Under construction!!!!

DMA and operator overload review

Workshop 8

In this workshop, you are to create a class that encapsulates string

LEARNING OUTCOMES

Upon successful completion of this workshop, you will have demonstrated the abilities to

- Dynamic Memory Management
- Operator overloading

SUBMISSION POLICY

The “in-lab” section is to be completed **during your assigned lab section**. It is to be completed and submitted by the end of the workshop. If you do not attend the workshop, you can submit the “in-lab” section along with your “at-home” section (a 20% late deduction will be assessed). The “at-home” portion of the lab is **due the day before your next scheduled workshop**

All your work (all the files you create or modify) must contain your name, Seneca email and student number.

You are responsible to regularly backup your work.

IN LAB:

Download or clone workshop 7 from https://github.com/Seneca-244200/OOP-Workshop8

STRING CLASS:

```
#ifndef SICT_STRING_H_
#define SICT_STRING_H_
#include <iostream>
const int ExpansionSize = 500;
namespace sict{
```

```
  class String{
    char* data_;
    int memSize_;
    int len_;
    void deallocate();
    void allocate(int size);
    void init(const char* str, int memSize);
  public:
    void resize(int newsize);
    int memSize()const;

    String();
    String(const char* str, int memsize = 500);
    String(const String& S);
    String& operator=(const String& S);
    virtual ~String();

    int length()const;

    operator const char*()const;
    // IO
    std::istream& read(std::istream& istr = std::cin);
    // operators

    String& operator+=(const char* str);
    String& operator+=(String& s);
    String& operator++();
    String& operator++(int);

    char& operator[](int index);


  };
  std::ostream& operator<<(std::ostream& ostr, const String &s);
  std::istream& operator>>(std::istream& ostr, String &s);
}


#endif
```

In String.cpp;
Complete the code of the class named **`String`** that encapsulates a string.

`void deallocate();`

Deletes the dynamic array of characters pointed by data_ and sets the pointer and the memSize attribute to nullptr and zero.

1- Deallocate memory pointed data_

2- Set the data_ attribute to null pointer

3- Set the memSize_ attribute to zero

`void allocate(int memsize);`

Deallocates the memory allocated by data_ and then allocates memsize memory and updates the memsize_ member variable.

1- Make sure memory pointed by data_ is deallocated.

2- Allocate memsize bytes and make data_ point to it

3- Set memsize_ attribute to memsize arg value;

`void init(const char* str, int memSize);`

This function is to avoid having the same code in the constructors, so make sure you understand that “init” can only be called when either the object is just created (in a constructor) or the object is in a safe empty state.

Init allocates memSize memory if memSize is big enough to hold the c-string pointed by str, otherwise it will reset the memSize argument to the length of the str + 1 and then does the allocation.

Afterwards it will copy the str into the newly allocated memory.

Init also makes sure memSize_ and len_ member variable have accurate values.

1- Set the data_ attribute to null

2- if memsize is smaller than the length of the string, we will set the memsize arg to the length of the string + 1

3- allocate memsize bytes pointed by data

4- copy str to data_

5- set len_ to the proper values;

```
void resize(int newsize);
```

Resizes the memory pointed by data_ keeping the c-string inside data_ intact.

1- allocate memsize bytes pointed by a temp char pointer

2- if data_ is not null copies the string pointed by data_ pointer character by character into the newly allocated memory up to the length of the string in data_ or memsize-1; whichever comes first.

3- Null terminates the string pointed by temp.

4- Deallocates old memory pointed by data_

5- Makes data_ point to where temp is pointing. (copies the address kept in temp into data_)

6- Updates memSize_ and len_ the their new values.

```
String();
```

No argument constructor sets the data_ attribute to nullptr and other member variables to zero (puts the object in a safe empty state).

```
String(const char* str, int memsize = 500);
```

Initializes the object using str and memsize values through the **init** function.

```
String(const String& S);
```

Initializes the object using S.data_ and S.memSize_ values through the **init** function.

```
String& operator=(const String& S);
```

If the object is not set to itself **(this != &S)**, First it will deallocate the already existing memory and then Initializes the object using S.data_ and S.memSize_ values through the **init** function.

Afterwards, it will return the reference of the current String object.

```
virtual ~String();
```

Deallocates the memory pointed by data_.

```
int length()const;
```

Returns len_

```
operator const char*()const;
```

When casted to a **const char*** this object returns the address kept in **data**_ member variable.

```
String& operator+=(const char* str);
```

If the size of the allocated memory permits, this operator overload concatenates the c-string **str** to the end of **data**_ cstring. If the memSize is less than the sum of length of two strings +1, then it will resize itself to the exact same size (the sum of two +1) and then does the Concatenation.
At the end it will return the reference of the current String object.

1- Keep the length of the **str** c-string in a temp variable
2- If the sum of two lengths + 1 is greater that **memSize**_ resize the memory to the sum of two lengths +1
3- Concatenate the **str** argument to the end of **data**_ using ***strcat.***
4- Update **len**_ to the new length.
5- Return the reference of the current String object.

```
String& operator+=(String& s);
```

Reuses the **operator+=(const char* str)** passing **s.data**_ as the **str** argument.

```
String& operator++();
```

Adds a space before the string.
Callenge! Let me see how you do this…

```
String& operator++(int);
```

Adds a space after the string.
Use **operator+=** and add a space.

```
char& operator[](int idx);
```

Returns the reference of the character of the data_ array sitting at the **idx** index. If the index is out of the range of the length of the string, this operator should resize the object to idx + ExpansionSize. ExpansionSize is the constant integer defined in String.h

```
operator int()const;
```

When casted to an integer, the length of the **data**_ c-string is returned.

```
std::istream& read(std::istream& istr = std::cin);
```

This is probably the most complicated part of this class.

Instead of getting the string using getline or cin>>, this function gets the string character by character and if the number of characters reaches the memSize_ value, it will resize the object to memSize_ + ExpansionSize.

All the characters are copies into **data**_ string until **'\n'** is reached. At this point a NULL is copied to the data_ c-string to null-terminate the array.

.

SUBMISSION

To test and demonstrate execution of your program use w7_in_lab.cpp.

If not on matrix already, upload **String.h , String.cpp** and **w7_in_lab.cpp** to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account:

```
Section SCC and SDD:
~fardad.soleimanloo/submit w8_at_home <ENTER>
```

and follow the instructions.