C++

Standard

C++ 14

lldb partial support ‘structured bindings’ which comes in C++17.

Env

// editor

vim, vscode, zed

https://godbolt.org/

cling REPL

online REPL: https://cpp.sh/

// compiler

```
clang++ -E -x c++ - -v < /dev/null
// to show default include files
```

Policy-Based Data Structures (PBDS) in g++ (not standard)

clang++ work with GNU’s libstdc++:

clang++ -stdlib=libstdc++ t.cpp

If libstdc++ isn’t in a standard location, need use flags like -I for includes and -L for libraries:

clang++ -I/path/to/libstdc++/includes -L/path/to/libstdc++/libs t.cpp/

// debugger

lldb (can read g++ build executable)

gdb (still not for M1)

main difference with Java

object var store value, not a reference. “a=b”, “a” is a copy.

string is mutable

```
int arr[3][3]; // 2nd [] must have number
```

Input/Output

```
#include <iostream>
string a;
cin>>a;
getline(cin, a);

cout<<a<<endl;

#include <iomanip>
cout << fixed << setprecision(2) << number;
// or
#include <cstdio>
printf("%.2f\n", a);

// ensure output appears immediately
cout << "Start of program" << flush;
// or use std::endl, \n won’t flush
```

Variable

```
std::swap(i,j) //for vector too
```

5 types:

Fundamental types: (primitives)

bool, char, short, int, long, float, double

long double

```
#include <cstdint>
int32_t, uint32_t etc
```

Type range:

C header: <climits>

INT_MIN

INT_MAX

C++ header: <limits>

```
numeric_limits<int>::min();
numeric_limits<int>::max();
```

Infinity:

```
// for float
<cmath>
float inf = INFINITY;

// for double
<limits>
numeric_limits<float>::infinity();
```

Compound types:

array, struct, class, enum

STL containers:

string (like), vector, set, map etc

Pointer type, Dynamic memory type:

allocated by new/delete

Object type:

Function object (functor) and Lambdas

Copy or not

```
vector<int> a={1,2,3};
vector<int> b={2,4,5};
// copy
a=b;
// not copy, but reference
vector<int> &c=b;

// transfers ownership
a=std::move(b)
```

Convert int and char

```
// 2 -> ‘2’
int num = 2;
char digit = '0' + num;
// ‘2’ -> 2
int num = digit - ‘0';

int n=80;
char c=n;
// or compile time
char c=static_cast<char>(n);
```

Pointer

```
int a=100;
int* i=&a;

value: a
address: &a
pointer: i
the value it points to: *i

// print var's address
cout<<&a<<endl;
// print the pointer
cout<<i<<endl;
```

Function

```
int sum(vector<int>& nums) {…}
```

Functions cannot be defined inside main() or other functions. but lambda and functor can.

Lambda

```
[capture list](parameter list)
mutable -> return type {
  ……
}
// [n=0] init-capture
// without mutable, cannot change n
[n = 0]() mutable {
  return n++;
}

//
auto add = [](int a, int b) ->int{
  return a + b;
};
// or without "-> int"
auto add = [](int a, int b) {
  return a + b;
};

// or capture local variable 'g'
auto add = [&g](int a, int b) {
    g[a].insert(b);
    g[b].insert(a);
};

// or capture all local variables
// [&] capture as reference
auto add = [&](int a, int b) {
    g[a].insert(b);
    g[b].insert(a);
};
// [=] capture as const copy
auto add = [=](int a, int b) {
    g[a].insert(b);
    g[b].insert(a);
};
```

functional object or functor

```
class Adder {
  public:
    int operator() (int a, int b)
    {return a + b;}
};

Adder add; // same as: Adder add()
int result = add(5, 3);
```

Macro

```
#define A 100
```

typedef

```
// c++ 11
using point=pair<int,int>;

// traditional way
typedef pair<int, int> point;
typedef uint32_t u32;
```

operators

```
xor: 1^2
```

math functions

```
#include <cmath>
abs, pow, sqrt, cbrt, round
#include <algorithm>
max, min
```

struct

The syntax for a struct and a class is nearly identical, and both can have data members, functions, constructors, destructors, and even inherit from other classes or structs

```
struct Point {
  // members are public by default
  int x, y;
  // use 'member initializer list'
  Point(int x, int y) : x(x), y(y) {}
  int getX() { return x; }
};

Point p(100,200);
```

override operator and output

```
struct MyInt {
  int value;
  MyInt(int v) : value(v) {}

  MyInt operator+(const MyInt& other)
const {
    string result = to_string(this-
>value) + to_string(other.value);
    int resultValue = stoi(result);
    return MyInt(resultValue);
  }

  friend ostream& operator<<(ostream&
os, const MyInt& obj) {
  os << obj.value;
  return os;
  }
};

MyInt a(1);
MyInt b(2);
MyInt c = a + b;
```

class (simple)

```
class Point {
private:
    int x, y; // Private by default
public:
  Point(int x, int y): x(x), y(y) {}
  int getX() { return x; }
};

// create an object
Point p(100,200);

//
map<int,Point> m;
m[key]=Point(100,200);
m[key].x;
//
map<int,Point*> m;
m[key]=new Point(100,200);
m[key]->x;
delete m[key];
```

template (at least able to read)

String (mutable, like sequence container, but it is not)

```
string literal vs a string object:
“abc” is an array ‘const char[4]’,
not an object.
“abc”.length() // not work

string s=“abc”;
s.length() // works, =s.size()

// Create
//Copy Initialization
string s=“100 200”;
string s; // same as string s = “"
//Direct Initialization (constructor)
string s(“100 200”);
string s(10, ‘a’); // 10 ‘a’ chars

// Read
s[1] // = s.at(1)
s.back() // = s.at(s.size() - 1)
s.substr(start, length)
s.substr(start)

// Loop
for (auto& c:s) {}
for (size_t i=0; i<s.size(); i++){}
for (auto it=s.begin(); it!=s.end();
it++) { cout<< *it <<endl; }

// Update
s[0]=“A”

// Update (add)
s+=“abc” //same as s.append(“abc”)
s.insert(1,”abc”)
// add a char
s.push_back(aChar)
// add multiple char
s.append(10,’a’) // add 10 ‘a’
s.insert(1,4,’2’) // insert 4 chars

// Update (replace a string)
string s="abcd abcd";
string old="ab";
auto p=s.find(“ab"); // size_t
// npos: the maximum value of size_t
(usually -1)
while (p!=string::npos) {
s.replace(p,old.length(),"AB");
    p=s.find(old, p);
    cout<<s<<endl;
}

// Update (replace a char)
#include <algorithm>
replace(s.begin(),s.end(),'A','z');

// delete
s.erase(1,3) // delete 3 chars
s.clear()

// delete all occurrences of ‘a’
string s = "ababa";
auto pos = remove(s.begin(), s.end(),
'a');
s.erase(pos, s.end());
// or
size_t pos;
while ((pos = s.find('a')) !
=string::npos) {
    s.erase(pos, 1);
}

// iterator
s.rbegin(), s.rend();
s.begin(), s.end();

// query
s.starts_with(“abc”) // C++20
s.ends_with(“abc”)

// find a string
size_t p=s.find(“ab")
size_t p=s.find(“ab”,startPos);

// find a char
size_t p=s.find(‘b’)

in <algorithm>, but slow:
find(s.begin(), s.end(), ‘b’)-
s.begin()

// count
count(s.begin(), s.end(),’a’); (only
count char)

// count a string
string s="abcd abcd";
string a="ab";
int p=s.find(a);
int count=0;
while (p!=string::pos) {
  count++;
```

```
  p=s.find(a,p+a.length());
}

// is empty?
s==“” // same as s.empty()

// Copy
s=“abc”
s1=s;

// Reverse (directly)
#include <algorithm>
reverse(s.begin(),s.end())

// Reverse (a copy)
string s = string(s.rbegin(),
s.rend());

// Sort
sort(s.begin(), s.end());

// split by space
string s = "100 200";
// stringstream ss(s);
istringstream iss(s);
string part;
while (iss >> part)
{
    cout << part << endl;
}

// split by +
string s = "100+200+300";
istringstream iss(s);
string part;
while (getline(iss, part, ‘+’)) {
   if (!part.empty()) {
     cout << part << endl;
   } else {
     cout << "[Empty part]" <<endl;
   }
}
```

```
// split by ++ manually or Boost
string s = "100++200++300++400";
string delimiter = "++";
size_t pos = 0;
size_t start = 0;
string token;

while ((pos = s.find(delimiter,
start)) != string::npos)
{
   token = s.substr(start, pos-
start);
   cout << token << endl;
   start = pos + delimiter.size();
}

// convert number <-> string
string numStr = to_string(42);
int num = stoi(numStr);
float num = stof(numStr);

// check type
cout << isalnum('z') << endl;
cout << isalpha('z') << endl;
cout << isdigit('1') << endl; // not
for floating num or num with signs
cout << isspace(' ') << endl;

// format
// add leading zeros
#include <iomanip>
s = "ab";
size_t width = 10;
cout << setfill('0') << setw(width)
<< s << endl;
// or
string(width - s.size(), '0') + s
// or
int num = width - (min(s.length(),
width));
s.insert(0, num, '0');
// or
// #include <format> // C++20
// cout << format("{:010}", s) <<
endl;

// case
// to upper
s = "abcdefg";
for (char& c : s)
{
    c = toupper(c); // tolower
}
cout << s << endl;

// char -> string
char c = 'a';
string s1(1, c);
// or if s1 exists
s1 += c;

// compare
cout << (a == b) << "\n";
cout << (a > b) << "\n";
cout << (a < b) << "\n";
```

Array

Array:

```
// list initialization
int a[] {1,2}
int a[]={1,2}
int a[3]; // [193599, 494625,0]
int a[3]={8}; // [8,0,0]
fill(a,a+10,100); // fill with 100

//2d
int a[4][4];
// the second dimension must be
specified
int a[][3]={{1,2,3},{3,4,5},{4,5,6}}

// note: VLAs are not standard in C++
int r,c; cin>>r>>c;
int a[r][c]; // this is Variable-
Length Array, so should use vector
```

```
vector<vector<int>> a(r,
vector<int>(c));
// #include <array> not support VLA
too

// print arr
for (int& e:a) {}

for (int i=0; i<sizeof(a)/
sizeof(a[0]); i++) {}

// arr is like a pointer
void p_arr(int arr[], int size) {
}

// sort
#include <algorithm>
#include <iterator> // for
std::begin(), std::end()
#include <functional>

sort(begin(arr),
end(arr),greater<int>());
// or
bool compare(int a, int b) { return a
> b; }
sort(v.begin(), v.end(), compare);

// copy
int a[]={1,2,3,4};
int b[4];
copy_n(a,3,b); //copy the first 3
elements from a into b
```

Random

```
#include <cstdlib>
srand(time(0));
cout << RAND_MAX << endl;
cout << rand() << endl;
// [0,RAND_MAX)
cout << (double)rand() / RAND_MAX <<
endl; // [0,1)
```

C++ containers

cheatsheet_cpp_containers.key

Regular Expression C++11

structured bindings C++17 (lldb not support)

```
Works for tuple-like types which have
certain properties (tuple_size,
tuple_element, get):
pair, tuple; struct, class; array
container, and array, but not string

// for unpack array, even it has no
those properties, but C++17 supports
int arr[3] = {1, 2, 3};
auto [x, y, z] = arr;

// for unpack tuple
auto rc = make_tuple(1, 2);
auto [r, c] = rc;

// for unpack struct
struct Position {
  char row;
  char col;
};

Position pos = {'A', '1'};
auto [r, c] = pos;
```

variadic templates (with Fold expressions)

```
template<typename... Args>
auto sum(Args... args) {
  return (args + ...); // Unary right
fold
}

template<typename... Args>
bool allTrue(Args... args) {
  return (true && ... && args); //
Binary right fold with initial value
}
```

Init

Direct Initialization: `int x(5);`
Copy Initialization: `int x = 5;`
Brace Initialization: C++11

```
int x{};
int x{5};
vector<int> v{1, 2, 3};
```

Aggregate Initialization: `int arr[3] = {1, 2, 3};`

goto

```
goto label;

label:
// code to execute
```

Other

```
system("ls") //to execute OS commands

char c;
c = cin.get(); // Read 1 character
including whitespace characters like
spaces and newlines

cin.get(); // pause until Enter key

TeensProgramming.com
```