Introduction to C++

Tony Wong

2015-01-24

Why C++?

- It is modern
- It is still being used
- Share *similar* syntax with the top 5 programming languages, which are
  - C, Java, PHP, Javascript
  - C++ ranks 5th
  - Pascal ranks 23rd
- Standard Template Library that is useful in competitive programming

Download Dev-C++

- Orwell Dev-C++ 5.8.3 with TDM-GCC 4.8.1
- http://sourceforge.net/projects/orwelldevcpp/files/latest/download

First and foremost

- C++ is case sensitive.
  Variables x and X are different.
- Every statement requires ;
- := becomes =
- = becomes ==

Basic Program Structure

- The starting point (i.e. begin end. In Pascal) of a C++ program is the **main()** function
- The **main()** function returns an integer, which is the exit code of the program
- Therefore we return 0; when the program ends normally.

```
1   #include <cstdio>
2   int main() {
3
4     return 0;
5   }
```

Variables

- Variables can be declared anywhere in the program

Pascal	C++	Size (normally)
`longint`	`int`	4 bytes
`int64`	`long long`	8 bytes
`char`	`char`	1 byte
`boolean`	`bool`	1 byte
`single`	`float`	4 bytes
`double`	`double`	8 bytes
`real / extended`	No equivalent types	

Comments

- `//` for inline comments
  (which also works in Pascal)
- `/*`
  `*/` for multiline comments

```
1   #include <cstdio>
2   int main() {
3     // single line comment;
4     int a = 2;
5     /* multi
6        line
7        comment */
8     return 0;
9   }
```

Input and Output

- There are two flavors of Input / Ouput
- I will be using C flavor

	C flavor	C++ flavor
Library	`#include <cstdio>`	`#include <iostream>`
Input	`scanf("%d", &n);`	`cin >> n;`
Output	`printf("%d\n", n);`	`cout << ans << endl;`

Input: scanf Output: printf

- Syntax:

scanf(*format string*, *address of var1*, *address of var2*, ...);

- Example:

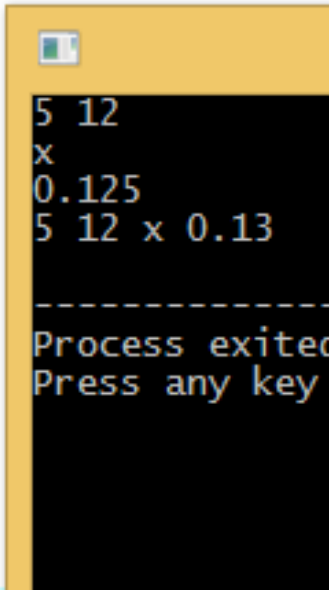
```
1  #include <cstdio>
2  int main() {
3    int a, b;
4    scanf("%d %d", &a, &b);
5    printf("%d\n", a + b);
6    return 0;
7  }
```

- **%d** in the format string is called token, it represents the expected variable type

Input: scanf Output: printf

- Token of the most common types
- long long (64-bit integer): %lld

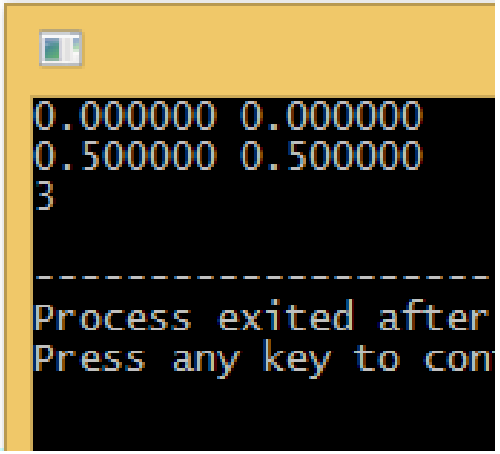
```
1   #include <cstdio>
2   int main() {
3     int a, b;
4     char c;
5     double d;
6     scanf("%d %d", &a, &b);
7     // note the space before %c
8     scanf(" %c", &c);
9     scanf("%lf", &d);
10    printf("%d %d %c %.2lf\n", a, b, c, d);
11    return 0;
12  }
```



Assignment and division

- If and only if any of the two operands is a real number, the result is also a real number.

```
1  #include <cstdio>
2  int main() {
3    int a = 1;
4    double b = 1;
5    printf("%lf %lf\n", a/2, b/2);
6    printf("%lf %lf\n", 1.0/2, 1/2.0);
7    printf("%d\n", 15 % 4);
8    return 0;
9  }
```



Assignment

`inc(x);`	`x++; or ++x;`
`dec(x);`	`x--; or --x;`
`x = x + y;`	`x += y;`
`x = x - y;`	`x -= y;`
`x = x * y;`	`x *= y;`
`x = x div 10;`	`x /= 10;`
Bitwise (also works if a and b are booleans)	
`a = a or b;`	`a |= b;`
`a = a and b;`	`a &= b;`
`a = a xor b;`	`a ^= b;`

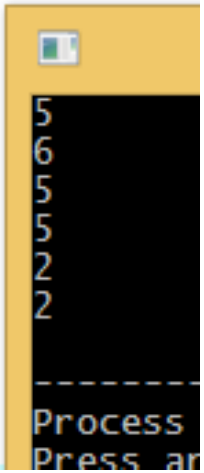
Bitwise:

- shl: <<
- shr: >>
- not: ~

Difference between x++ and ++x

- **x++** returns the original value of x, and then increase x by 1
- **++x** increases x by 1 first, and returns reference of x
- **++x** can be chained, **x++** cannot

```
1   #include <cstdio>
2   int main() {
3     int x = 5;
4     printf("%d\n", x++);
5     printf("%d\n", x);
6     printf("%d\n", --x);
7     printf("%d\n", x);
8     printf("%d\n", ------++--x);
9     printf("%d\n", x);
10    return 0;
11  }
```

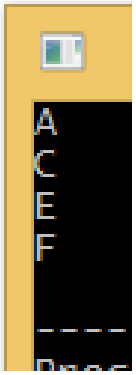


If and boolean expressions

```
if (x == y && y == z) {
  printf("Equilateral triangle\n");
} else if (x == y || y == z || !(x != z)) {
  printf("Isoceles triangle\n");
} else printf("Scalene triangle\n");
```

- Non zero values are treated as true

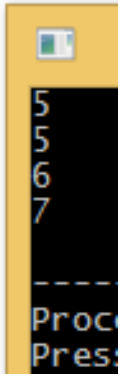
```
if (-1) printf("A\n");
if (0) printf("B\n");
if (1) printf("C\n");
if (0.0) printf("D\n");
if (0.01) printf("E\n");
if ('x') printf("F\n");
```



Shortcut evaluation

- Boolean sub-expressions separated by && and || are evaluated left-to-right
- If the left expression of the && operator is false, the right expression would not be evaluated.
- Similarly, if the … || operator is true ……

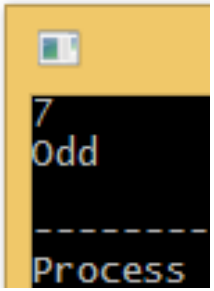
```
int a = 5, b;
if (true || a++) b = 0;
printf("%d\n", a);
if (false && a++) b = 0;
printf("%d\n", a);
if (a++ || true) b = 0;
printf("%d\n", a);
if (a++ && false) b = 0;
printf("%d\n", a);
```



? : operator

- Syntax:
  condition ? expr1 : expr2;
- If condition is true, returns expr1
- If condition is false, returns expr2

```
1   #include <cstdio>
2   int main() {
3     int n;
4     scanf("%d", &n);
5     printf(n % 2 ? "Odd\n" : "Even\n");
6     return 0;
7   }
```



Switch-case

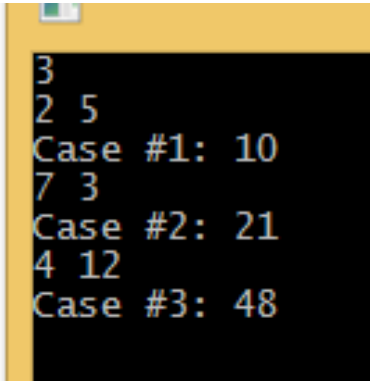
- The switch-case control structure is very different from Pascal's case-of
- Statements below the case will also get executed, if not **break**ed
- **`else`** becomes **`default:`**

```
1  #include <cstdio>
2  int main() {
3    char c;
4    scanf("%c", &c);
5    switch (c) {
6      case 'a':
7      case 'e':
8      case 'i':
9      case 'o':
10     case 'u':
11       printf("Vowel\n");
12       break;
13     default:
14       printf("Consonant\n");
15   }
16   return 0;
17 }
```

For-loop

- Syntax:
  `for (initial; condition; increment)`

```
int n;
scanf("%d\n", &n);
for (int i = 1; i <= n; i++) {
  int a, b;
  scanf("%d %d", &a, &b);
  printf("Case #%d: %d\n", i, a * b);
}
```



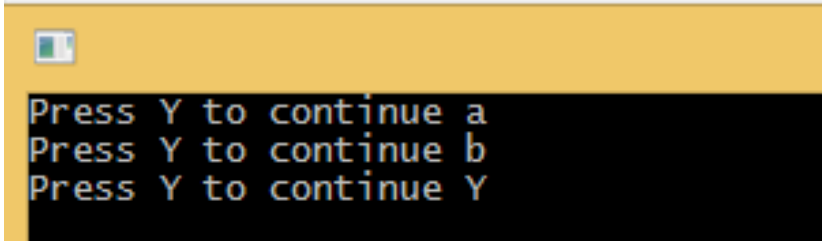
Block and Scope

- `begin .. end;` becomes `{ }`
- Within a block `{ }`, variables declared inside will be restricted to that block
- This is called Block Scope
- Note: many other programming languages, including Pascal, Javascript and PHP, uses Function Scope instead

While-loop and do-while loop

```
int x = 0, y = n-1;
bool found = false;
while (x <= y) {
  int mid = (x + y) / 2;
  if (a[mid] == m) {
    found = true;
    break;
  }
  if (a[mid] < m) {
    x = mid + 1;
  } else {
    y = mid - 1;
  }
}
```

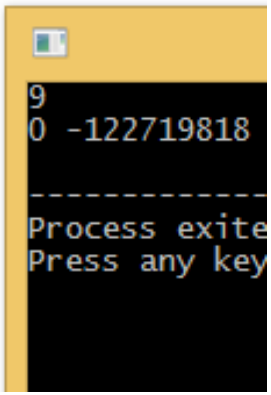
```
char c = ' ';
do {
  printf("Press Y to continue ");
  scanf(" %c", &c);
} while (c != 'Y');
// caution! opposite logic
```



Array

- C++ arrays are zero-based
- Tip: variables declared in the global scope requires no initialization

```
1   #include <cstdio>
2   int a[100000];
3   int main() {
4     int c[] = {1, 3, 6, 9, 10};
5     printf("%d\n", c[3]);
6     int b[100000];
7     int suma = 0, sumb = 0;
8     for (int i = 0; i < 100000; i++) {
9       suma += a[i];
10      sumb += b[i];
11    }
12    printf("%d %d\n", suma, sumb);
13    return 0;
14  }
```



String

- Again, strings have two flavors
- C strings is an array of character (similar to Pascal)

	C flavor	C++ flavor
Library	`#include <cstring>`	`#include <string>`
Declaration	`char s[50];`	`string s;`
Declare + init	`char s[] = "ABC";`	`string s = "ABC";`
Input	`scanf("%s", s);`	`cin >> s;`

C Strings are null-terminated

- Strings are enclosed with double quote "
- Chars are enclosed with single quote '

```
1   #include <cstdio>
2   #include <cstring>
3   int main() {
4     char s[] = "ABC";
5     printf("%d %d %d %d\n", s[0], s[1], s[2], s[3]);
6     printf("%d\n", sizeof(s));
7     s[0] = 'D';
8     printf("%s\n", s);
9     printf("%s\n", s + 1);
10    return 0;
11  }
```

```
65 66 67 0
4
DBC
BC
```

AMENDED

String Functions

- `int strlen(const char* s);`
- `strcpy(char* dest, const char* src);`
- `int strcmp(const char* s, const char* t);`
  - returns **NEGATIVE INTEGER** if `s < t`
  - returns 0 if `s == t`
  - returns **POSITIVE INTEGER** if `s > t`
- `int strncmp(const char* s, const char* t, size_t len);`
- `memset(void* dest, int data, size_t size);`

Math Functions

```
#include <cmath>
```

- Almost every math function returns double

```
printf("%lf %d %d\n", sqrt(4), sqrt(4), round(12.34));
printf("%lf %lf %lf\n", M_PI, sin(M_PI / 2), pow(2, 4));
printf("%lf %lf %lf\n", round(3.5), floor(3.5), ceil(3.5));
printf("%lf %lf %lf\n", round(-3.5), floor(-3.5), ceil(-3.5));
printf("%lf %lf %lf\n", log10(1000), log(M_E * M_E), log2(65536));
```

```
2.000000 0 1073741824
3.141593 1.000000 16.000000
4.000000 3.000000 4.000000
-4.000000 -4.000000 -3.000000
3.000000 2.000000 16.000000
```

- Though double abs(**double** x) is in <cmath>, abs(**int** x) is in <cstdlib>

Practice: 01005 - Napster Cheating

- **Length < 100**
- To read a line with spaces, use `gets(s);`

* DO NOT use `gets(s)` in real life!

```
1   #include <cstdio>
2   #include <cstring>
3   int main() {
4     char s[100];
5     gets(s);
6     int len = strlen(s);
7     for (int i = 0; i < [redacted]; i++) {
8       char c = s[i];
9       s[i] = s[[redacted]];
10      s[[redacted]] = c;
11    }
12    printf("%s\n", s);
13    return 0;
14  }
```

Coding Conventions

- For consistent display by editors / browsers
  - All indentation is to be done using spaces.
  - Each indentation level is 2 spaces wide.
- One space after comma (like English)
- **`if`**, **`for`**, and functions

```
if/for (...) {
   ...;
}
```

- All binary operators have a space on each side

```
1   #include <cstdio>
2   #include <cstring>
3   int main() {
4     char s[100];
5     gets(s);
6     int len = strlen(s);
7     for (int i = 0; i < (len - 4) / 2; i++) {
8       char c = s[i];
9       s[i] = s[len - 5 - i];
10      s[len - 5 - i] = c;
11    }
12    printf("%s\n", s);
13    return 0;
14  }
```

Functions

- Unlike Pascal, the return statement causes execution to exit the function immediately

```
 2  function absdiff(a, b: longint): longint;
 3  begin
 4    if (a < b) then absdiff := b - a
 5    else absdiff := a - b;
 6  end;
 7  begin
 8    writeln(absdiff(10, 3));
 9    writeln(absdiff(-2, 4));
10  end.
```

```
1   #include <cstdio>
2   int absdiff(int a, int b) {
3     if (a < b) return b - a;
4     return a - b;
5   }
6   int main() {
7     printf("%d\n", absdiff(10, 3));
8     printf("%d\n", absdiff(-2, 4));
9     return 0;
10  }
```

```
7
6

-----
Proce
```

Procedures and passing arrays

- Procedures are functions with a **`void`** return type
- Similar to Pascal, we can't pass an array to a function directly

```
1  #include <cstdio>
2  void method1(int b[]) {
3    printf("%d\n", sizeof(b) / 4);
4  }
5  void method2(int* b) {
6    printf("%d\n", sizeof(b) / 4);
7  }
8  void method3(int b[3]) {
9    printf("%d\n", sizeof(b) / 4);
10 }
11
12 int main() {
13   int n;
14   int a[100];
15   scanf("%d", &n);
16   for (int i = 0; i < n; i++) {
17     scanf("%d", &a[n]);
18   }
19   method1(a);
20   method2(a);
21   method3(a);
22   return 0;
23 }
```

```
3
6 8 9
1
1
1

-------
Process
Press a
```

Passing arrays

- In C++, we have three solutions
- The first and second solution is to pass the array size separately

```cpp
#include <cstdio>
int n;
void average(int a[]) {
  int sum = 0;
  for (int i = 0; i < n; i++) {
    sum += a[i];
  }
  printf("Average = %lf", 1.0 * sum / n);
}
int main() {
  int a[100];
  scanf("%d", &n);
  for (int i = 0; i < n; i++) {
    scanf("%d", &a[i]);
  }
  average(a);
  return 0;
}
```

```cpp
#include <cstdio>
void average(int a[], int size) {
  int sum = 0;
  for (int i = 0; i < size; i++) {
    sum += a[i];
  }
  printf("Average = %lf", 1.0 * sum / size);
}
int main() {
  int n;
  int a[100];
  scanf("%d", &n);
  for (int i = 0; i < n; i++) {
    scanf("%d", &a[i]);
  }
  average(a, n);
  return 0;
}
```

Passing arrays

- The third solution is similar to Pascal:
  By defining a new data type (struct)

```
1   #include <cstdio>
2   struct array {
3     int n;
4     int content[100];
5   };
6   void average(array a) {
7     int sum = 0;
8     for (int i = 0; i < a.n; i++) {
9       sum += a.content [i];
10    }
11    printf("Average = %lf", 1.0 * sum / a.n);
12  }
13  int main() {
14    array a;
15    scanf("%d", &a.n);
16    for (int i = 0; i < a.n; i++) {
17      scanf("%d", &a.content[i]);
18    }
19    average(a);
20    return 0;
21  }
```

```
3
6 8 9
Average = 7.666667
--------------------
Process exited after
Press any key to cor
```

Quiz: struct

- What is the output of the following program?

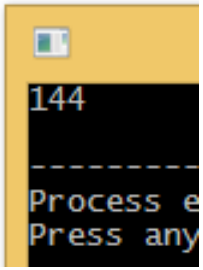
```
1   #include <cstdio>
2   struct student{
3     char name[30];
4     int number;
5   };
6   int main() {
7     printf("%d\n", sizeof(student));
8     return 0;
9   }
```

Passing by reference

- In Pascal, we add the **var** keyword to indicate that the parameter should be passed by reference
- In C++, we add **&** after the data type

```pascal
var x: longint;
procedure square(var n: longint);
begin
  n := n * n;
end;
begin
  x := 12;
  square(x);
  writeln(x);
end.
```

```cpp
#include <cstdio>
void square(int& n) {
  n *= n;
}
int main() {
  int x = 12;
  square(x);
  printf("%d\n", x);
  return 0;
}
```



Pointers

- A pointer can store the address of a variable
- It is always 4 bytes (on 32 bit systems)
- `int* a;`
- Some people use `int *a;` (note the space)

When declaring a pointer variable or argument, you may place the asterisk adjacent to either the type or to the variable name:

```
// These are fine, space preceding.
char *c;
const string &str;

// These are fine, space following.
char* c;    // but remember to do "char* c, *d, *e, ...;"!
const string& str;
```

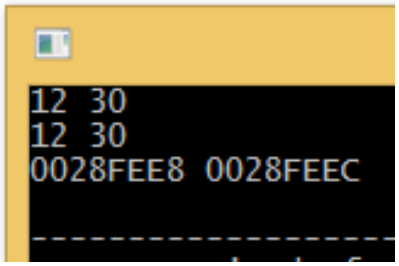
```
char * c;  // Bad - spaces on both sides of *
const string & str;  // Bad - spaces on both sides of &
```

You should do this consistently within a single file, so, when modifying an existing file, use the style in that file.

Getting the address of a variable (reference)

- The address of an variable can be obtained by prepending it with **&**
- In `scanf()`, we need to add **&** because C does not support passing by reference
- Instead, C functions use passing by address

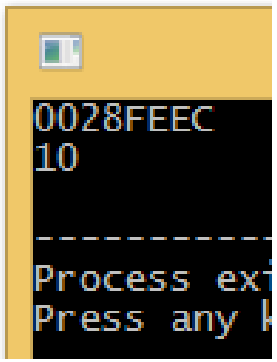
```
1  #include <cstdio>
2  int main() {
3    int x, y;
4    scanf("%d %d", &x, &y);
5    printf("%d %d\n", x, y);
6    printf("%p %p\n", &x, &y);
7    return 0;
8  }
```



Getting the value at that address (dereference)

- We can get back the value pointed by the pointer by prepending *

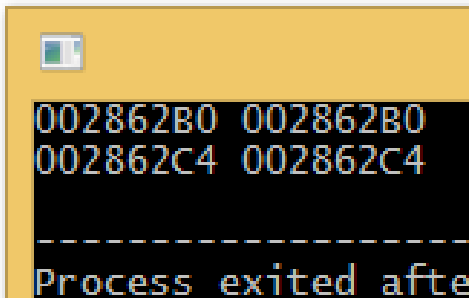
```
1  #include <cstdio>
2  int main() {
3    int a = 10;
4    int* p = &a;
5    printf("%p\n" , p);
6    printf("%d\n" , *p);
7    return 0;
8  }
```



The variable name of an array is a pointer

- Note that it takes account of the underlying data type
- i.e. 5 x 4 bytes = increments by 20 bytes (0x14)

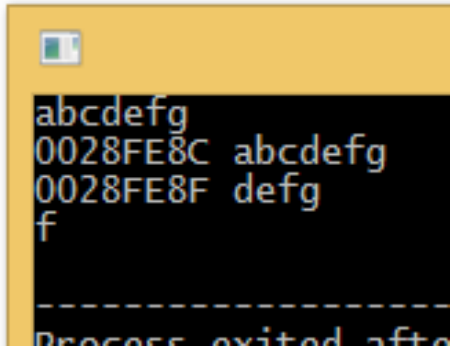
```
1  #include <cstdio>
2  int main() {
3    int a[10000];
4    printf("%p %p\n", a, &a[0]);
5    printf("%p %p\n", a + 5, &a[5]);
6    return 0;
7  }
```



cstring is an array, so it is a pointer!

- char is 1 byte
- s + n is the substring starting from the n-th (zero-based) position but be careful not to add more than the length of the string! (n <= length)

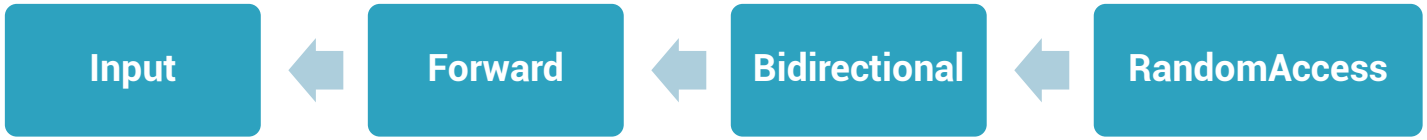
```
1  #include <cstdio>
2  int main() {
3    char s[100];
4    scanf("%s" , s);
5    printf("%p %s\n", s, s);
6    printf("%p %s\n", s + 3, s + 3);
7    printf("%c\n", *(s + 5));
8    return 0;
9  }
```



Standard Template Library (STL)

- C++ provides a set of libraries that are useful in programming competitions
- Templates are classes and functions that can operate with any data type

Algorithms



Algorithm	Requires	Purpose
reverse	BidirectionalIterator	Reverses content
sort stable_sort	RandomAccessIterator	Sort content
lower_bound upper_bound equal_range binary_search	ForwardIterator	Binary search
min_element max_element	ForwardIterator	Finding max and min
nth_element	RandomAccessIterator	Finding $(n-1)^{th}$ smallest element (Quick Select)
set_union set_intersection set_difference set_symetric_difference	InputIterator	$A \cup B$ (or) $A \cap B$ (and) $A \setminus B$ (-) $A \triangle B$ (xor)

Example: reverse

- A pointer is a RandomAccessIterator

function template

std::**reverse**

```
template <class BidirectionalIterator>
  void reverse (BidirectionalIterator first, BidirectionalIterator last);
```

Reverse range

Reverses the order of the elements in the range `[first,last)`.

The function calls `iter_swap` to swap the elements to their new locations.

The behavior of this function template is equivalent to:

```
template <class BidirectionalIterator>
  void reverse (BidirectionalIterator first, BidirectionalIterator last)
{
  while ((first!=last)&&(first!=--last)) {
    std::iter_swap (first,last);
    ++first;
  }
}
```

```
1  #include <cstdio>
2  #include <cstring>
3  #include <algorithm>
4  using namespace std;
5  int main() {
6    char s[100];
7    gets(s);
8    int len = strlen(s);
9    reverse(s, s + len - 4);
10   printf("%s\n", s);
11   return 0;
12 }
```

Example: sort

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;
int main() {
  int n = 7;
  int a[] = {5, 0, 1, 2, 6, 3, 4};
  sort(a, a + n);
  for (int i = 0; i < n; i++) {
    printf("%d%c", a[i], i == n - 1 ? '\n' : ' ');
  }
  return 0;
}
```

```
0 1 2 3 4 5 6
```

- More advanced sorting will be covered in "Searching and Sorting"

Containers

- vector: Dynamic array
- deque: Double-end queue
  (can used as queue/stack)
- list: Bidirectional linked-list
- set / map: Binary search tree
- unordered_set / unordered_map: Hash table
- priority_queue: Heap

- e.g.: `vector<int>` is a dynamic array of `int`
- However, cstring cannot be used with STL containers. Use C++ string instead.

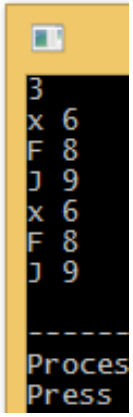
Containers

Container	Insertion	Access	Erase	Find	Iterator
(Regular array) vector / string	Back: O(1) Other: O(n)	O(1)	Back: O(1) Other: O(n)	Sorted: O(log n) Other: O(n)	RandomAccess Iterator
deque (stack / queue)	Back/Front: O(1) Other: O(n)	O(1)	Back/Front: O(1) Other: O(n)	Sorted: O(log n) Other: O(n)	RandomAccess Iterator
list	Back/Front: O(1) With iterator: O(1) Index: O(n)	Back/Front: O(1) With iterator: O(1) Index: O(n)	Back/Front: O(1) With iterator: O(1) Index: O(n)	O(n)	Bidirectional Iterator
set / map	O(log n)	-	O(log n)	O(log n)	Bidirectional Iterator
unordered_set / unordered_map	O(1) or O(n)	O(1) or O(n)	O(1) or O(n)	O(1) or O(n)	ForwardIterator
priority_queue	O(log n)	O(1)	O(log n)	-	

Pair

- pair<T1, T2> is a struct that stores two variables (of same type or different types)

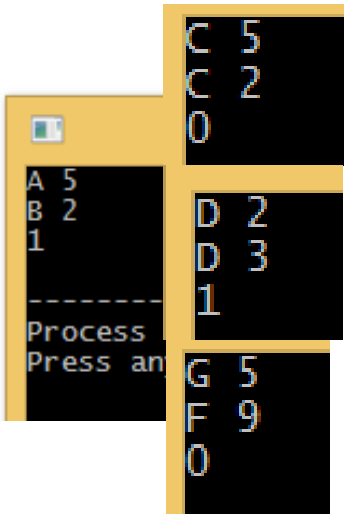
```cpp
#include <cstdio>
#include <algorithm>
using namespace std;
int main() {
  pair<char, int> p[5];
  int n;
  scanf("%d", &n);
  for (int i = 0; i < n; i++) {
    scanf(" %c %d", &p[i].first, &p[i].second);
  }
  for (int i = 0; i < n; i++) {
    printf("%c %d\n", p[i].first, p[i].second);
  }
  return 0;
}
```



Pair

- Pairs of matching T1 T2 can be compared
- The `operator<(const T1& a, const T2& b)` is:
  - `if (a.first != a.first) return a.first < b.first;`
  - `return b.second < b.second;`

```
1   #include <cstdio>
2   #include <algorithm>
3   using namespace std;
4   int main() {
5     pair<char, int> a, b;
6     scanf("%c %d %c %d", &a.first, &a.second, &b.first, &b.second);
7     printf("%d\n", a < b);
8     return 0;
9   }
```



Practice: 01020 - Inner Join

- Not mentioned in the problem statement: **the input keys are sorted**
- The range of the key can be up to $2^{31} - 1$
- We can use binary search on the first table when processing rows in the second table
- Time complexity: O(N) or O(N lg N)
- There are MANY other ways to solve this task

Practice: 01020 - Inner Join

- By using STL, the problem is much simplified
- map is a **binary search tree** that allows us to find a key in O(lg N) time
- For each B, we add count(A) * count(C) to ans

```cpp
#include <cstdio>
#include <map>
using namespace std;
int main() {
  int n, m, x, y;
  scanf("%d %d", &n, &m);
  map<int, pair<int, int> > b;
  for (int i = 0; i < n; i++) {
    scanf("%d %d", &x, &y);
    b[y].first++;
  }
  for (int i = 0; i < m; i++) {
    scanf("%d %d", &x, &y);
    b[y].second++;
  }
  int ans = 0;
  for (map<int, pair<int, int> >::iterator it = b.begin(); it != b.end(); it++) {
    ans += it->second.first * it->second.second;
  }
  printf("%d\n", ans);
  return 0;
}
```

- Iterators need to be dereferenced to get the value **`(*it)`**
- When a map iterator is dereferenced, its **`first`** is the key and **`second`** is the value
- The **`a->b`** operator means **`(*a).b`**
- This solution works even if input is not sorted / requires sorted output

Practice: 01021 - Left Join

- Unfortunately, this time we need to output the rows, so we must also store the values of A and C
- Each B may contain many A and C
- We cannot declare a 25000x25000 array
- Max number of output rows = 250000

Practice: 01021 - Left Join

- vector is a **dynamic array** that **grows in size** as items are inserted
- We change the map to
  `map<int, pair<vector<int>, vector<int> > > b`
- The key of the map stores column b
- The value pair of vectors stores a and c respectively

```cpp
#include <cstdio>
#include <vector>
#include <map>
using namespace std;
int main() {
  int n, m, x, y;
  scanf("%d %d", &n, &m);
  map<int, pair<vector<int>, vector<int> > > b;
  for (int i = 0; i < n; i++) {
    scanf("%d %d", &x, &y);
    b[y].first.push_back(x);
  }
  for (int i = 0; i < m; i++) {
    scanf("%d %d", &x, &y);
    b[y].second.push_back(x);
  }
  for (map<int, pair<vector<int>, vector<int> > >::iterator it = b.begin(); it != b.end();
    for (int i = 0; i < it->second.first.size(); i++) {                    it++) {
      if (it->second.second.size() == 0) {
        printf("%d %d null\n", it->second.first[i], it->first);
      }
      for (int j = 0; j < it->second.second.size(); j++) {
        printf("%d %d %d\n", it->second.first[i], it->first, it->second.second[j]);
      }
    }
  }
  return 0;
}
```

- To insert elements into vector, use `.push_back()`.
- To get its length, use `.size()`