C++ Features

2015 IOI Camp 1

Robert Spencer

February 21, 2015

C++ Introduction

- Built off c
- Endowed with OOP
- Rich STL (more later)
- Fast
- Not as class besotted as Java

#define (macros)

- Make code shorter
- Useful for constant declarations
- Useful for common functions
  - Always put parameters in brackets
- Recommend try use ALL CAPS

```
#define INFINITY 100000000
#define FORI(_st,_en) for(int i = (_st);i<(_en);i++)
#define mp make_pair

#define DEBUG 1

if (DEBUG)
  cerr<<status<<'\n';
```

Pointers

- Allow access to memory
- Powerful and dangerous
- Arrays
- & and *

```
int R = 5;
int* pR = &R;
*pR == 5;

char[10] str = "HelloWorld";   // str is a char*
int* arr = new int[100];       // int array
int* arr2 = new int[R];

// WOW: arr[4] is the same as 4[arr]
// WHY???  Because arr[4] == *(arr + 4)
//                        == *(4 + arr)
//                        == 4[arr]
```

References

- “Safe pointers”
- Save your stack!

```
int A = 5;
int& rA = A;
rA = 6;           // Now A = 6

int foo(int x, int& result)
{
  result = x*x;
}

foo(3,A);         // Now A = 9

int bar(vector<int>& vec);
int foobar(vector<int> const & vec);
```

Templates

- Attempt to get over static typing
- Allow same code for many types
- Can get quite complex (but not often used in contest except for STL)

```
template <typename T>
T max(T a, T b)
{
  if (a>b) return a;
  return b;
}

max<int>(2,5);            // The integer 5
max<double>(4.0,2.1);     // The double 4.0
max<double>(4,2.1);       // The double 4.0
max(3,5);                 // The (implicit) integer 5
```

Standard Template Library

- Large collection of useful things (containers and algorithms mainly)
- Entire lecture on its own
- Must have in olympiad toolkit
- Most elements defined using templates - huge versatility (eg nested containers)
- Elements defined in *namespace* `std`

```
#include <queue>

using namespace std;

queue<int> shoppingQueue;
```

Standard Template Library - `vector`

- Dynamically resizing array
- Amortised constant operations (if you are worried)
- `[]`, `size`, `empty`, `front`, `back`, `push_back`, `pop_back`,`resize`

```
#include <vector>
using namespace std;

vector<int> arrayOfInt;
vector< vector<int> > arrayOfArraysOfInt;

arrayOfInt.push_back(4);
arrayOfArraysOfInt.resize(100);
arrayOfArraysOfInt[49] = arrayOfInt;
```

Standard Template Library - pair

- Two tuple
- Templatised in two classes
- No operations (except equality comparison)
- `first`, `second`

```
#include <utility>
using namespace std;

pair<char,int> pr;

pr = make_pair<char,int>('x',1);
pr = make_pair('y',2);
pr.first;                               // 'y'
pr.second;                              // '2'
```

Standard Template Library - map

- Associative container
- Keys and values
- [], size, empty, insert, erase

```
#include <map>
using namespace std;

map<char,int> mp;

mp.insert(make_pair('a',4));
mp['a'] == 4;                    // True
mp['b'] = 7
```

Standard Template Library - iterators

- Used for traversing containers. Fancy pointers.
- begin, end and ++
- Never compare > or <, only ==
- Dereference to get value.

```
#include <vector>
using namespace std;

vector<int> v;

for (vector<int>::iterator it  = v.begin();
                           it != v.end();
                           ++it)
  cout<<*it<<endl;

for (auto it = v.begin(); it!=v.end() ; ++it)
  cout<<*it<<endl;
```

Standard Template Library - `algorithm`

- Lots of nice functions.
- Look up usages
    - `for_each`
    - `find`, `find_if`
    - `count`
    - `sort`
    - `make_heap`, `push_heap`, `pop_heap`
    - `min`, `max`, `min_element`, `max_element`

```
#include <algorithm>
using namespace std;
```