ENSF 337 Fall 2018: Tutorial 10

Problem I - Consider the following class definition and write the implementation of the `constructor` that receives a vector argument and the implementation of member function `append`.

```
class String {
public:
   String(); // PROMISES: length() == 0.
   String(const char *s);
   String(const vector<char>& v);
  ~String();
   int length() const;

   void append(const vector<char> & tail);
   // PROMISES: If tail.size() == 0 or tail.at(0) == '\0', nothing happens. Otherwise, the string will be lengthened by appending all the chars of tail
   //  to the end of  string or up to the first occurrence of a '\0' in the tail (whichever comes first).

private:
   char *storageM;
   int lengthM;
};
String::String(const vector<char>& v){

    int size = (int)v.size();
    int length = 0;
    while( length < size && v.at(length) != '\0')
        length++;

    if(size > length) size = length;

    lengthM = size;
    storageM = new char[lengthM + 1];
    int i;
    for(i = 0 ; i < lengthM; i++)
        storageM[i] = v.at(i);
    storageM[i] = '\0';

}

void String::append(const vector <char>& tail)
{
    if(tail.size() != 0) {

        int size = (int)tail.size();
        int length = 0;
        while(length < size && tail.at(length) != '\0' )
            length++;
        if(size > length) size = length;

        int len = lengthM + size;
        char *stor = new char[len + 1];
        char *p = stor;
        for(int i = 0; i < lengthM; i++, p++)
            *p = storageM[i];

        for (int i = 0; i < size; i++, p++) {
            *p = tail.at(i);
        }
        *p = '\0';

        lengthM = len;
        storageM = stor;

    }
}
```

Problem II: Consider the following main function, and write a function called `resize_array` that dynamically resizes the number of elements of an array and preserves its existing values of the array up to the new size. If the new size is less than current size extra elements will be set to zero.

```
int main() {
  int n = 4;
  int *p = new int[n];
  display(p, n);
  assert(p != nullptr);
  resize_array(p, n + 2);
  display(p, n + 2);
  return 0;
}
```

```
void resize_array(int ** x, int n, int m) {
   int *p = *x;
   *x = new int[m];
   if( m <= n)
     for(int I = 0; I < m ; I++)
      (*x)[I] = p[I];
   else {
     int I;
     for(I = 0; I < n; I++)
       (*x) [I] = p [I];
     for(int J = I; J < m; J++)
       (*x) [J] = 0;
   }

   delete [] p;
}
```