This returns a reference to the class.
        mystring &operator=(const char *src);
        //Assignment of an rvalue mystring. All you have to do is copy the pointer
        //and length. You must not reallocate the pointer since this is an rvalue
        //which is what the && denotes.
```

```
        mystring &operator=(const mystring &&rhs);

        //Simply return mStringLength
        int length() const;
        //Find the location of the needle. Return -1 if the needle
        //was not found, or the index if it was.
        int find(const mystring &needle) const;
        //Same as find above, except this is given the needle as a C-style string.
        int find(const char *needle) const;

        //Simply return mString pointer.
        const char *c_str() const;

        //This allows us to cout << mystring. This is a friend, so this is
        //static with no this pointer. The ostream class knows how to output
        //a c-style string, so that is what needs to be written to s. You
        //will need to return the reference of ostream &s.
        friend ostream &operator<<(ostream &s, const mystring &rhs);
};
```

Mangled Name Table

As you know, C++ will mangle the names of the functions to allow for operator and function overloading. The following table shows you the mangled name of each function in the mystring class:

C++ Name	Assembly Name
~mystring()	_ZN8mystringD1Ev
mystring()	_ZN8mystringC1Ev
mystring(const char *s)	_ZN8mystringC1EPKc
mystring(const mystring &rhs)	_ZN8mystringC1ERKS_
mystring &operator=(const char *s)	_ZN8mystringaSEPKc
mystring &operator=(const mystring &&rhs)	_ZN8mystringaSEOKS_
int length() const	_ZNK8mystring6lengthEv

`int find(const mystring &rhs) const`	`_ZNK8mystring4findERKS_`
`int find(const char *s) const`	`_ZNK8mystring4findEPKc`
`const char *c_str() const`	`_ZNK8mystring5c_strEv`
`friend ostream &operator<<(...)`	`_ZlsRSoRK8mystring`

Hints / Tricks

1. It might be in your best interest to write the functions in C++ first. Then take each function individually, comment out the C++ implementation, and implement it in assembly. Otherwise, you will spend the majority of your time chasing bugs.

2. The << operator for the ostream class that prints a const char * is mangled to the nice, long function name below (notice PKc at the end--this means Pointer to Konstant char). The << operator for ostream must be given the ostream parameters as the *this* pointer, and your string pointer as the second parameter. For example, in C++, you'd write this as: s.operator<<(rhs.c_str()). Do NOT add an end line!

```
ostream << is: _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc@plt
```

3. You will notice that many parameters are constant. However, assembly does not have any method to enforce this, so you must take care that you do not change any of the values of a constant parameter.

4. You will need to compile your .S and .cpp test files together on the Hydra machines using the following command:

```
g++ -std=c++11 -o lab2 lab2.cpp lab2.S
```

A newer g++ compiler will not need any -std parameter.

5. Remember to write your code in the Intel syntax by starting your .S file with: .intel_syntax noprefix

Plagiarism

Please remember that this lab is an individual effort. Please review the **plagiarism policy in the course syllabus**.

Submission

Make sure you comment your assembly code. It will be imperative for the TAs to be able to follow your logic, especially if your function(s) do not work.

Submit your .S file on Canvas.

Lab 2

Criteria	Ratings				Pts
~mystring() *~mystring() must check the validity of the pointer before free'ing it, either through the pointer itself or the length of the string. The student may use C++ or C-style memory deallocation functions free@plt or _ZdlPv@plt.*	15.0 to >12.0 pts Excellent	12.0 to >9.0 pts Good	9.0 to >3.0 pts Fair	3.0 to >0 pts Unsat	15.0 pts
mystring constructors *mystring() must construct a 1-byte string if empty. This string will contain \0. The student may allocate this string either using C or C++-style functions, such as _Znwm@plt, calloc@plt, or malloc@plt.*	15.0 to >12.0 pts Excellent	12.0 to >9.0 pts Good	9.0 to >3.0 pts Fair	3.0 to >0 pts Unsat	15.0 pts
operator= functions *The rvalue operator= (&&) must move the pointer from the r-value to the l-value. The student must either exchange the pointers or set the r-value's pointer to 0 to ensure that the destructor does not delete a null pointer.*	15.0 to >12.0 pts Excellent	12.0 to >9.0 pts Good	9.0 to >3.0 pts Fair	3.0 to >0 pts Unsat	15.0 pts
length() function *strlen() is NOT permitted in this function, since the length of the string is kept as a separate member variable.*	5.0 to >4.0 pts Excellent	4.0 to >3.0 pts Good	3.0 to >2.0 pts Fair	2.0 to >0 pts Unsat	5.0 pts

Criteria	Ratings				Pts
find() functions *The function strstr() is permitted for this function. However, the return value must be converted from a memory address to an index, or -1 if the needle was not found.*	25.0 to >20.0 pts Excellent	20.0 to >15.0 pts Good	15.0 to >10.0 pts Fair	10.0 to >0 pts Unsat	25.0 pts
c_str() function *Student must return a const char *, which is essentially mString.*	5.0 to >4.0 pts Excellent	4.0 to >3.0 pts Good	3.0 to >2.0 pts Fair	2.0 to >0 pts Unsat	5.0 pts
friend << function *Student must use the ostream, mangled output for const char (given in the lab). They must not use printf or any other C-style function. Their function must not add a newline, and it must return the ostream object in rax.*	20.0 to >16.0 pts Excellent	16.0 to >12.0 pts Good	12.0 to >8.0 pts Fair	8.0 to >0 pts Unsat	20.0 pts
					Total Points: 100.0