C++ and tools

Geany

preferred text editor to write C++ code

C++ Basics

Sample program:

```
#include <iostream>

using namespace std; // this is a comment (ignored by the compiler)

int main() {
    cout << "Welcome to the SOI workshop!\n";
    /* this is a multi-
    line comment */
}
```

Data types:

```
int a = -6;
long long int b = 12345678987654321;
float c = 1.234;
double d = 2.3456;
char e = 'w';
```

Strings: (requires `#include <string>`)

```
string s = "hello";
cout << s[1]; // prints 'e'
```

for-loop:

```
for (int i = 4; i < 8; i++) {
    cout << i << "\n";
}
```

while-loop:

```
int i = 4;
while (i < 8) {
  cout << i << "\n";
  i++;
}
```

do-while-loop:

```
int i = 4;
do {
  cout << i << "\n";
  i++;
} while (i < 8);
```

C++ Input/Output

Reading a value from standard input: `cin >> myVariable;`

Printing a value to standard output: `cout << myExpression;`

If input is very large, do `ios_base::sync_with_stdio(false);` at the beginning of `main`

Terminal (= command line, = shell) 🖳

Help pages ("manual"):

- for commands (such as `diff`, `echo`, `grep`, …): `man 1 COMMAND_NAME`
- for C functions (such as `fprintf`, `sin`, `atan2`, …): `man 3 c_function_name`

Paths:

- Absolute paths: e.g. `/home/stofl/programming`
- Relative paths start with `.` or `..`
    - `.` means current directory ("working directory")
    - `..` means "parent directory"

Some commands:

- `cat file1 file2 …` "concatenate": Show contents of `file1`, `file2`, …
- `pwd` "print working directory": show current directory
- `ls` "list": show content of current directory
  (`ls -1` 1 column only, `ls -l` with details, `ls -a` include hidden files)
- `cd PATH` "change directory": change to directory at `PATH`
- `diff file1 file2` "difference": compare `file1` and `file2`

Redirecting output to a file: `./myprogram > ./outputfile.txt`

Sending a file to the input of a program: `./myprogram < ./inputfile.txt`

Piping output of a program to another program: `./myprogram | grep -e 'hello'`

Keyboard shortcuts:

- Use TAB for auto-completion instead of typing everything
- Use Arrow-up/down to navigate through commands history
- Use Ctrl+R to search commands history
- Copy/paste: Ctrl+Shift+C / Ctrl+Shift+V
- Kill program: Ctrl+C

Compiling C++ with GCC

```
g++ -Wall -Wextra hello.cpp -o hello
```

Useful options:

- `-Wall` and `-Wextra` to enable all warnings
- `-lm` if you use math functions
- `-O2` optimizations level 2 (used on the grader)
- `-D_GLIBCXX_DEBUG` nicer runtime error messages instead of just „Segmentation fault“

C++ I/O Sample

Reading the input map of the task „oilspill“:

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

using namespace std;

int h;
int w;

string area[100];

int main() {
    // read input
    cin >> h >> w;
    string line;
    cin >> line; // consume end-of-line
    for (int i = 0; i < h; i++) {
        cin >> area[i];
    }

    // print it differently
    for (int i = 0; i < h; i++) {
        for (int j = 0; j < w; j++) {
            if (area[i][j] == '~') {
                cout << 'w';
            } else {
                cout << 'i';
            }
        }
        cout << "\n";
    }
}
```

C++ Standard Template Library

Depending on what you use, you need to import (some of) the following headers:

```
#include <vector>
#include <stack>
#include <queue>
#include <set>
#include <map>
```

Vector: „the better array“, resizable. The `push_back` operation runs in amortized constant time.

```
// new vector of size 0
vector<int> v;
v.push_back(10);
v.push_back(11);
cout << v[0] << " " << v[1] << " ";
cout << v.size() << "\n";
// change the size:
v.resize(10);
```

Stack: LIFO (last in, first out)

```
stack<int> s;
s.push(10);
s.push(11); // s = [10, 11]
cout << s.top(); // 11
s.pop(); // s = [10]
s.push(12); // s = [10, 12]
```

Queue: FIFO (first in, first out)

```
queue<int> q;
q.push(10);
q.push(11); // q = [10, 11]
cout << q.front(); // 10
q.pop(); // q = [11]
q.push(12); // q = [11, 12]
```

Sets: implemented using a tree-like data structure, `insert`, `find`, & `erase` runs in $O(\log n)$ time

```
set<int> s;
s.insert(10);
s.insert(11);
s.insert(11); // s = {10, 11}
s.erase(11); // s = {10}
if (s.find(10) != s.end()) {
  cout << "10 is in the set!\n";
}
for (
  set<int>::iterator iter = s.begin();
  iter != s.end();
  iter++
) {
  cout << *iter << '\n';
}
```

Maps (dictionaries, key/value pairs): `[…]`, `find` & `erase` runs in $O(\log n)$ time

```
map<string, double> m;
m["bananas"] = 3.90;
m["apples"] = 3.20;
if (m.find("apples") != m.end()) {
  cout << "apples cost " << m["apples"] << '\n';
}
for (
  map<string, double>::iterator iter = m.begin();
  iter != m.end();
  iter++
) {
  cout << iter->first << " cost " << iter->second << '\n';
}
m.erase("apples");
```

Priority queues: Smallest element (= highest priority) comes first

```
priority_queue<int> pq;
pq.push(11);
pq.push(10); // pq = [10, 11]
cout << pq.top(); // 10
pq.pop(); // pq = [11]
pq.push(12); // pq = [11, 12]
pq.push(9); // pq = [9, 11, 12]
```

STL Reference: http://cppreference.com/, http://www.cplusplus.com/reference/