C++ basic templates.

Vitalii Drohan

Examples.

```
template <typename T>
const T& min(const T& a,const T&b){
  return a < b ? a : b;
}

int main(){
  int b = min(10,11);
  double c = min<double>(11.1,12.1);
  return 0;
}
```

```
template<typename T>
class SmartPtr{
public:
  SmartPtr(T* obj) : obj(obj) {};
  ~SmartPtr(){ if(obj){ delete obj; } };
private:
  T* obj;
};

int main(){
  SmartPtr<int> sp(new int);
  return 0;
}
```

Examples.

```
template <int N>
struct Fact{
  const static int val = Fact<N-1>::val*N;
};

template<>
struct Fact<1>{
  const static int val = 1;
};

int main(){
  int a = Fact<5>::val; //a = 120
  return 0;
}
```

Tips.

Code is generated during compilation. What is not used it not generated.

Implemented in header(.h) files.

Type of template argument is often deduced.(no need to write <...,...,...>)

Usually templated code is written by library developers and used by other programmers.

One can specialize templated object for specific arguments + generate different code based on type traits(property of types, is movable? is integral? is copyable?)

Could be used to write programs that are executed during compilation(metaprogramming).

STL.

Important templated code available is C++ standard library. Used everywhere.

STL = (Containers + Iterators) + Algorithms

Containers.

<vector> - dynamic array

<array> - fixed size array

<map> - key -> value storage

<stack> - push / pop (top)

<queue> - push / pop (front)

<set> - unique storage of elements

...

Iterators

Iterators - object that is like pointers (can be dereferenced). Used for iteration over containers.

C-style Pointer is iterator for C array:

It is forward iterator: ++Pointer

also backward: --Pointer

It is random access iterator: Pointer + 5 (will point to 5th element after Pointer.)

Pointer1 - Pointer2 = distance between elements 1 and 2

C-pointer as iterator

```
#include <iostream>
int main(){
  int arr[5] = {1,2,3,4,5};

  for(int* it = arr; it != arr + 5; ++it){
    std::cout << *it << '\n';
  }

  return 0;
}
```

STL containers iterators.

(some container).begin() - iterator pointing to first element(or to end())

(some container).end() - iterator pointing to one element after last one.

cbegin() , cend() - constant versions of this

rbegin(), rend() - reverse iterators

(!) Iterators used for deleting elements from containers while iterating.

Vector iterator

```
#include <iostream>
#include <vector>

int main(){
  std::vector<int> vec = {1,2,3,4,5};

  for(auto it = vec.begin(); it != vec.end(); ++it){
    std::cout << *it << '\n';
  }

  return 0;
}
```

Algorithms.

<algorithm> - header

http://www.cplusplus.com/reference/algorithm/

Usually arguments to algorithm - iterators.

Sort

```
bool compare(const T& a,const T& b); // should work as < to sort like -> 1,2,3

 std::vector<T> vec;
  std::sort(vec.begin(),vec.end());
  std::sort(vec.begin(),vec.end(),compare);
  std::sort(vec.begin(),vec.end(),
            [ ](const T& a,const T& b){
              return a < b;
            }
            );
```

Popular algorithms

sort(begin,end, predicate)

find_if(begin,end, predicate), find(begin, end, value)

copy(from_begin, from_end, to_begin)

to remove elements on vector(remove_if + vector::erase)

min(a,b), max(a,b), swap(a,b)

fill(begin,end, val), generate(begin,end, generate function)

accumulate from <numeric>

rotate, partition