

# VE280 Final

---

@qinhangwu

## Toolchain

---

### g++

```
g++ -Wall -Werror -O2 -pedantic -Wextra -std=c++17 -o ptr.exe ptr.cpp
```

### diff

```
diff FILE1 FILE2
```

- **c** change-内容改变
- **a** addition
- **d** deletion
- **num1 c num2** FILE1 num1行变成 FILE2 num2行
- **<** 从FILE1种删去
- **---** 分隔符
- **>** FILE2中新增

ref: [http://www.ruanyifeng.com/blog/2012/08/how\\_to\\_read\\_diff.html](http://www.ruanyifeng.com/blog/2012/08/how_to_read_diff.html)

### valgrind

```
valgrind --leak-check=full ./ex1
```

alternative: `g++ -fsanitize=leak`

### ASCII

<https://theasciicode.com.ar/>

## Abstract Rules

---

### RME

RME: abbreviation for REQUIRES, MODIFIES, EFFECTS (comment style)

Lec6, Data Abstraction

### substitution principle

Subtype S from supertype T:

- code written to correctly use T is still correct if it uses S

### interface

Hide both the data members and method implementations so as to

- simplify the class definition

- hide implementation detail

Solution:

- create an "interface-only" virtual base class

single static instance

```

1 // in impl.cpp:
2 class IntSetImpl : public IntSet{/*...*/};
3
4 static IntSetImpl impl; // derived-class instance, only visible to impl.cpp
5 IntSet *getIntSet(){
6     return &impl;
7 }
8
9 // in main.cpp for users:
10 IntSet *set = getIntSet();

```

ref: Lec 16

## representation invariant

A set of conditions that the data members of an ADT must always satisfy.

- There exist representation invariant(s) for a certain ADT
- Each data element is truly private
- The representation invariant holds immediately before exiting each method (including **constructor**)

example:

```

1 class IntSet{
2     private:
3         int elts[MAXELTS];
4         int numElts;
5         // ...
6 }

```

- unsorted `IntSet`: The first `numElts` members of `elts` contain the **integers** comprising the set, with **no duplicates**
- sorted `IntSet`: The first `numElts` members of `elts` contain the **integers** comprising the set, **from lowest to highest**, with **no duplicates**.

ref: Lec 16

## container of pointers

- At-most-once invariant
  - any object can be linked to at most one container at any time through pointer
- Existence
  - An object must be dynamically allocated before a pointer to it is inserted
- Ownership
  - Once a pointer to an object is inserted, that object becomes the property of the container. No one else may use or modify it in any way

- Conservation
  - When a pointer is removed from a container, either the pointer must be inserted into some container, or its referent must be deleted.

## 文件

### redirection

- `<` **input** redirect
- `>` **output** redirect

### mode

#### r vs rb

- When using `'r'` (*text mode*) to read the text from a file, `\r\n` will be automatically converted into `\n` on Windows.
- When using `'rb'` (*binary mode*) to read the text from a file, there is no format transformation. Therefore, you will read two characters `\r` `\n` separately when meeting a newline on Windows.
- Input redirection `<` will read the file with text mode by default.
- 即使上传的testcase是CRLF格式, JOJ依然会以 `'r'` 方式读取, 将其转换成 `\n`

#### w vs rw

TBA

ref: <https://blog.csdn.net/guyue6670/article/details/6681037>

## C++ file stream

```
1  #include <fstream>
2  using namespace std;
3
4  ifstream ifs;
5  ifs.open("a.txt");
6  if(!ifs) {
7      cerr << "cannot open a.txt\n";
8      return -1;
9  }
10
11  int digit;
12  ifs >> digit;
13
14  string line;
15  getline(ifs, line);
16
17  ofstream ofs;
18  ofs.open("a_out.txt");
19  ofs << digit;
20
21  ifs.close();
22  ofs.close();
```

pass to function by reference

```

1 void input(ifstream &ifs){
2     int digit;
3     ifs >> digit;
4 }
5
6 // ...
7 ifstream ifs;
8 ifs.open("a.txt");
9 input(ifs);



```

## Characters

[Link-to: ASCII Table](#)

sizeof(char) = 1

keyword in C

ASCII	显示	描述
0		NULL, 空字符, <code>\0</code> , 字符串结束符, 不是空格! (char) 0 is a char, while <code>'\0'</code> is (unintuitively) an <code>int</code>
-1 / #undefined	#random	<b>EOF</b> , 非ASCII字符, 被 <code>stdio.h</code> 定义为-1, 打印时可能是 unsigned char 范围内的任意字符. It is system-defined, and you should not care what its value is. 需要用 <code>int</code> 来hold EOF, 而不是 <code>char</code> .
32		<b>space</b> , 空格
10		LF, <b>Line Feed</b> , <code>\n</code> , 换行符 (光标移动到下一行). moves the cursor down to the next line without returning to the beginning of the line — <i>In a *nix environment \n moves to the beginning of the line.</i>
13		CR, <b>Carriage return</b> , <code>\r</code> , 回车 (光标移动到行首). moves the cursor to the beginning of the line without advancing to the next line

## NULL (\0)

NULL is an ASCII character, whose decimal value is 0.

You can try this code to check your understanding:

```

1 printf("%d", 0=='\0');
2 printf("%c %d", '\0', '\0');

```

And NULL is always a mark of the end of character array ( `char *` ). For example, `strlen` will count from the start until it finds a `\0`. Here is a possible realization of `strlen`.

```

1 | int strlen(char *s)
2 | {
3 |     int res = 0;
4 |     while (*s != '\0')
5 |         res++, s++;
6 |     return res;
7 | }

```

Likewise, if you print a character array, it will print the character until a `\0`. And if you read a sequence of character, like

```

1 | char s[20];
2 | scanf("%s",s);//input "Hello, world."

```

The program will automatically **set** a `\0` and the end of the string (right after `.`). Due to the existence of `\0`, you should always reserve extra spaces for your character array, to avoid data overwritten.

```

1 | //This code may behave differently according to the machine.
2 | #include <stdio.h>
3 | int main()
4 | {
5 |     char s[2], s[2];
6 |     s[0] = 'a', s[1] = 'b';
7 |     scanf("%s", s);
8 |     printf("%s", s);
9 | }
10 | //If you enter "a", s[1] will be set by 1

```

## Newline (LF,\n)

Newline is an ASCII character, whose decimal value is 10. It moves the cursor to the next line.

You can try

```

1 | printf("%d",10=='\n');
2 | printf("%c %d",'\n','\n');

```

## Carriage Return (CR,\r)

CR is an ASCII character, whose decimal value is 13. It moves the cursor to the beginning of the line.

You can try

```

1 | printf("%d\n",13=='\r');
2 | printf("123\r23");//This allows you rewrite this line.

```

## CRLF (CR-LF)

`CRLF` means a sequence of two characters where `CR` is followed by `LF`. They're used to note the termination of a line. However, they're dealt with differently in today's popular Operating Systems. For example: in Windows both a `CR` and `LF` are required to note the end of a line, whereas in Linux/UNIX/MacOS X a `LF` is only required.

If you are using Windows, you can open `notepad.exe`, just type a return and save the file. You will see the file takes up 2 Bytes disk memory, indicating the file contains `\r\n` (each of which is 1 Bytes).

If interested, you may use software like `Binary Viewer` to check the binary code of a text file.

#### 编辑器

- VSC: 默认CRLF
- Clion: 默认LF
- notepad: 默认CRLF
- 可以通过Binary Viewer之类的软件检查

NOTES: more in VG101-SU20-Lab11-Manual

ref: EOF, <https://stackoverflow.com/questions/7622699/what-is-the-ascii-value-of-eof-in-c>

ref: CLRF, <https://stackoverflow.com/questions/47761108/replace-cr-lf-with-lf-in-c>

ref: C array initialization, <https://stackoverflow.com/questions/18688971/c-char-array-initialization>

ref: (char) 0 vs. '\0', <https://stackoverflow.com/questions/7578632/what-is-the-difference-between-char0-and-0-in-c>

ref: clearing a char array in c, <https://stackoverflow.com/questions/632846/clearing-a-char-array-c>

## C++ string

`std::string`

#### 基本性质

- `#include <string>`
- `using namespace std;`
- 默认初始化为空 `""`
- `str.length()` 返回长度
  - 慎用 `\0`
- 可以通过索引 `[]` 得到 `char` 类型
- `str.erase`
  - e.g. string去头去尾
  - ref: <https://www.cnblogs.com/ylwn817/articles/1967689.html>

#### get

单个字符

```

1 // getchar() works well in c++, but you cannot assign the stream.
2 // a better way is to use .get()
3 char ch;
4
5 while(cin.get(ch)){
6     cout.put(ch);
7     cout << "[" << (int)c << "] | ";
8 }

```

## cin

过滤空格、换行，持续读入字符串

```

1 using namespace std;
2 // ...
3 string str;
4
5 // 过滤空格、换行，持续读入字符串
6 while(!(cin >> str).eof()) cout << str; // Issue: 如果最后一个字符串后无空格/换行
    符，直接EOF的话，该字符串不会被读到
7
8 while(cin >> str) cout << str; // A better approach, 没有这个问题
9
10 bool isEOF = (bool)(cin >> str);
11 while(isEOF){
12     cout << str << "|| ";
13     isEOF = (bool)(cin >> str);
14 } // Equivalent, 但必须有强制类型转换

```

## getline

```

1 // EFFECTS: read a whole line each time until reaching EOF;
2 // printing it whenever the input status is normal
3 using namespace std;
4 // ...
5 string line;
6
7 while(!getline(cin, line).eof) cout << line; // Issue: 如果最后一个字符串后无空
    格/换行符，直接EOF的话，该字符串不会被读到
8
9 while(getline(cin, line)) cout << line; // A better approach

```

## string char 互转

char -> string

```

1 b = 'a';
2
3 // string a = b; // wrong: 没有从char到string的构造函数
4
5 string a;
6 a = b; // Correct

```

alternative: `ostringstream`

## string -> char

```
1 string str = "q";
2
3 char ch = str[0]; // Correct
```

alternative: `istreamstream`

## string char\* 互转

常用于处理program argument

### char\* -> string

```
1 int main(int argc, char* argv[]){
2     string keyStr;
3
4     for(int i=1;i<argc;i++){
5         keyStr = argv[i]; // Correct, 直接赋值
6         if(keyStr=="A3") cout << "true\n";
7         else cout << "false\n";
8         // ..
9     }
10    // ..
11 }
```

### string -> char\*

```
1 string str = "abcd";
2
3 const char *ch2 = str.c_str(); // Correct, but you cannot modify it
4
5 char *cstr = new char[str.length() + 1];
6 strcpy(cstr, str.c_str()); // Correct, you can modify it
7 // ...
8 delete [] cstr; // remember to delete it
9
10 // char *ch = str.data(); // not standard
11
12 std::vector<char> cstr(str.c_str(), str.c_str() + str.size() + 1); //
    alternative, in modern c++
```

ref: <https://stackoverflow.com/questions/7352099/stdstring-to-char>

## string int/float 互转

### int to string

```
1 #include <string>
2 std::string s = std::to_string(42);
```

### string to int

- `atoi` (c)
  - 会fail style checker



- `stoi` (c++, throw an exception when the string is not convertible to an int; when `atoi` will silently fail)

you can also consider using string stream.

## string stream

```
1 #include <sstream>
2 #include <string>
3 using namespace std;
4
5 string str = "-19 39 109 2 A 33";
6 istringstream iss;
7 iss.str(str);
8
9 int cnt=0, num;
10 while(iss >> num){
11     cout << num << " "; // -19 39 109 2
12     cnt+=num;
13     if(iss.fail()) break; // 这里没必要加
14 }
15 iss.clear(); // 不加的话，下一次iss就不能重复使用
```

## const string

```
1 string str1 = "123"; // not const
2 const string str2 = "123"; // const
3
4 // c string to be added
```

## isEmpty

```
1 getline(cin, line);
2 istringstream iss;
3 iss.str(line);
4 int i;
5 bool isIssNotEmpty = (bool)(iss >> i); // 过滤行末空格、换行符
6
7 if(iss.rdbuf()->in_avail()==0){notEnoughOperand err; throw err;} // 计算包含
8 // ...
9 iss.clear();
10
11 string str1 = ""; cout << str1.empty(); // 1
12 string str1 = "1"; cout << str1.empty(); // 0
13 string str1 = " "; cout << str1.empty(); // 0
14 string str1 = "\n"; cout << str1.empty(); // 0
15 string str1 = "\0"; cout << str1.empty(); // 1
```

## Qualifier

### const

- only types with values may be declared as `const`.

- array can hold const values, but cannot be declared as const
- high level function can not be declared as const
  - member function in class can be declared as const
- reference cannot be declared as const
  - can declare reference to const object

## 2+2

```

1  int x = 3;
2
3  int *ptr1 = &x;
4  int * const ptr2 = &x; // const pointer pointing to a non-const int
5  const int *ptr3 = &x; // non-const pointer pointing to a const int
6  const int * const ptr4 = &x; // const pointer pointing to a const int

```

## conversion

general rule: permit that const object cannot be modified

```

1  int x = 3;
2  int *ptr1 = &x; // OK -- does not permit const object to be modified
3
4  const int *ptr2 = ptr1; // OK -- does not permit const object to be modified
5
6  ptr1 = ptr2; // ERROR -- compiler sees that ptr2 is pointing to a const
   object, but ptr1 is not a pointer to const

```

```

1  // void f(int &a){ cout << a;} // Wrong: const qualifier will be discarded
2  void f(const int &a){ cout << a;} // Correct
3  // ...
4  const int x=3;
5  f(x);

```

ref: eecs280 notes

## const member func

```

1  class A{
2  public:
3      void f(int x) const;
4  };
5  void A::f(int x) const{cout << x;}
6  // ...
7  A a;
8  a.f(1);

```

## virtual

- We define a virtual function in the base class and use base class pointers to access the function, so that all the grouped types share the interface.
- The virtual property of a member function is **inherited**.
- `virtual` keyword **cannot be applied to the constructor**. Indeed, when creating an instance of a derived class, the base class constructor will also be called (right before the

derived class constructor is called).

- When we use a pointer of class A to invoke a non-virtual function foo, A::foo would always be invoked.
- When we use a pointer of class A to invoke a virtual function bar, which function is invoked depends on the type of the object the pointer is pointing to.

```
1 // Code in VG101-SU20-Final-Review-Notes
2 // we use the class defined in the previous section
3 Bird b("1");
4 b.talk(); // tweet
5 Chicken c("2");
6 c.talk(); // chee
7 Duck d("3");
8 d.talk(); // quack
9
10 Bird *ptrB = &b;
11 Bird *ptrC = &c;
12 Bird *ptrD = &d;
13 ptrB->talk();
14 ptrC->talk();
15 ptrD->talk();
16 // if the virtual keyword is included, it will print tweet chee quack
17 // if not, it will print tweet tweet tweet
```

{}

## pure virtual function

```
1 virtual void insert(int v) = 0; // a member function from an interface-only
  base class
```

- such an interface-only class cannot be instantiated, because certain methods lack implementation.
- methods should be implemented in derived class.

ref: Lab 7

ref: <https://stackoverflow.com/questions/22750855/undefined-reference-to-method-in-parent-class>

## static

ref: <https://www.cnblogs.com/XuChengNotes/p/10403523.html>

## extern

## explicit

## override

申明该函数将 override parent class 的对应函数。compiler 会检查是否有对应的 virtual member function in the parent class, 没有则报错

## dynamic\_cast

An alternative to dynamic binding.

assert object是你期望的type, 若不是, 返回nullptr

要求: 操作对象是多态类型(至少一个virtual method)

```
1  Chicken chicken("Myrtle");
2  Bird *b_ptr = &chicken;
3  Chicken *c_ptr = dynamic_cast<Chicken *>(b_ptr);
4  if (c_ptr) { // check for null
5      // do something chicken-specific
6  }
```

## friend

A marks be as `friend` -> A grant B access to visit A's private member

```
1  class Bar {
2  private:
3      int data=3;
4      friend void foo (const Bar &b);
5      friend class Baz;
6  };
7
8  class Baz {
9  private:
10     Bar B;
11 public:
12     void print(){ cout << B.data << endl;} // access granted
13 };
14
15 void foo (const Bar &b){
16     cout << b.data << endl; // access granted
17 }
```

**Not mutual:** If Class A declares Class B as friend. Class B can access Class A's private member, but the other way around doesn't work.

## Func Signature

Factors that determine the function signature:

- function name
- type of the parameter
- (if the function is a class member) **const** / volatile qualifier
- (for function template) the type of its template arguments & the **return type**

Note: the **Standard does forbid a function declaration that only differs in the return type**

ref: [Is the return type part of the function signature?](#)

ref: [Top-level const doesn't influence a function signature](#)

## ADT

from supertype to subtype:

1. add operations

1. new data member
2. new member function
2. strengthen the postconditions
  1. EFFECTS clause
  2. return type
3. weaken the preconditions
  1. REQUIRES clause
  2. argument type

## function overload

- must have different signature

## operator overload

- most (but not all) operators can be overloaded
- ordinary nonmember functions
- class member functions

```

1  A operator+ (const A &left, const A &right);
2
3  A A::operator+ (const A &right);
4
5  A &operator= (); // to be added

```

quick review on prototypes:

```

1  // Unary
2  A A::operator-() const; // -a
3  A& A::operator++(); // ++a
4  A A::operator++(int); // a++
5  A& A::operator--(); // --a
6  A A::operator--(int); // a--
7
8  // Binary
9  A& A::operator= (const A& rhs);
10 A& A::operator+= (const A& rhs);
11 A& A::operator-= (const A& rhs);
12 A operator+ (const A& lhs, const A& rhs);
13 istream& operator>> (istream& is, A& rhs);
14 ostream& operator<< (ostream& os, const A& rhs);
15 bool operator== (const A& lhs, const A& rhs);
16 bool operator!= (const A& lhs, const A& rhs);
17 bool operator< (const A& lhs, const A& rhs);
18 bool operator<= (const A& lhs, const A& rhs);
19 bool operator> (const A& lhs, const A& rhs);
20 bool operator>= (const A& lhs, const A& rhs);
21 // Especially:
22 const T& A::operator[] (size_t pos) const;
23 T& A::operator[] (size_t pos);

```

## constructor

## member-initializer list

少赋值一次, better efficiency

## default argument

error: default argument given for parameter

既可以在类的声明中, 也可以在函数定义中声明缺省参数, 但不能既在类声明中又在函数定义中同时声明缺省参数。

**You can declare default arguments in the class declaration or in the function definition, but not both.**

## template

typename

template: where to add

- template <class T>
- <typename T>

## Big Three

```
1 // destructor
2 template <class T>
3 Dlist<T>::~~Dlist() {
4     removeAll();
5 }
6 // copy constructor
7 template <class T>
8 Dlist<T>::Dlist(const Dlist &l): first(nullptr), last(nullptr) {
9     copyAll(l);
10 }
11 // assignment operator
12 template <class T>
13 Dlist<T> &Dlist<T>::operator=(const Dlist &l) {
14     if (this != &l) {
15         removeAll();
16         copyAll(l);
17     }
18     return *this;
19 }
```

## ADT in slides

---

### IntSet

Slide sample Q1

### linked list

审题

single-ended vs doubly-ended

singly-linked vs doubly-linked

# Iterator

pointer > iterator > subscript

## STL

- Sequential Containers
  - let the programmer control the order in which the elements are stored and accessed. The order doesn't depend on the values of the elements.
- Associative Containers
  - store elements based on their values.
- Container Adaptors
  - take an existing container type and make it act like a different type

## vector | sequential container

### initialization

```
1 vector<T> v1; // empty vector v1
2 vector<T> v2(v1); // copy constructor
3 vector<T> v3(n,t); // construct v3 that has n elements with value t
4 vector<T> v4(begin,end); // create vector v4 with a copy of the elements
  from the range denoted by iterators begin and end.
5
6 vector<int> front(v.begin(), mid); // init with another vector
7
8 deque<string> ds(10, "abc"); // init with another deque
9 vector<string> vs(ds.begin(), ds.end());
10
11 int a[] = {1, 2, 3, 4}; // init with another array
12 unsigned int sz = sizeof(a) / sizeof(int);
13 vector<int> vi(a, a+sz);
```

**size** `vector<int>::size_type sz = v.size()`

**subscripting** `v[i]`

```
1 vector<int>::size_type n;
2 v[n] = 0; //must ensure that the v.size() > n
3 for (n = 0; n != v.size(); ++n) {
4     v[n] = 0;
5 }
```

### iterator

\* ++ -- == !=

```

1 // ivec is of type vector<int>
2 vector<int>::iterator begin = ivec.begin();
3 vector<int>::iterator end = ivec.end();
4 while (begin != end) {
5     cout << *begin++ << " ";
6     // 1. get the value of *begin
7     // 2. cout << *begin << " ";
8     // 3. begin++;
9 }

```

- end 在末尾元素的后一个
- special operation:
  - iter+n, iter-n
  - >, >=, <, <=

### add/remove

v.push\_back(t); // copy

v.pop\_back();

### insert/erase

v.insert(p, t)

- inserts element with value `t` right before the element referred to by iterator `p`.
- returns an **iterator** referring to the element that was added

v.erase(p)

- removes element referred to by iterator `p`.
- returns an iterator referring to the element after the one deleted, or an off-the-end iterator if `p` referred to the last element.

### Notes

- 改变元素数量，之前的iterator失效

## deque | sequential container

大多和vector一样，新增两method

- zici `d.push_front(t)`
- zici `d.pop_front(t)`

## list | sequential container

大多和vector一样，差异：

- not support subscripting
- not support iterator arithmetic, i.e. cannot do `it+3`, you **can only** use `++/--`
- no relational operation `<`, `<=`, `>`, `>=`, you **can only** use `==` and `!=` to compare
- zici `l.push_front(t)`
- zici `l.pop_front(t)`

## map | associative container

(key, value) pairs



Rule: key cannot be duplicate

## set | associative container

contains only a key

## stack | container adaptor

### LIFO

- default container used is deque
- can be implemented with vector or list

```
1 // Assume deq is deque<int>
2 stack<int> stk(deq);
3
4 // Assume ivec is vector<int>
5 stack<int, vector<int> > stk(ivec); // space: not to confuse with >> opr
6
7 // Assume ilst is list<int>
8 stack<int, list<int> > stk(ilst);
```

```
stack<string, vector<string> >
```

## queue | container adaptor

### FIFO

- default container used is deque
- can be implemented with list, **but not vector**

More see notes P12 / cppreference