National University of Singapore
School of Continuing and Lifelong Education

TIC1001: Introduction to Computing and Programming I
Semester I, 2019/2020

# Tutorial 6
# Strings and Vectors

1. `string`s and `vector`s are passed by value copy, in and out of functions. This is different from C-strings and arrays, which are passed by pointers.

   Consider the following functions:

```
char * capitalize(char *s) {
    for (int i = 0; s[i]; i++)
        if ('a' <= s[i] and s[i] <= 'z')
            s[i] -= 32;
    return s;
}

string capitalize(string s) {
    for (int i = 0; i < s.size(); i++)
        if ('a' <= s[i] and s[i] <= 'z')
            s[i] -= 32;
    return s;
}
```

   *Note: in C++ it is ok to have functions of the same name, provided the input signature is different. The compiler will match the type of the arguments to determine which function to call.*

   (a) What is the output when the following lines of code are executed? Try to figure it out before running the program.

```
char cstring[20] = "Hello World!";
char *new_cstr;
string stdstring = "Hello World!", new_stdstr;

new_cstr = capitalize(cstring);
new_stdstr = capitalize(stdstring);

printf("C-string: %s\n", cstring);
printf("New C-string: %s\n", new_cstr);
cout << "std::string: " << stdstring << endl;
cout << "New std::string: " << new_stdstr << endl;
```

   (b) What happens when the signature of the second function is changed to

   i. `string capitalize(string &s)`

   ii. `string &capitalize(string &s)`

(c) Come up with some code to demonstrate the same effect with arrays and vectors.

2. `string`s and `vector`s are also assigned by copy. What does this mean and how is this different from C-strings and arrays? Try to come up with some code to demonstrate the difference.

3. Strings and vectors include the member functions `.insert` and `.erase`. As shown in lecture, they can be fairly complicated to use because they have to be used in conjunction with the `.begin` and `.end` functions.

   However, it is not entirely necessary to depend on these functions to insert and erase elements from a string/vector. We can always write our own insert and erase function to suit our needs as such:

   ```
   vector<int> my_vector = {0, 1, 2, 3, 4, 5};
   insert(my_vector, 2, 10);  // inserts into index 2
   erase(my_vector, 5);       // removes element at index 5

   for (int i = 0; i < my_vector.size(); i++)
       cout << my_vector[i] << " ";
   ```

   This will result in an output of `0 1 10 2 3 5`.

   Provide an implementation for the functions `insert` and `erase` using only the member functions `.push_back`, `.pop_back` and `.size`.

4. The function `void mutate` takes in a vector of int, and sets each element to the sum of its neighbours, with the ends wrapping around.

   For example, the vector `0, 1, 2, 3, 4, 5` will mutate to `1+5, 0+2, 1+3, 2+4, 3+5, 4+0`.

   Provide an implementation for the function `void mutate`.