LAB 10

STL-Container

2024/5/17

G++ optimization

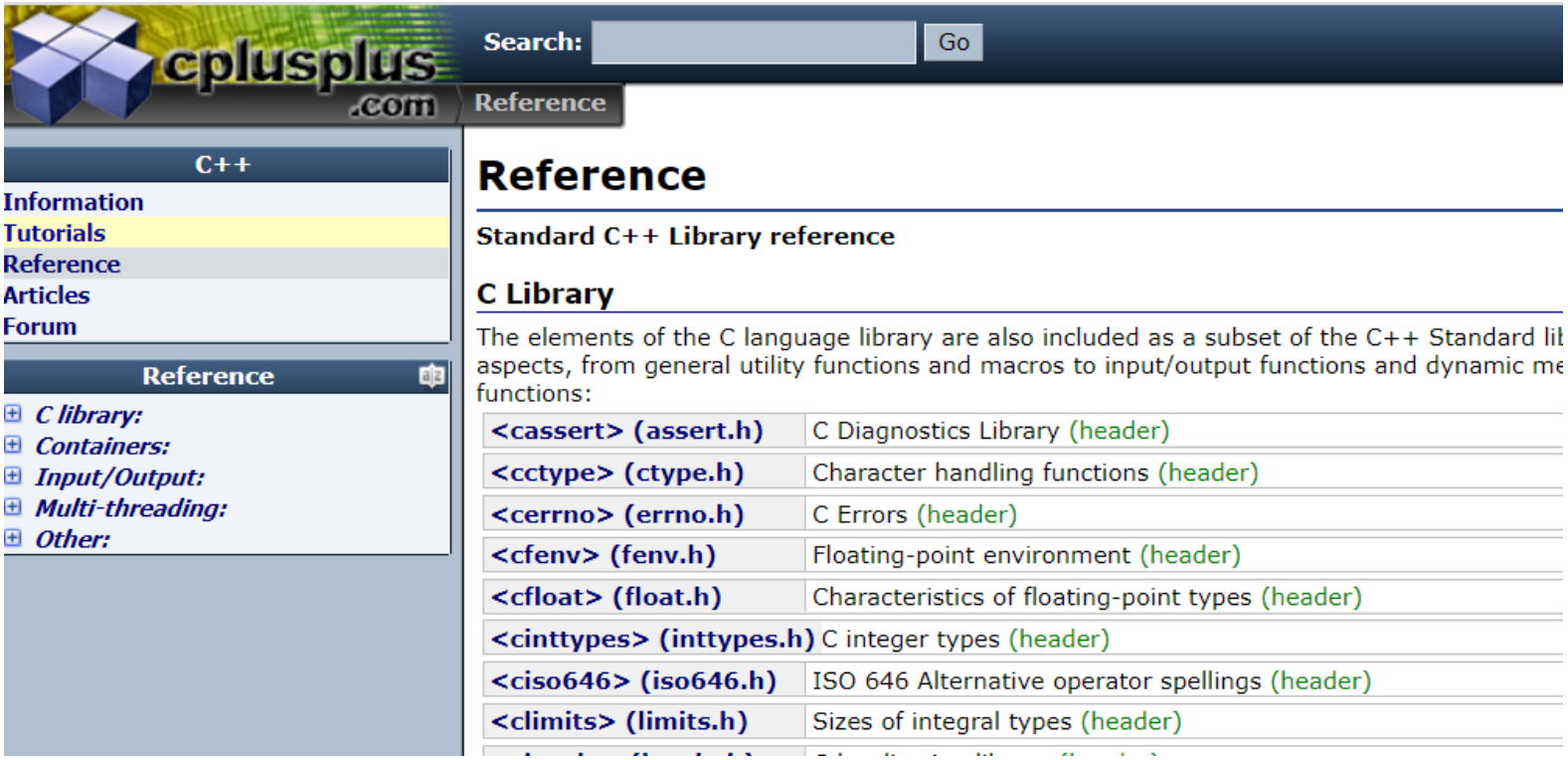
- -O0, -O1, -O2, -O3, -Os
  - Make your program more faster
  - g++ -std=c++11 -O3 main.cpp -o test.o

STL: Standard Template Library

- **Containers**: hold objects, all of a specified type
- **Iterators**: a generic pointer to access objects in containers
- **Algorithms**: act on objects in containers
- Save you a lot of efforts to use those common data structures or algorithms

STL: Standard Template Library

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



Container class templates

Sequence containers	Container adaptors	Associative containers	Unordered associative containers
array	stack	set	unordered_set
vector	queue	multiset	unordered_multiset
deque	priority_queue	map	unordered_map
forward_list		multimap	unordered_multimap
list			

Introduction to vector

- #include <vector>
- Vectors elements are indexed starting with 0
  - [ ]'s are used to read or change the value of an item
  - v[i]=42; cout<<v[i];
  - Elements are added to a vector using the member function push_back()
- Declaration
  - You can declare vector with any types
    - `std::vector<int> v;`
    - `std::vector<char> v;`
  - <type> identifies vector as a template class
  - Initialization
    - vector<int> v1(5,0); v1=[0,0,0,0,0]
    - vector<int> v2{5,0, 3,5,7}; v2=[5,0,3,5,7]
    - vector<int> v3 = v2; v3=[5,0,3,5,7]

Size vs. Capacity in vector

- Size is the number of elements that actually used
  - The range that [ ]'s can access the element
  - Function resize() can be used to shrink or expand a vector
- A vector's capacity is the number of elements allocated in memory
  - Accessible using the capacity( )
- When a vector runs out of space, the capacity is automatically increased
  - Function reserve( ) increases the capacity manually
- The two numbers are not the same. (capacity ≥ size)

Useful functions of vector

Function	Effect
vector.empty()	Tell whether vector is empty
vector.clear()	Clear all elements of v
vector.size()	Return the size of vector
vector.begin()	Return the address of first element
vector.end()	Return next address of last element
vector.front()	Return the value of first element
vector.back()	Return the value of last element
vector.push_back()	Add element at the end of vector
vector.pop_back()	Delete last element
vector.resize()	Resize the capacity of vector

more functions : http://www.cplusplus.com/reference/vector/vector/swap/

Example

```
vector<char> v2;
v2.push_back('A');
v2.push_back('B');
v2.push_back('C');
v2.push_back('D');
vector<char>::iterator p = v2.begin()
for (; p != v2.end(); p++) // A B C D
    cout << *p << endl;

cout << v2[2] << endl; //C
cout << p[2] << endl; //C
cout << *(p+2) << endl; //C
p = v2.begin( );
cout << *p << endl; //A
p++; cout << *p << endl; //B
p++; cout << *p << endl; //C
p--; cout << *p << endl; //B
```

```
A B C D
C
C
C
A
B
C
B
```

Add value to vector

- push_back (automatically extend capacity if it's not enough)

```
vector<int> v;
for(int i=0;i<5;i++)
{
    v.push_back(5-i);
}
```

after for loop, v contains: 5 4 3 2 1

- Directly assign (will not automatically extend capacity)

```
vector<int> v;
for(int i=0;i<5;i++)
{
    v[i] = i;
}
```

wrong! Initial size is 0,
need to resize capacity first.

```
vector<int> v;
v.resize(5);
for(int i=0;i<5;i++)
{
    v[i] = i;
}
```

after for loop, v contains: 0 1 2 3 4

Go through whole vector(1/2)

```
vector<int> v1;
for(int i=0;i<5;++i)
{
    v1.push_back(i);
}
for(unsigned int i=0;i<v1.size();++i)
{
    cout<<v1[i]<<" ";    0 1 2 3 4
}
```

Insert value to vector at specific position

```
vector<int> v1;
for(int i=0; i<5; i++) //0 1 2 3 4
    v1.push_back(i);

            Position
v1.insert(v1.begin(), 33); // "33" 0 1 2 3 4
v1.insert(v1.begin()+3, 8); // 33 0 1 "8" 2 3 4

for(int i=0; i<v1.size(); i++)
    cout << v1[i] << " ";
cout << endl << endl;
```

```
33 0 1 8 2 3 4
```

Delete value to vector at specific position

```
vector<int> v1 = {1, 2, 3, 4, 5};
v1.erase(v1.begin());// 2 3 4 5
v1.erase(v1.begin()+2);// 2 3 5
                Position

for(int i=0; i<v1.size(); i++)
    cout << v1[i] << " ";
cout << endl << endl;
```

```
2 3 5
```

Map

- Associative container.
- Map provides sorting and friendly storing data structure, and it features one-on-one mapping system.
- Mapping from one data item (a key) to another (a value).
- Ex: student's ID and student's name

Map

- Key value : used to sort and uniquely identify the elements; in other words, there is no two same key values.

- Mapped value : store the data associated with the key values.

Map

- #include <map>
- Announce
  - map< (key) , (value) > (name)
  - EX: `map<int,data*> table;`
- m.insert(pair< type , type >(key , mapping));
  - Same as m[key] = value ;
  - EX: `table.insert(pair<int,data*> (id,p_data) );`
- If key is number/string, it will be auto. sorting by key.

Map(4/4)

- iterator->first / (*iterator).first
  - Get key value
- iterator->second / (*iterator).second
  - Get mapping value
- m.find(key)
  - Return the iterator if found; otherwise, return m.end()
- m.clear()
  - Remove all elements
- m.erase(iterator), m.erase(key-value)
  - Remove the specified element
- m.erase(iterator , iterator)
  - Remove the range of elements

Map reference

- http://www.cplusplus.com/reference/map/map/

Element access:

operator[]	Access element (public member function)
at C++11	Access element (public member function)

Modifiers:

insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace C++11	Construct and insert element (public member function)
emplace_hint C++11	Construct and insert element with hint (public member function)

Iterators

- ‘Abstraction’ of iterators
  - Designed to hide details of implementation
  - Provide uniform interface across different container classes
- Using overloaded operators:
  - ++, --, ==, !=, *
  - If p is iterator variable, *p gives access to data pointed to by p

Iterator Classifications

- Forward iterators:
  - ++ works on iterator
- Bidirectional iterators:
  - Both ++ and -- work on iterator
- Random-access iterators:
  - ++, --, and random access all work with iterator
- These are ‘kinds’ of iterators, not types!

Cycling with Iterators

- Cycling ability using for loop:

```
for (p = c.begin() ; p! = c.end() ; p++)
      process *p       //*p is current data item
```

- Keep in mind
  - Each container type in STL has own iterator types
    - Even though they're all used similarly

Example of map Iterator

- Declare:

```
map<int,data*> cluster;
map<int,data*>::iterator iter;
```

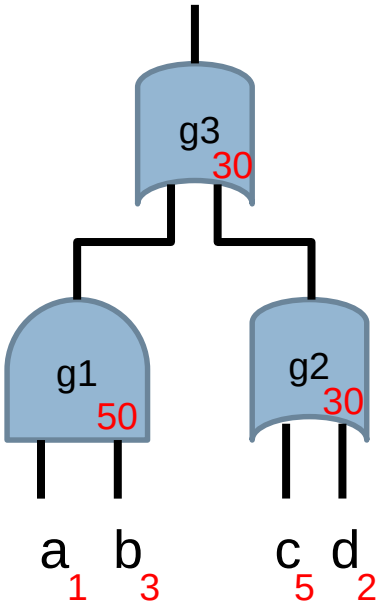
- traval:

```
for(iter=cluster.begin();iter!=cluster.end();iter++){
    cout<<*iter->first<<endl;
}
```

Lab Exercise

- Read a netlist by using argv
- Input
  - Lib count
    - Pin size
    - Gate size
  - Node count
    - All node name
  - Gate count
    - type name fanin1 fanin2

```
1  6
2  a 1
3  b 3
4  c 5
5  d 2
6  and 50
7  or 30
8  7
9  a b c d g3 g1 g2
10 3
11 or g3 g1 g2
12 or g1 a b
13 and g2 c d
```



- Output
  - Print out all nodes after sorting by name
  - Print out total fanin size of each gate

```
a b c d g1 g2 g3
g1:4
g2:7
g3:80
```

TODO Function

- Read the input netlist
  - Store each size into map<string, int>
  - Create node and store into the vector<NODE*>
    - Include the pin and the gate

```
void Read_Netlist(string dir, map<string, int> &libsize, vector<NODE*> &allnode){
    // TODO
}
```

- Calculate the fanin size of each gate then print out

```
void TotalFaninSize(map<string, int> &libsize, vector<NODE*> &allnode){
    // TODO
}
```

Compile & OJ Command

- Compile
  - g++ main.cpp -o Lab11
- OJ
  - /home/share/demo_OOP112_2 Lab 11

Submission

- You should exactly follow the output format
- Upload all your cpp to new E3
- Naming rule : studentID_LAB11.cpp
- Deadline is on E3
- Make sure your code can run on server