Workshop 10

Dynamic Memory Allocation and Operator Overload review

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.

# At-Home

Download or clone workshop 10 from <https://github.com/Seneca-244200/OOP-Workshop10>

# String Class

|  |
| --- |
| #ifndef ICT\_STRING\_H\_  #define ICT\_STRING\_H\_  #include <iostream>  const int ExpansionSize = 500;  namespace ict  {  class String  {  char\* m\_data;  int m\_memSize;  int m\_len;  void deallocate();  void allocate(int size);  void init(const char\* str, int memSize);  public:  void resize(int newsize);  // special member functions  String();  String(const char\* str, int memsize = 500);  String(const String& str);  String& operator=(const String& str);  virtual ~String();  // accessors  int getLength() const;  int getMemSize() const;  // cast operators  operator const char\*() const;  operator int() const;  // operators  String& operator++();  String operator++(int);  String& operator+=(const char\* str);  String& operator+=(String& s);  String operator+ (const String& str) const;  char& operator[](int index);  // IO  std::istream& read(std::istream& = std::cin);  };  std::ostream& operator<<(std::ostream&, const String&);  std::istream& operator>>(std::istream&, String&);  }  #endif |
| -><- len: 0, mem: 0  ->Two<- len: 3, mem: 10  ->Three<- len: 5, mem: 6  ->Three<- len: 5, mem: 6  ->Thirty One<- len: 10, mem: 500  Thirty One  ->One<- len: 3, mem: 4  ->One <- len: 4, mem: 5  ->One Two<- len: 7, mem: 8  ->One Two<- len: 7, mem: 8  ->One Two Three<- len: 13, mem: 500  -> One Two Three<- len: 14, mem: 500  > One Two Three<  ->XOneXTwoXThree<- len: 14, mem: 500 |

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 m\_data and sets the pointer and the m\_memSize attribute to nullptr and zero, respectively.

1. Deallocate memory pointed m\_data;
2. Set the m\_data attribute to null pointer;
3. Set the m\_memSize attribute to zero.

**void** allocate(**int** memsize);

* Deallocates the memory allocated by m\_data and then allocates memsize memory and updates the m\_memSize member variable.

1. Make sure memory pointed by m\_data is deallocated;
2. Allocate memsize bytes and make m\_data point to it;
3. Set m\_memSize attribute to memsize argument 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 m\_memSize and m\_len member variable have accurate values.

1. Set the m\_data attribute to null;
2. If memsize is smaller than the length of the string, set the memsize argument to the length of the string + 1;
3. Allocate memsize bytes pointed by data;
4. Copy str to m\_data;
5. Set m\_len to the proper value.

**void** resize(**int** newsize);

* Resizes the memory pointed by m\_data, keeping the C-string inside m\_data intact.

1. Allocate newsize bytes and stores is in a temporary char pointer;
2. If m\_data is not null, copies the string pointed by m\_data pointer character by character into the newly allocated memory up to the length of the string in m\_data or newsize - 1; whichever comes first;
3. Null terminate the string stored in the temporary array;
4. Deallocate old memory pointed by m\_data;
5. Make m\_data point to the temporary array (copies the address kept in temporary pointer into m\_data) ;
6. Update m\_memSize and m\_len the their new values.

String();

* No argument constructor; sets the m\_data attribute to null 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& other);

* Initializes the object using other.m\_data and other.m\_memSize values through the init(…) function.

String& **operator**=(**const** String& other);

* Assignment operator. If the object is not set to itself, it will deallocate the already existing memory and then initialize the object using other.m\_data and other.m\_memSize values through the init(…) function.
* Afterwards, it will return the reference of the current object.

**virtual** ~String();

* Deallocates the memory pointed by m\_data.

**int** getLength() **const**;

* Getter; returns m\_len.

**int** getMemsize() **const**;

* Getter; returns m\_memSize.

**operator** **const** **char**\*()**const**;

* Cast operator. When converted to a **const** **char**\*, the address kept in m\_data member variable is returned.

**operator** **int**() **const**;

* Cast operator. When converted to an integer, the length of the m\_data is returned.

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

* Concatenates two strings.
* If the size of the allocated memory permits, this operator overload concatenates the str to the end of m\_data. If the m\_memSize is less than the sum of lengths of the 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 object.

1. Keep the length of the str in a temporary variable;
2. If the sum of two lengths + 1 is greater that m\_memSize, resize the memory to the sum of two lengths + 1;
3. Concatenate the str argument to the end of m\_data using strcat function (available in cstring library);
4. Update m\_len to the new length;
5. Return the reference of the current object.

String **operator**+(**const** String& str) **const**;

* Concatenates two strings.
* Out of the m\_data of the current object, create a temporary String like this: String temp(m\_data) and then reuses **operator**+= to do the concatenation.

String& **operator**+=(String& str);

* Reuses the **operator**+=(**const** **char**\*) passing str.m\_data as the argument.

String& **operator**++();

* Unary operator; adds a space *before* the string.

String **operator**++(**int**);

* Unary operator; adds a space *after* the string.
* Use **operator**+= to add a space.

1. Make a copy of the current object;
2. Add a space;
3. Return the copy instead of \***this**.

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

* Indexing operator.
* Returns the reference to the character of the m\_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).

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

* Instead of getting the string using getline or >>, this function gets the string character by character. If the number of characters reaches the m\_memSize value, it will resize the object to m\_memSize + ExpansionSize.
* All the characters are copied into m\_data string until \n is reached. At this point a \0 is copied to the m\_data C-string to null-terminate the array.

# Submission

To test and demonstrate execution of your program, use w10\_at\_home.cpp.

If not on matrix already, upload String.h, String.cpp and w10\_at\_home.cpp to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account (and follow the instructions):

**~profname.proflastname/submit w10\_at\_home <ENTER>**