VG101 Final RC Part 2

STL: Standard Template Library

powerful toolkit for everyone

`std::vector`

STL vector realizes a dynamic array container so that we could use it as normal arrays, plus more functions like `insert()` and `push_back()` to add elements, and `erase()` to remove elements.

- Pros:
  - Array of given type that can automatically increase size
  - Can be returned from sub function while values remain
  - Access with `[]` operator as array
  - Fast random access (e.g. `vec[101]`) $O(1)$
  - Fast insert/delete at the back (`push_back`, `pop_back`) $O(1)$
- Cons:
  - Inserting / deleting at other position is slow (`insert`, `erase`) $O(n)$
  - But usually this does not matter in VG101

Believe me, at least in VG101, `vector` can meet all your expectations for an array. So please get familiar with `vector` and frequently use it in your homework/lab.

- How to use it?
- ```
  #include <vector>
  using namespace std;

  vector<int> vec1;     // holds int
  vector<Vector2> vec2; // holds Vector2
  vector<string> vec3;  // holds string
  ```
- Initialization:
  ```

- ```
  vector<T> v1;          // empty vector v1
  vector<T> v2(v1);      // copy constructor, v2 == v1
  vector<T> v3(n, t);    // construct v3 that has n elements with value
  t
  vector<int> v(10, 0); // example
  // Similar to:
  int arr[10] = {0};
  ```

- Size:
  - `v.size()` returns a value of `size_type` corresponding to the vector type.
  - Example: `vector<int>::size_type`
    - a companion type of vector (to make the type machine-independent).
    - essentially an **unsigned** type, so it can be directly converted to `unsigned int` but not `int`.
    - C style cast: `int s = (int)v.size();`
    - C++style cast: `int s = static_cast<int>(v.size());`
  - Check whether the `vector` is empty: `v.empty()`.
- Add/Remove:
  - `vec.push_back(t)`: add element `t` to the end of `vec`.
  - Elements are copies: no relationship between the element in the container and the value from which it was copied.
  - `vec.pop_back()`: remove the last element in `vec`. `vec` must be non-empty.
- Other useful operations:
- ```
  v1 = v2;   // copy assignment
  v.clear(); // clear all elements, size = 0
  v.front(); // The first element of v, must be non-empty
  v.back();  // The last element of v, must be non-empty
  ```

Iterator

Iterators are a **generalization of pointers**: they are objects that point to other objects.

All STL containers define iterator types:

- Declaration: `vector<int>::iterator it;`
  - `v.begin()` returns an iterator to the first element in the vector.
  - `v.end()` returns an iterator that refers to the **next position after the end** of the vector.
  - Usually used to indicate when we have processed all the elements in the vector.
  - If the vector is empty, then `v.begin() == v.end()`.
- Operations:

- Dereference: can read/write through `*iter` (cannot dereference the iterator `v.end()`)
- `++iter`, `iter++`: next iterator (cannot increment the iterator `v.end()`)
- `--iter`, `iter--`: go back to the previous iterator
- `iter == iter1` and `iter != iter1`: check whether two iterators point to the same data item

```
vector<int>::iterator begin = ivec.begin();
auto end = ivec.end(); // Thanks to C++11.
while (begin != end) {
  cout << *begin++ << " ";
  // 1. get the value of *begin
  // 2. cout << *begin << " ";
  // 3. begin++;
}
```

- Iterator Arithmetic
  - `iter + n`, `iter - n`, where `n` is an integer

```
// Example 1: Go to the middle
auto mid = v.begin() + v.size() / 2;

// Example 2: Random access through iterator
auto begin = v.begin();
cout << *(begin + 7) << endl;
cout << v[7] << endl; // Same
```

- Relational Operation: `>`, `>=`, `<`, `<=`, `==`, `!=`
  - To compare, iterators must refer to elements **in the same container**.

```
// Example: Traverse a vector through iterator
for (auto it = v.begin(); it != v.end(); ++it) {
    cout << *it << endl;
}

// C++11 style: for-range based loop
for (auto &item : v) {
    cout << item << endl;
}
```

- More about initialization of `vector`
  - `vector<T> v(b,e)`: create vector `v` with a copy of the elements from the range denoted by iterators `b` and `e`.

```
vector<int> v1(10, 5);
vector<int> v2(v1);
vector<int> v3(v1.begin(), v1.end());
vector<int> v4(5, 5);
vector<int> v5(v1.begin(), v1.begin() + v1.size()/2);
// v1, v2, v3 are the same
// v4, v5 are the same
```

    - You can even use array to initialize vector:

```
int a[] = {1, 2, 3, 4};
unsigned int sz = sizeof(a) / sizeof(int);
vector<int> vi(a, a + sz); // pointer
```

- More about add/remove:
    - `v.insert(it, t)`, `it` **is an iterator**
        - Insert an element with value `t` **right before** the element referred to by iterator `it`.
        - Return an iterator referring to the element that was added.
    - `v.erase(it)`, `it` **is an iterator**
        - Remove the element that iterator `it` refers to.
        - Return an iterator referring to the element after the deleted one, or `v.end()` if `it` refers to the last element.
- **Learn and practice by yourself: https://en.cppreference.com/w/cpp/container/vector**

`std::string`

Besides `std::vector`, C++ also provides an useful string library `<string>`. Also, at least in VG101, I think you can always use `string` rather than C-style string `char[]`.

- Pros:
    - No need to dynamic memory allocation
    - Automatically size increasing
    - Can be return from sub function while value remain
    - No need to worry about ending `\0`
    - Access with `[]` as C-style string
    - Allow `+` to concat strings
    - Allow `==` to compare strings
    - **More powerful with** `stringstream`
- Initialization:

- ```
  #include <string>
  using namespace std;

  string str1 = “blablabla”; // Overload assignment operator
  string str2(“blablabla”);  // Copy constructor
  ```
- The `string` can automatically store infinite numbers of characters without worrying about memory leak. `cin` and `cout` can also take string as parameters.
- Other useful and straight-forward operations (No longer anti-human like `strcmp`!)
  - Assignment:
    - C++ `string`: `str1 = str2;`
    - C-style string: `strcpy(str1, str2);`
  - Concatenate:
    - C++ `string`: `str3 = str1 + str2;`
    - C-style string: `strcpy(str3, str1); strcat(str3, str2);`
  - Compare:
    - C++ `string`: `str1 == str2`, `str1 > str2`, ...
    - C-style string: `strcmp(str1, str2) == 0`, ...
  - Get length:
    - C++ `string`: `str.length();` or `str.size();` they are the same
    - C-style string: `strlen(str);`
    - This is also an example of the OOP style.
  - Convert to a C-style string (so it’s compatible with C library): `str.c_str();`
- Actually, `string` is also a STL container like `vector`, so it has the iterator, and some methods similar to the `vector`
  - `opeartor[]`: access string as a char array, e.g. `str[10]` -> char
  - Check whether the `string` is empty: `str.empty()`.
  - Methods quite similar to `vector`: `str.front()`, `str.back()`,`str.push_back`, `str.pop_back()`, ...
  - iterator-based: `str.insert()`, `str.erase()`, ...
  - Special and useful methods: `str.append()`, `str.substr()`, `str.find()`, ...
- **Learn and practice by yourself: https://en.cppreference.com/w/cpp/string/basic_string**
- Useful non-member functions:
  - `stoi()`: convert string to number
  - `getline()`: taking a `istream` and a `string` as argument, read a line from `istream` and store it into `string`
- The best way to get familiar with them: 1. Read the documentation, 2. Try it by yourself!
  ```

`<algorithm>` Library

- `#include <algorithm>`
- Jigang introduced some methods in Lecture 20, you can check the slide or recording if you missed it.
- Extremely **useful** in future programming.
- Examples provided will be a good starting point.
- `sort`: http://www.cplusplus.com/reference/algorithm/sort/
- `find`: http://www.cplusplus.com/reference/algorithm/find/
- `swap`: http://www.cplusplus.com/reference/algorithm/swap/
- `count`: https://www.cplusplus.com/reference/algorithm/count/
- `max_element`: http://www.cplusplus.com/reference/algorithm/max_element/
- `min_element`: http://www.cplusplus.com/reference/algorithm/min_element/
- `reverse`: http://www.cplusplus.com/reference/algorithm/reverse/
- `remove`: http://www.cplusplus.com/reference/algorithm/remove/

I love `getline + stringstream`

`#include <sstream>`

Reverse the words in a sentence (Using `stringstream`)

Practice

- Reverse the words in a sentence, for example

  “this is an example” ➔ “example an is this”

See `reverse.cpp`

```cpp
#include <iostream>
#include <algorithm>
#include <sstream>
#include <string>
#include <vector>

using namespace std;

int main() {
    vector<string> vec;
    string str, ans = "";
    cout << "Input string: ";
```

```
    getline(cin, str);
    istringstream input(str);
    while (input >> str) {
        vec.push_back(str);
    }

    reverse(vec.begin(), vec.end()); // reverse the vector
    for (auto &s : vec) {
        ans += s + " "; // traverse the vector, and append to the answer
string
    }
    ans.pop_back(); // delete the last white space

    cout << "Output string: " << ans << endl;
    return 0;
}
```

Count how many numbers in a string (Using `stringstream`)

2. (20 points) Finish the function "void P2()" to accept user input of several numbers, which are separated by spaces, then count how many numbers and display the result. For example, if the user input 1.2 5 34 6.4 -2.33, the program would display There are totally 5 numbers. Note that there might be more than one spaces between the numbers, and there might be spaces before the first number or after the last number. The following is an example run of the function (where the "Input some numbers: " is the prompt to accept user input):

Input some numbers: 1.2 5 34 6.4 -2.33
There are totally 5 numbers.

See `count.cpp`

```
#include <iostream>
#include <sstream>
#include <string>

using namespace std;

int main () {
    cout << "Input some numbers: ";
    string str;
    getline(cin, str);
    istringstream input(str);
    int count = 0;
    double num;
    while (input >> num) {
        ++count;
```

```
    }
    cout << "There are totally " << count << " numbers." << endl;
    return 0;
}
```

Reminders (Not only for exam, but also for future programming)

- Keep a good coding style (indent, curly bracket, ...)
  - It's highly recommended to get help from the IDE/Editor's code formatter.
  - For example, **Ctrl + Alt + L** in Clion on Windows.
- Use the given prototype for your functions.
- Don't ask user input anything/output anything unless the Jigang asks you to do so!
- Get familiar with `class`, `vector`, `string` and `stream I/O`.
  - They are exactly all you need to know to complete past year's final exam.
- **Learn by yourself**
- **Debug by yourself**
- **Search by yourself**
- **Enjoy yourself**
- The thing I regret most is that I didn't systematically teach you how to debug your programs 😟