Cpp skills

struct v class

- The default access specifier for members of struct is public, whereas for member of class, it is private.
- Template type parameters can be declared with classes, but not with the struct keyword.

algorithms class

for_each

```
// for_each example
#include <iostream>     // std::cout
#include <algorithm>    // std::for_each
#include <vector>       // std::vector

void myfunction (int i) {  // function:
  std::cout << ' ' << i;
}

struct myclass {           // function object type:
  void operator() (int i) {std::cout << ' ' << i;}
} myobject;

int main () {
  std::vector<int> myvector;
  myvector.push_back(10);
  myvector.push_back(20);
  myvector.push_back(30);

  std::cout << "myvector contains:";
  for_each (myvector.begin(), myvector.end(), myfunction);
  std::cout << '\n';

  // or:
  std::cout << "myvector contains:";
  for_each (myvector.begin(), myvector.end(), myobject);
  std::cout << '\n';

  return 0;
}
```

find

```
// find example
#include <iostream>     // std::cout
#include <algorithm>    // std::find
#include <vector>       // std::vector

int main () {
  // using std::find with array and pointer:
  int myints[] = { 10, 20, 30, 40 };
  int * p;

  p = std::find (myints, myints+4, 30);
  if (p != myints+4)
    std::cout << "Element found in myints: " << *p << '\n';
  else
    std::cout << "Element not found in myints\n";

  // using std::find with vector and iterator:
  std::vector<int> myvector (myints,myints+4);
  std::vector<int>::iterator it;

  it = find (myvector.begin(), myvector.end(), 30);
  if (it != myvector.end())
    std::cout << "Element found in myvector: " << *it << '\n';
  else
    std::cout << "Element not found in myvector\n";

  return 0;
}
```

```
Element found in myints: 30
Element found in myvector: 30
```

binary_search

```
// binary_search example
#include <iostream>     // std::cout
#include <algorithm>    // std::binary_search, std::sort
#include <vector>       // std::vector

bool myfunction (int i,int j) { return (i<j); }

int main () {
  int myints[] = {1,2,3,4,5,4,3,2,1};
  std::vector<int> v(myints,myints+9);                         // 1 2 3 4 5 4 3 2 1

  // using default comparison:
```

```
  std::sort (v.begin(), v.end());

  std::cout << "looking for a 3... ";
  if (std::binary_search (v.begin(), v.end(), 3))
    std::cout << "found!\n"; else std::cout << "not found.\n";

  // using myfunction as comp:
  std::sort (v.begin(), v.end(), myfunction);

  std::cout << "looking for a 6... ";
  if (std::binary_search (v.begin(), v.end(), 6, myfunction))
    std::cout << "found!\n"; else std::cout << "not found.\n";

  return 0;
}
```

min_element

```
// min_element/max_element example
#include <iostream>     // std::cout
#include <algorithm>    // std::min_element, std::max_element

bool myfn(int i, int j) { return i<j; }

struct myclass {
  bool operator() (int i,int j) { return i<j; }
} myobj;

int main () {
  int myints[] = {3,7,2,5,6,4,9};

  // using default comparison:
  std::cout << "The smallest element is " << *std::min_element(myints,myints+7) << '\n';
  std::cout << "The largest element is "  << *std::max_element(myints,myints+7) << '\n';

  // using function myfn as comp:
  std::cout << "The smallest element is " << *std::min_element(myints,myints+7,myfn) <<
'\n';
  std::cout << "The largest element is "  << *std::max_element(myints,myints+7,myfn) <<
'\n';

  // using object myobj as comp:
  std::cout << "The smallest element is " << *std::min_element(myints,myints+7,myobj) <<
'\n';
  std::cout << "The largest element is "  << *std::max_element(myints,myints+7,myobj) <<
'\n';

  return 0;
}
```

begin(), end()

```cpp
// std::begin / std::end example
#include <iostream>     // std::cout
#include <vector>       // std::vector, std::begin, std::end

int main () {
  int foo[] = {10,20,30,40,50};
  std::vector<int> bar;

  // iterate foo: inserting into bar
  for (auto it = std::begin(foo); it!=std::end(foo); ++it)
    bar.push_back(*it);

  // iterate bar: print contents:
  std::cout << "bar contains:";
  for (auto it = std::begin(bar); it!=std::end(bar); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

reverse

```cpp
// reverse algorithm example
#include <iostream>     // std::cout
#include <algorithm>    // std::reverse
#include <vector>       // std::vector

int main () {
  std::vector<int> myvector;

  // set some values:
  for (int i=1; i<10; ++i) myvector.push_back(i);   // 1 2 3 4 5 6 7 8 9

  std::reverse(myvector.begin(),myvector.end());    // 9 8 7 6 5 4 3 2 1

  // print out content:
  std::cout << "myvector contains:";
  for (std::vector<int>::iterator it=myvector.begin(); it!=myvector.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

rotate

```
// rotate algorithm example
#include <iostream>     // std::cout
#include <algorithm>    // std::rotate
#include <vector>       // std::vector

int main () {
  std::vector<int> myvector;

  // set some values:
  for (int i=1; i<10; ++i) myvector.push_back(i); // 1 2 3 4 5 6 7 8 9

  std::rotate(myvector.begin(),myvector.begin()+3,myvector.end());
                                                  // 4 5 6 7 8 9 1 2 3
  // print out content:
  std::cout << "myvector contains:";
  for (std::vector<int>::iterator it=myvector.begin(); it!=myvector.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

all_of

- Returns `true` if pred returns `true` for all the elements in the range `[first,last)` or if the range is empty, and `false` otherwise.
- Parameters
  - first, last

    Input iterators to the initial and final positions in a sequence. The range used is `[first,last)`, which contains all the elements between first and last, including the element pointed by first but not the element pointed by last.
  - pred

    Unary function that accepts an element in the range as argument and returns a value convertible to `bool`. The value returned indicates whether the element fulfills the condition checked by this function. The function shall not modify its argument. This can either be a function pointer or a function object.

- ```
  #include <iostream>     // std::cout
  #include <algorithm>    // std::all_of
  #include <array>        // std::array

  int main () {
    std::array<int,8> foo = {3,5,7,11,13,17,19,23};

    if ( std::all_of(foo.begin(), foo.end(), [](int i){return i%2;}) )
      std::cout << "All the elements are odd numbers.\n";

    return 0;
  }
  ```
-

Vectors class

insert

insert a vector to a vector

- ```
  a.insert(a.end(), b.begin(), b.end());
  ```

insert at position

```
    // inserts 3 at front
    auto it = vec.insert(vec.begin(), 3);
```

pop_back()

- Only removes the last element
- Does not return it

back()

- Returns the last element

= operator

- ```
  std::vector<int> v1{1,2,3},v2;
  v2=v1;
  v1.push_back(4);
  v2.push_back(5);

  //v1:{1,2,3,4}; v2:{1,2,3,5};
  ```
  ```

```
std:: vector<int> *v1 = new std::vector<int>({1,2,3});
std:: vector<int> *v2;
v2=v1;
v1->push_back(4);
v2->push_back(5);

// *v1:{1,2,3,4,5}; *v2:{1,2,3,4,5};
```

initialize

```
/*1. Initializing by pushing values one by one :*/
// Create an empty vector
    vector<int> vect;

    vect.push_back(10);
    vect.push_back(20);
    vect.push_back(30);

    for (int x : vect)
        cout << x << " ";

// 10 20 30

/*2. Specifying size and initializing all values :*/

    int n = 3;

    // Create a vector of size n with
    // all values as 10.
    vector<int> vect(n, 10);

    for (int x : vect)
        cout << x << " ";

//10 10 10

/*3. Initializing like arrays :*/

// CPP program to initialize a vector like
// an array.
    vector<int> vect{ 10, 20, 30 };

    for (int x : vect)
        cout << x << " ";

//10 20 30

/*4. Initializing from an array :*/
// CPP program to initialize a vector from
// an array.
```

```
    int arr[] = { 10, 20, 30 };
    int n = sizeof(arr) / sizeof(arr[0]);

    vector<int> vect(arr, arr + n);

    for (int x : vect)
        cout << x << " ";

//10 20 30

/*5. Initializing from another vector :*/

// CPP program to initialize a vector from  another vector.
    vector<int> vect1{ 10, 20, 30 };

    vector<int> vect2(vect1.begin(), vect1.end());

    for (int x : vect2)
        cout << x << " ";

//10 20 30

/*6. Initializing all elements with a particular value : */
    vector<int> vect1(10); //reserve space for
    int value = 5;
    fill(vect1.begin(), vect1.end(), value);
    for (int x : vect1)
        cout << x << " ";

// 5 5 5 5 5 5 5 5 5 5
```

unordered_map

- Hash table
- Key value pair
- find an element by key
-

```
// CPP program to demonstrate implementation of
// find function in unordered_map.
#include <bits/stdc++.h>
using namespace std;

int main()
{
    unordered_map<int, bool> um;
```

```
    um[12] = true;
    um[6789] = false;
    um[456] = true;

    // Searching for element 23
    if (um.find(23) == um.end())
        cout << "Element Not Present\n";
    else
        cout << "Element Present\n";

    // Searching for element 12
    if (um.find(12) == um.end())
        cout << "Element Not Present\n";
    else
        cout << "Element Present\n";

    return 0;
}
```

-

map

- Binary Search Tree
- Each element has a key value and a mapped value.
- No two mapped values can have the same key values.
- In C++, maps store the key values in *ascending order* by default.

method

- begin() – Returns an iterator to the first element in the map.
- end() – Returns an iterator to the theoretical element that follows the last element in the map.
- size() – Returns the number of elements in the map.
- max_size() – Returns the maximum number of elements that the map can hold.
- empty() – Returns whether the map is empty.
- pair insert(keyvalue, mapvalue) – Adds a new element to the map.
- erase(iterator position) – Removes the element at the position pointed by the iterator.
- erase(const g)– Removes the key-value 'g' from the map.
- clear() – Removes all the elements from the map.

- `begin()`: Returns an iterator to the first element in the map.
- `size()`: Returns the number of elements in the map.
- `empty()`: Returns a boolean value indicating whether the map is empty.
- `insert( pair(key, value))`: Adds a new key-value pair to the map. An alternate way to insert values in the map is:

```
map_name[key] = value;
```

- `find(val)`: Gives the iterator to the element *val*, if it is found otherwise it returns m.end()
- `erase(iterator position)`: Removes the element at the position pointed by the iterator.
- `erase(const g)`: Removes the key value *g* from the map.
- `clear()`: Removes all the elements from the map.

Example

```
#include <string.h>
#include <iostream>
#include <map>
#include <utility>
using namespace std;

int main()
{
  // Initializing a map with integer keys
  // and corresponding string values
  map<int, string> Employees;

  //Inserting values in map using insert function
  Employees.insert ( std::pair<int, string>(101,"Jon") );
  Employees.insert ( std::pair<int, string>(103,"Daenerys") );
  Employees.insert ( std::pair<int, string>(104,"Arya") );

  // Inserting values using Array index notation
  Employees[105] = "Sansa";
  Employees[102] = "Tyrion";

  cout << "Size of the map is " << Employees.size() << endl << endl;

  // Printing values in the map
  cout << endl << "Default Order of value in map:" << endl;
  for( map<int,string>::iterator iter=Employees.begin(); iter!=Employees.end(); ++iter)
  {
    cout << (*iter).first << ": " << (*iter).second << endl;
  }

  // Finding the value corresponding to the key '102'
  std::map<int, string>::iterator it = Employees.find(102);
  if (it != Employees.end()){
    std::cout <<endl<< "Value of key = 102 => " << Employees.find(102)->second << '\n';
  }
}
```

find

```
std::map<char,int> mymap;
  std::map<char,int>::iterator it;

  mymap['a']=50;
  mymap['b']=100;
  mymap['c']=150;
  mymap['d']=200;


  it = mymap.find('b');
  if (it != mymap.end())
    mymap.erase (it);
```

queue

Methods

queue::empty()	**Returns whether the queue is empty.**
queue::size()	Returns the size of the queue.
queue::swap()	Exchange the contents of two queues but the queues must be of the same type, although sizes may differ.
queue::emplace()	Insert a new element into the queue container, the new element is added to the end of the queue.
queue::front()	Returns a reference to the first element of the queue.
queue::back()	Returns a reference to the last element of the queue.
queue::push(g)	Adds the element 'g' at the end of the queue.
queue::pop()	Deletes the first element of the queue.

unordered_set

- An **unordered_set** is implemented using a hash table where keys are hashed into indices of a hash table so that the insertion is always randomized.
- All operations on the **unordered_set** takes constant time **O(1)** on an average which can go up to linear time **O(n)** in worst case
- Set is an ordered sequence of unique keys whereas unordered_set is a set in which key can be stored in any order, so unordered

- For unordered_set many functions are defined among which most used are the
    - size and empty for capacity,
    - find for searching a key,
    - insert and erase for modification.
- The Unordered_set allows only unique keys, for duplicate keys unordered_multiset should be used.

```cpp
// C++ program to demonstrate various function of unordered_set
#include <bits/stdc++.h>
using namespace std;

int main()
{
    // declaring set for storing string data-type
    unordered_set <string> stringSet ;

    // inserting various string, same string will be stored
    // once in set

    stringSet.insert("code") ;
    stringSet.insert("in") ;
    stringSet.insert("c++") ;
    stringSet.insert("is") ;
    stringSet.insert("fast") ;

    string key = "slow" ;

    // find returns end iterator if key is not found,
    // else it returns iterator to that key

    if (stringSet.find(key) == stringSet.end())
        cout << key << " not found" << endl << endl ;
    else
        cout << "Found " << key << endl << endl ;

    key = "c++";
    if (stringSet.find(key) == stringSet.end())
        cout << key << " not found\n" ;
    else
        cout << "Found " << key << endl ;

    // now iterating over whole set and printing its
    // content
    cout << "\nAll elements : ";
    unordered_set<string> :: iterator itr;
    for (itr = stringSet.begin(); itr != stringSet.end(); itr++)
        cout << (*itr) << endl;
}
```

```
slow not found

Found c++

All elements :
is
fast
c++
in
code
```

stack

Methods all O(1)

- empty() – Returns whether the stack is empty – Time Complexity : O(1)
- size() – Returns the size of the stack – Time Complexity : O(1)
- top() – Returns a reference to the top most element of the stack – Time Complexity : O(1)
- push(g) – Adds the element 'g' at the top of the stack – Time Complexity : O(1)
- pop() – Deletes the top most element of the stack – Time Complexity : O(1)

- pop() only deletes does not return

find ceiling in division

```
unsigned int x, y, q;

q = 1 + ((x - 1) / y); // if x != 0
```

Limits on Integer Constants

Constant	Meaning	Value
CHAR_BIT	Number of bits in the smallest variable that is not a bit field.	8
SCHAR_MIN	Minimum value for a variable of type `signed char`.	-128
SCHAR_MAX	Maximum value for a variable of type `signed char`.	127
UCHAR_MAX	Maximum value for a variable of type `unsigned char`.	255 (0xff)
CHAR_MIN	Minimum value for a variable of type `char`.	-128; 0 if /J option used
CHAR_MAX	Maximum value for a variable of type `char`.	127; 255 if /J option used
MB_LEN_MAX	Maximum number of bytes in a multicharacter constant.	5
SHRT_MIN	Minimum value for a variable of type `short`.	-32768
SHRT_MAX	Maximum value for a variable of type `short`.	32767
USHRT_MAX	Maximum value for a variable of type `unsigned short`.	65535 (0xffff)
INT_MIN	Minimum value for a variable of type `int`.	-2147483647 - 1
INT_MAX	Maximum value for a variable of type `int`.	2147483647
UINT_MAX	Maximum value for a variable of type `unsigned int`.	4294967295 (0xffffffff)
LONG_MIN	Minimum value for a variable of type `long`.	-2147483647 - 1
LONG_MAX	Maximum value for a variable of type `long`.	2147483647
ULONG_MAX	Maximum value for a variable of type `unsigned long`.	4294967295 (0xffffffff)

Constant	Meaning	Value
LLONG_MIN	Minimum value for a variable of type `long long`.	-9,223,372,036,854,775,807 - 1
LLONG_MAX	Maximum value for a variable of type `long long`.	9,223,372,036,854,775,807
ULLONG_MAX	Maximum value for a variable of type `unsigned long long`.	18,446,744,073,709,551,615 (0xffffffffffffffff)

polymorphism

- Function overloading and operator overloading is a type of Compile time polymorphism. We call this type of polymorphism as early binding or Static binding.
- ```
  #include <iostream>
  using namespace std;
  class Addition {
  public:
      int ADD(int X,int Y)   // Function with parameter
      {
          return X+Y;     // this function is performing addition of  two Integer value
      }
      int ADD() {              // Function with same name but without parameter
          string a= "HELLO";
          string b="SAM";   // in this function concatenation is performed
         string c= a+b;
         cout<<c<<endl;

      }
  };
  int main(void) {
      Addition obj;   // Object is created
      cout<<obj.ADD(128, 15)<<endl; //first method is called
      obj.ADD();  // second method is called
      return 0;
  }
  ```
- Function overriding is a part of runtime polymorphism. In function overriding, more than one method has the same name with different types of the parameter list.
- ```
  #include <iostream>
  using namespace std;
  class Animal {
      public:
  void function(){
  cout<<"Eating..."<<endl;
      }
  };
  ```

```
class Man: public Animal
{
 public:
 void function()
    {
       cout<<"Walking ..."<<endl;
    }
};
int main(void) {
 Animal A =Animal();
   A.function();   //parent class object
   Man m = Man();
   m.function();   // child class object

   return 0;
}
```

-

string

Find index

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

int main()
{
    std::string s = "C++20";
    char c = '+';

    int index = s.find(c);

    if (index != std::string::npos) {
        std::cout << "Character found at index " << index << std::endl;
    } else {
        std::cout << "Character not found" << std::endl;
    }

    return 0;
}
```

push_back

- Only adds a character to the back

append

- adds string to the back
- ```
  string& string::append (const string& str)
  ```
- Appends at most, str_num characters of string str, starting with index str_idx
- ```
  string& string::append (const string& str, size_type str_idx, size_type str_num)
  ```
- Appends num occurrences of character c
- ```
  string& string::append (size_type num, char c)
  ```
-

insert

- **insert()** is used to insert characters in string at specified position.
- ```
  string& string::insert (size_type idx, const string& str)

  s.insert(5,"hello");  // strinhellog
  ```
- Inserts at most, str_num characters of str, starting with index str_idx.
- ```
  string& string::insert (size_type idx, const string& str, size_type str_idx, size_type
  str_num)
  ```
- Inserts num occurrences of character c at the position specified by idx.
- ```
  // Only character not string

  string& string ::insert (size_type idx, size_type num, char c)

        s.insert(5,5,'h'); //strinhhhhhg
  str.insert(str.begin() + 5, 5, '$');
  ```
- Inserts all characters of the range [beg,end) before the character to which iterator pos refers.
- ```
  void string ::insert (iterator pos, InputIterator beg, InputIterator end )
  ```

pair

```
// pair::pair example
#include <utility>      // std::pair, std::make_pair
#include <string>       // std::string
#include <iostream>     // std::cout
```
```

```
int main () {
  std::pair <std::string,double> product1;                     // default constructor
  std::pair <std::string,double> product2 ("tomatoes",2.30);   // value init
  std::pair <std::string,double> product3 (product2);          // copy constructor

  product1 = std::make_pair(std::string("lightbulbs"),0.99);   // using make_pair (move)

  product2.first = "shoes";                  // the type of first is string
  product2.second = 39.90;                   // the type of second is double

  std::cout << "The price of " << product1.first << " is $" << product1.second << '\n';
  std::cout << "The price of " << product2.first << " is $" << product2.second << '\n';
  std::cout << "The price of " << product3.first << " is $" << product3.second << '\n';
  return 0;
}
```

sstream

matrix

breadth first traversal matrix

```
void bfsMatrix(vector<vector<int>> mat){
  vector<vector<bool>> visited(mat.size(), vector<bool>(mat[0].size(), false));

  queue<pair<int,int>> q;

  q.push(make_pair(0,0));

  while(!q.empty()){
    pair<int, int> a = q.front(); q.pop();

    if(a.first < 0 || a.first >= mat.size() || a.second < 0 || a.second >=
mat[a.first].size()){
      continue;
    }

    if(visited[a.first][a.second]) {
      continue;
    }

    cout << a.first << " " << a.second << ": "<< mat[a.first][a.second] << endl;
```

```
    visited[a.first][a.second] = true;

    q.push(make_pair(a.first, a.second-1));  // left
    q.push(make_pair(a.first, a.second+1));  // right
    q.push(make_pair(a.first - 1, a.second));  // top
    q.push(make_pair(a.first + 1, a.second));  // bottom

  }

}
```

depth first traversal

```
void dfsMatrix(vector<vector<int>> mat){

  vector<vector<bool>> visited(mat.size(), vector<bool>(mat[0].size(), false));

  stack<pair<int,int>> s;

  s.push(make_pair(0,0));

  while(!s.empty()){
    pair<int, int> a = s.top(); s.pop();


    if(a.first < 0 || a.first >= mat.size() || a.second < 0 || a.second >=
mat[a.first].size()){
      continue;
    }

    if(visited[a.first][a.second]) {
      continue;
    }

    cout << a.first << " " << a.second << ": "<< mat[a.first][a.second] << endl;
    visited[a.first][a.second] = true;

    s.push(make_pair(a.first, a.second-1));  // left
    s.push(make_pair(a.first, a.second+1));  // right
    s.push(make_pair(a.first - 1, a.second));  // top
    s.push(make_pair(a.first + 1, a.second));  // bottom

  }

}
```

- The only difference is using a stack or using a queue
-

Operator overloading

```
      // overloaded prefix ++ operator
      Time operator++ () {
         ++minutes;          // increment this object
         if(minutes >= 60) {
            ++hours;
            minutes -= 60;
         }
         return Time(hours, minutes);
      }

      // overloaded postfix ++ operator
      Time operator++( int ) {

         // save the orignal value
         Time T(hours, minutes);

         // increment this object
         ++minutes;

         if(minutes >= 60) {
            ++hours;
            minutes -= 60;
         }

         // Overload + operator to add two Box objects.
Box operator+(const Box& b) {
   Box box;
   box.length = this->length + b.length;
   box.breadth = this->breadth + b.breadth;
   box.height = this->height + b.height;
   return box;
}

         // overloaded minus (-) operator
      Distance operator- () {
         feet = -feet;
         inches = -inches;
         return Distance(feet, inches);
      }

      void operator = (const Distance &D ) {
         feet = D.feet;
         inches = D.inches;
      }
```

```
      int &operator[](int i) {
         if( i > SIZE ) {
            cout << "Index out of bounds" <<endl;
            // return first element.
            return arr[0];
         }

         return arr[i];
      }
```

chaining function

```
class foo
{
  private:
    int x;
    int y;
  public:
    foo& setx(int x_)
    {   x = x_;
        return *this; }
    foo& sety(int y_)
    {   y = y_;
        return *this; }
    foo& print()
    {std::cout << x << ' ' << y;
     return *this;}
};

int main()
{
    foo bar;
    bar.setx(1).sety(2).print();
}
```