第 1 章 快速入门

1. iostream □□□□□□□□□□ istream □ ostream □□□□□□□□□□□□□□
□□ cerr □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
clog □□□□□□□□□□□□□□□□

2. endl □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□ endl □□□

3. □□□□□□□□

第 2 章 变量和基本类型

1. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□

2. C++ □□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□
□□=□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□

```
int ival(1024); // direct-initialization
int ival = 1024; // copy-initialization
```

□□□ C++ □□□“□□□□□□□”□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□

3. □□□□□□□□□□□□□□□ 0□□□□□□□□□□□□□□□□□□□□□
□□□

4. extern □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□ □ extern □□□

```
extern int i; // declares but does not define i
int i; // declares and defines i
```

5. const □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□ const □□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
const □□□ extern□□□□□□□□□□□□ const □□□ □ const □□□□□

6. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“&”□□□
□□□ const □□□□□ const □□□□□□

```
const int ival = 1024;
const int &refVal = ival;
```

□ const □□□□□□□□□□□□□□□□□□ const □□□□□□□□□□□□□
□□□□□□□□□□□□□ □□□□□□□

7. typedef □□□□□□□□□□□□□ □ typedef □□□

```
typedef double wages; // wages is a synonym for double
typedef int exam_score; // exam_score is a synonym for int
typedef wages salary; // indirect synonym for double
```

8. □□□□□□□□□□ enum□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□

```
enum open_modes {input, output, append};
```

□ 1□□□□□□ 0□□□□□□□□□□□□□□□□□□ 1□

9. □□□□□□□□□□□□□□□□□□□

10. □□□□□□□□□□□□□□□□□□□□□□

11. □□□□□□□□□□□□□□□□□□□□□□□□□

```
#ifndef SALESITEM_H
#define SALESITEM_H

#endif
```

12. □□□□□□□□□□□□<>□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□

```
#include <standard_header>
#include "my_file.h"
```

第 **3** 章 标准库类型

1. string □□□□□□□□□□□□□ vector □□□□□□□□□□□□□□□□ string □□□ vector □□□

2. □□□□□□□□□□ string □ vector□□□□□□□□□□□□□□□□□□□□ □□□□□□□□ bitset□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□

3. □□ using □□□□□□□□□□□ namespace_name:: □□□□□□□□□□□□□ □□ using □□□□□□□□

```
using namespace::name;
```

4. □□□ vector □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□ vector □□□□□□□ class template□ □

5. □□□□□□□□□□□□□□ vector□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□ begin □ end □□□□□ □□□□□□□

6. □□□□□□ iterator □□□□□□□□□□□□□□□□ const □□□□□□□□□ const_iterator □□□□□□□□□□□□□□□□ const □□□□□□□□□□□□ □□□□□□□□□□□□□

7. □□□□□□□□□□□□□□□□□□□□□□□ 0□□□ 1□□□□□□□□ □□□□□□□□□ yes/no □□□□□□□□□□□□□□□□□□□□□□ bitset □□□□□□□□□□

8. □□□ bitset □□□□□ bitset □□□□□□□□□□□□□□□□□□□□

```
bitset<32> bitvec; // 32 bits, all zero
```

第四章 数组和指针

1. □□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□

- □□□□□□□□□□□□□□□□□□□□□ 0□ □□ 0□□□
- □□□□□□□□□□□□□□□□□□□□□
- □□□

2. □ vector □□

3. C++ □□□□□□□□ const □□□□□□□□□□ const □□□□const□□ const□

```
const double *cptr;
```

const □□□ cptr □□□□□□□□□□□□□□ cptr □□□

4. □□□ const □□□□□□□□□□□□□□ const □□□□□□□□□□□□□□□□□□□□ void* □□□□ const □□□□□□□□□□□ const void* □□□□□□□ const □□□□□□□

5. □□□ const □□□□□□□C++ □□□□□□ const □□□□□□□□ const□ ——□□□□□□□□□

```
int errNumb = 0;
int *const curErr = &errNumb; // curErr is a constant pointer
```

6. □□□ const □□□□const □□□□□□□□□□□□□

7. □□□□□ C □□□□□□□□□□□□□□□□□□□□□□□□ null□□□

```
char ca[] = {'C', '+', '+'}; // not null-terminated
cout << strlen(ca) << endl; // disaster: ca isn't
null-terminated
```

8. □□□C □□□□□□□□□□□□□ malloc □ free □□□□□□□□□□□□□□□ C++ □□□□□ new □ delete □□□□□□□□□□□

9. □□□new □□□□□□□□□□□□□□□□□□□□□□ □new □□□

```
int *pia = new int[10];
```

10. □□□□□□□□ □□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□

```
string *psa = new string[10];          // array of 10 empty strings
int *pia = new int[10];                // array of 10 uninitialized ints
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
int *pia2 = new int[10] ();
```

11. C++ □□□□□□□□□□ 0 □□□□□□□□□□□□□□ new □□□□□□□ 0 □□□□□□□□

第五章 表达式

1. □□□□□□□□□□□□□□□□□□□□□□□□□□

- □□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□ bool □□□
- □□□

2. □□□□□□□□□□ int □□□□□□□ char□ signed char□ unsigned char□ short □ unsigned short□ □□□□□□□□□□□□□□□□□ int □□□□□□□□□□ int □□□□□□□□□□□□ unsigned int□
3. □□□□ const □□□□□ const □□□□□□□□□□□ const □□□□□ const □ □□□□□□□□□□ const □□□□□□□□ const □□□□□□□□□□ const □ □□□□□

```
int i;
const int ci = 0;
const int &j = i;
const int *p = &ci;
```

4. □□□□□□□□□□□□□□ cast□ □□□□□□□□□□□□□□□□□□□ □□ static_cast□ dynamic_cast□ const_cast □ reinterpret_cast□ □□□□□□□□□ □□□□□□□□□□ □ static□ dynamic□ const□ reinterpret□

```
cast-name<type>(expression);
```

- □□□□□□□□□□□□□□□□□□ static_cast □□□□□

```
double d = 97.0;
char ch = static_cast<char>(d);
```

- dynamic_cast □□□□□□□□□□□□□□□□□□□
- const_cast □□□□□□□□□□□□□□ const □□□
- reinterpret_cast □□□□□□□□□□□□□□□□□□□□□□

5. □□□□□□□□□□□□□□□□□□□□□□□□□□C++ □□□□□□□□□

- throw □□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□ throw □□□□□□□□
- try □□□□□□□□□□□□□□□□□try □□□□ try □□□□□□□□□□□ □□ catch □□□□□□ try □□□□□□□□□□□throw□ □□□□ □□□□□□ □□ catch □□□□□□□□□“□□”□□□ catch □□□□□□□□□□
- □□□□□□□□□□□□□□□□ throw □□□□ catch □□□□□□□□□□ □□

6. □□□□ throw □□□□□□□□ throw □□□□□□□ throw □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ throw □□□□□□□□□□□□□□□□□□

```
if (!item1.same_isbn(item2))
throw runtime_error("Data must refer to same ISBN");
std::cout << item1 + item2 << std::endl;
```

7. try □□□□□□□□□□

```
try {
      program-statements
} catch (exception-specifier) {
      handler-statements
} catch (exception-specifier) {
      handler-statements
} //...
```

try □□□□□ try □□□□□□□□□□□□□□□□□□□ try □□□□□□□□□ catch □□□□□ catch □□□□□□□□□□□ catch□ □□□□□□□□□□□□□□□□□——□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□ catch □□□□□□□□□□□□□□□□□□□□ catch □□□□□□□ □□□□□□□□□□□□□□□□□ catch □□□□□□

8. □□□□□□□□□□□□□□□□

- exception □□□□□□□□□□□□□□□□□□□ exception□ □□□□□□□□□□□□□□□□□□□□□□
- stdexcept □□□□□□□□□□□□□□□
- new □□□□□□ bad_alloc □□□□□□□□□□□□□□□□ new □□□□□□
- type_info □□□□□□ bad_cast □□□□

第七章 函数

1. □□□ □□□□□□□□□□

2. □□□ static□ □□□□□□ static □□

3. □□□□□□ inline □□□ □□□□□□□□□□□□□□□□□□“□□□”□□ inline □□□□ shorterString □□□□□□□□□□

```
inline const string &
shorterString(const string &s1, const string &s2)
{
return s1.size() < s2.size() ? s1 : s2;
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

4. □□□□□□□□□□□□□□□□□□□□□□□□□

5. □□□□□ const □□□□□□□□□□□□□□□□□□□□□□□□□□□□ const□ □ C++□ □□□□□□□ const □□□□□□□□□□ const □□□□□□ const □□□□□□□□□□□□□□□□□□□□□□□□□□□□

6. □□□□□□□□ const □□□□□□□□□□□□□□□□□□□□□ const□

```
double Sales_item::avg_price() const
{
    if (units_sold)
        return revenue/units_sold;
    else
        return 0;
}
```

7. □□□

```
Sales_item(): units_sold(0), revenue(0.0) { }
```

8. □□□□□□□□ const □□□□□□□□□□□□□□□□□ const □□□□□□ □□□□ const □□□□□□□□□□□□

9. □□□□□□□□□□□□□□□□□□□□□□□ typedef □□□□□□□□□□□ □□□□□□□□□□□□□□

```
typedef bool (*cmpFcn)(const string &, const string &);
```

□□□□□□□□□□□□□□□□□□□□ cmpFcn □□□□□□□□□□□□□ □□□□□□□□

第八章 标准 IO 库

1. IO □□□□□□□□□□ □IO□□□

2. □□□□□□□□□□□□□□□□□ vector □□□□□□□□ □□□□□□□□ □□□ vector □□□□□□□□□□□□□□□□□□□□□□

3. IO □□□□□□□□□□□□□condition state□ □□□□□□□□□□ IO □□□□ □□□□□□□□□□□□□□□□□□□□ badbit □□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□ failbit □□□ □□□□□□ failbit □□□□□□□□□□□□ eofbit □□□□□□□□□□□□□□ □□□□□□□□ failbit□

4. □□ IO □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ endl □□□□□□□□□□□□□□□□□□□□□□□□□ C++ □□□□□□□ □□□□□□□□□□□□□□□□□□ flush□ □□□□□□□□□□□□□□□ □□□□□□□□□□□□□□ ends□ □□□□□□□□□□□□□□□ null□ □□ □□□□□

```
cout << "hi!" << flush;
cout << "hi!" << ends;
cout << "hi!" << endl;
```

□□□□□□□□□□□□□□□ unitbuf □□□□□□□□□□□□□□□□□□ □□□□□□

```
cout << unitbuf << "first" << " second" << nounitbuf;
```

5. fstream □□□□□□□□□□□□ IO □□□□
- ifstream□ □ istream □□□□□□□□□□□□□□
- ofstream□ □ ostream □□□□□□□□□□□□□□
- fstream□ □ iostream □□□□□□□□□□□□□□□□□□

6. fstream □□□□□□□□□□□□□□□□□□□□□□□□□——open □ close□ □□□□□□□□□□□□□□□□□□ fstream□ ifstream □ ofstream □□□□□□ □□□□□□□□ IO □□□□□□□□

```
ifstream infile(ifile.c_str());
ofstream outfile(ofile.c_str());
ifstream infile;
ofstream outfile;
```

7. □□□□□□□□□□□□□□□□
- istringstream□ □ istream □□□□□□□□ string □□□□
- ostringstream□ □ ostream □□□□□□□□ string □□□□
- stringstream□ □ iostream □□□□□□□□□ string □□□□

第九章 顺序容器

1. □□□□□□□□□□□□□□□ vector□ list □ deque□ □□□□□ “double-ended queue”□ □□□□□□ “deck”□ □ □ v□ l□ d□
2. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□

```
vector<string> svec;
list<int> ilist;
deque<Sales_item> items;
```

3. □□□□□□□□□□□□□□□□□ □□□□□□□
- □□□□□□□□□□□□□
- □□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ IO □□□□□ □□□□□□□□□□□□□□□□ IO □□□□□□□□

4. □□□□□□□□□□□□□□□□□□□□□□□ □□□□□

```
vector< vector<string> > lines;
vector< vector<string>> lines;
```

□□□□□□□□□□□□ > □□□□□□□□□□□□□□□□□□□□□□□□ > > □□□□□□□□□□□□□□□□□□□□□□

5. □□□ `vector` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□vector □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□

6. vector □□□□□□□□□□□capacity □ reserve □□□□□□ vector □□□
□□□□□□□□□□□□□capacity□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□ □ reserve□□□□□ vector □□□□□□□□□□□□
□□□□□

7. □□□□□□□□□□□□□□□□□□□□□□□□queue□□□□□□□□□□
priority_queue□□□□□□ □ stack□ □□□□□□□□□□□□ deque□□□
□□□□

第十章 关联容器

1. □□□□□Associative containers□ □□□□□□□□□□□□□□□□□□□□□
□□□□□□□□ map□ set□ map□□□□□□□□key-value□□□□□□□□□□
□□□□□ map□□□□□□□□□□□□□□□□□□□□set□□□□□□□□
□□□□□□□□□□□□□□□□□□□map□□set□□

2. □□□□□ pair□□□□□□□□□□□□□□pair□□□□□□□□□□□□□□
□□□□□□□□□□□ pair□□□□□□□□□□□□□□pair□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
pair<string, string> anon;
pair<string, int> word_count;
pair<string, vector<int> > line;
```

pair□□□□□□□□□□□□□□□□□□□□□□□□ pair□□□□□□□□□
□ typedef□□□□□□

```
typedef pair<string, string> Author;
Author proust("Marcel", "Proust");
Author joyce("James", "Joyce");
```

3. □□□□□□□□□——□□□□□——□□□□□□□□□□□□□□□ front□
push_front□ pop_front□ back□ push_back□□ pop_back□□□

4. □□□□□□□□□□□□□

5. map□□□□□□□□□□□□□associative array□ □□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□ map□□□□□□□□□□□□□□□□
□ value type□ □

```
map<string, int> word_count;
```

□□□□□□□□□□□ <□□□□□□□□key type□□□□□□□□□□□□□
□□□□□□□□□ <□□□□□□□□□□□□□□□□□□□□□□□□□□□

6. map□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

map<K, V>::key_type	在 map 容器中，用做索引的键的类型
map<K, V>::mapped_type	在 map 容器中，键所关联的值的类型
map<K, V>::value_type	一个 pair 类型，它的 first 元素具有 const map<K, V>::key_type 类型，而 second 元素则为 map<K, V>::mapped_type 类型

□□□ map □□□□□□□□ value_type □ pair □□□□□□□□□□□□□□□□□□□□□□

7. map □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ set □□□□□□□□□□□□□□□□□□□□□□□□□□□□ set □□□□□□□□□□□□□□□□□ set □□□□□□□□ map □□□□□□□□□□ set □□□□□□□□□□□□□□□ mapped_type □□□

8. map □ set □□□□□□□□□□□□□□□□□ multiset □ multimap □□□□□□□□□□□□□□□ multimap □ multiset □□□□□□□□□ map □ set □□□□□□□□□□□□□ multimap □□□□□□□□

第十一章 泛型算法

1. □□□“□□”□□□□□□□□□□□□□□□□□□□□□□□

2. □□□□ accumulate □□□□□□□□

3. find_first_of □□ end □□□□

4. □□□□□□□□□□□□□□□ fill □□□ fill_n □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

5. □□□

6. back_inserter □□□□□□□□□□ back_inserter □□ push_back □□□□□□□□□□□□□□□□□□□ back_inserter □□□□□□□□ fill_n □□□□□□□□□

```
vector<int> vec;                     // □□□
fill_n (back_inserter(vec), 10, 0);  //□□ 10 □□□□ vec
```

7. C++ □□□□□□□□□□□□□□ □□□□□□□

- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- iostream □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ IO □□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□ □ reverse_iterator □ □ □ □ rbegin □ rend □ □ □ □ □ □ □

8. C++ □

- back_inserter□ □ □ □ □ push_back □ □ □ □ □ □ □ □ □
- front_inserter□ □ □ push_front □ □ □ □ □
- inserter□ □ □ insert □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ inserter □

9. □ □ iostream □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ iostream □ □ □ □ □ □ □ □ □ □ istream_iterator □ □ □ □ □ □ □ □ □ ostream_iterator □

10. □ istream □ □ □ □ □ □ □ □ □ □ □ □ □ □ ostream □ □ □ □ □ □ □ □ □ □ □ □

11. □ >> □ □ □ □ □ □ □ □ □ □ □ □ istream_iterator□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ << □ □ □ □ □ □ □ □ □ □ □ ostream_iterator□ □

```
istream_iterator<int> cin_it(cin);
istream_iterator<int> end_of_stream;
ofstream outfile;
ostream_iterator<Sales_item> output(outfile, " ");
```

12. □ ++ □ □ □ □ □ □ □ □ □ □ □ □ □ – □ □ □ □ □ □ □ □ □ □ □

13. □ □ □ □ □ -- □ □ □ ++ □

第十二章 类

1. □ 7.7.3 □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

```
Sales_item(): units_sold(0), revenue(0.0) { }
```

2. □ □ □ □ □ □ □ □ □ □ □ □ inline□

3. □ □ □ □ □ const □

```
double avg_price() const;
```

const □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ const □

4. □ □ □ □ □ □ □ □ □ □ □ □ □

```
class Screen;
```

□ Screen□ □ □ □ □ □ □ □ □ □ □ □ □ Screen □ □ □ □ □ □ □ □ □ □ □ □ Screen □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

5. □

6. □

□ □ □ □ □ □ □ □ □

```
class Sales_item { /* ... */ };
class Sales_item { /* ... */ } accum, trans;
```

7. □ □ □ □ □ const □ □ □ □ □ □ this □ □ □ □ □ □ □ □ □ □ □ □ const □ □ □ □ □ □ □ this □ □ □ □ □ □ □ □ □ □ □ □ this □ □ □ □ □ □ □ □ const □ □ □ □ □ □ this □ □ □ □ □ □ □ □ const □ □ □ □ □ □ const □ □ □ □ □ □ □ □ this □ □ □ □ □ □ □ □ □ □ □ □ this □ □ □ □ □ □ □
8. mutable □ □ □ □ □ □ □ □ □ const □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ const □ mutable □ □ □ □

```
class Screen {
public:
private:
      mutable size_t access_ctr;
};
```

9. □

```
class Screen {
public:
      typedef std::string::size_type index;
      index get_cursor() const;
};

inline Screen::index Screen::get_cursor() const
{
      return cursor;
}
```

10. □ □ □ □ □ □ □ □ □ const□
11. □
12. □ □ □ □ □ □ □ □ □ □ □ □ explicit□ □

```
class Sales_item {
public:
      explicit Sales_item(const std::string &book = ""):
            isbn(book), units_sold(0), revenue(0.0) { }
      explicit Sales_item(std::istream &is);
};
```

13. □ friend □

```
class Screen {
      friend class Window_Mgr;
};
```

□ □ □ Window_Mgr □ □ □ □ □ □ □ □ □ Screen □ □ □ □ □ □

14. □

```
class Screen {
      friend Window_Mgr&                              //Window_Mgr □ □ □ □ □
            Window_Mgr::relocate(Window_Mgr::index,
                  Window_Mgr::index,
                        Screen&);
};
```

15. □

16. □ □ □ □ □ □ □ □ □ □ static □ static □

17. □ □ □ □ □ □ □ □ □ □ □ static □ □ □ □ □ static□ static □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

```
class Account {
public:
      void applyint() { amount += amount * interestRate; }
      static double rate() { return interestRate; }
      static void rate(double); // sets a new rate
private:
      std::string owner;
      double amount;
      static double interestRate;
      static double initRate();
};
```

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ static □

```
rate = Account::rate();
```

18. □ □ □ □ □ □ □ □ □ □ static □ □ □ □ □ □ □ □ □ □ static □ static □ static □ □ □ □ □ □ this □ □ □ □ static □ □

19. static □ static □ const static □ static □ □ □ □ □

```
class Account {
public:
      static double rate() { return interestRate; }
      static void rate(double);
private:
```

```
    static const int period = 30; // interest posted every 30 days
    double daily_tbl[period]; // ok: period is constant expression
};
```

20. static□□□□□□□□□□□□□□□□□□□□□□□□□□

第十三章 复制控制

1. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□const□□
2. □□□□□□□□□□□□□□□□□□□□□□□□□□const□□

- □□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□
- □□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□

3. □□
4. □□□□□□□□□□□□□□□□□□□□□□□□□
5. □□□□□□□□□□□□□□□□□□□□□

```
Invoked vector<string> svec(5);
```

□□□□□□□string□□□□□□□□□□□□□□□□□svec□□□□□□□□□□□□□□□□□svec□□□□□□

6. □□□

```
Sales_item primer_eds[] = { string("0-201-16487-6"),
                            string("0-201-54848-8"),
                            string("0-201-82470-1"),
                            Sales_item()
};
```

7. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□
8. □□□□□□□□□□□□□□□□□□□□□□private□ □□□□private□
9. □□□□□□□□□□ □□□□□□□□□□□□□□□□ □□□□□□□□□□
10. □□□□□□□□□□□□□□□operator□□□□□□□□□□□□□□□□□□□□□□□□operator=□□□□□□□□□□□□□□□□

11. □ this □ □ □

12. □

```
class Sales_item {
public:
    Sales_item& operator=(const Sales_item &);
};
```

13. □

14. □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

```
Sales_item *p = new Sales_item;
{
    Sales_item item(*p);
    delete p;
}
```

□ □ □ □ □ □ □ □ □ □ □ □ □ □

15. □

16. □ static □

17. □

18. □ □ □ C++ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

- □
- □ □ □ □ □ □ □ □ “□ □ □ □”□ auto_ptr □
- □

19. □ 0 □

20. □ 1□ 0□ 0□ □ □ □ □ □ □ □ □

第十四章 重载操作符与转换

1. □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ operator □

```
Sales_item operator+(const Sales_item&, const Sales_item&);
```

2. □

```
int operator+(int, int);
```

3. □ && □ ||□ &&□ || □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
4. □
5. □ 1□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ this □
6. □
7. □
8. □

- □ □ □ =□ □ □ □ □ []□ □ □ □ □ ()□ □ □ □ □ □ □ □ □ ->□ □
- □
- □
- □

9. □ □ □ IO □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ostream& □ □ □ □ □ □ □ □ □ □ □ □ const □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ ostream □ □ □ □ □ □

```
ostream&
operator<<(ostream& out, const Sales_item& s)
{
    out << s.isbn << "\t" << s.units_sold << "\t"
        << s.revenue << "\t" << s.avg_price();
    return out;
}
```

10. □ □ □ □ □ □ □ □ iostream □
11. □ const □ □ □ □ □ □ □ □ □ □ const□ □

```
istream&
operator>>(istream& in, Sales_item& s)
{
    double price;
```

```
    in >> s.isbn >> s.units_sold >> price;
    if (in)
        s.revenue = s.units_sold * price;
    else
        s = Sales_item();
    return in;
}
```

□□

12. □□□□□□□□□□□□□□□□□□□□□□□□□□□□ operator[]□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □ const □□ const □□□□□□□□□□□□□□□□ const □□□□□□□□□ const □□□□□□□□□□□□□□□

13. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□ operator □□□□□□□□□□□□

```
class SmallInt {
public:
    SmallInt(int i = 0): val(i)
    {
        if (i < 0 || i > 255)
        throw std::out_of_range("Bad SmallInt initializer");
        }
    operator int() const { return val; }
private:
    std::size_t val;
};
```

□□□□□□□□□□□□□

```
operator type();
```

14. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

15. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□

16. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□

17. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□

```
void manip(const SmallInt &);
double d; int i; long l;
manip(d);                         // ok
manip(i);                         // ok
manip(l);                         // □□□□□ long □ double □□ long □ int □□□
```

18. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
void compute(int);
```

```
void compute(double);
SmallInt si;
compute(static_cast<int>(si));
```

第十五章 面向对象编程

1. □□□□□□□□□□□□ static □□
2. □□□□□□□□□□□□□□□□□□□□□□□□□□
3. □□□
4. □□□
5. public □□□□□□□□□□□□□□□□□□□□□□□□□ private □ protected □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
6. □□ using □□□□□□□

```
class Derived : private Base {
public:
      using Base::size;
protected:
      using Base::n;
};
```

7. □□□□□□□□ private □ protected □□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□
8. □□□□□□ static □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ static □□□□□□□□□ static □□□□□□□□□□□□□□□□□□□□ private□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ static □□□□□□□□□□□□□ static □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
9. □□
10. □□□□□□□□□□□□□□□□□□□□□□□□
11. □□□
12. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
class Bulk_item : public Item_base {
public:
      Bulk_item(const std::string& book, double sales_price,
            std::size_t qty = 0, double disc_rate = 0.0):
            Item_base(book, sales_price),
            min_qty(qty), discount(disc_rate) { }
};
```

13. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□

14. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□

15. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□

```
class Item_base {
public:
	virtual ~Item_base() { }
};
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□
□□□□□□□□□□□□□□□□□□□

16. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

17. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□

18. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□

19. □□□□□□□□□□ = 0 □□□□□□□□

```
class Disc_item : public Item_base {
public:
	double net_price(std::size_t) const = 0;
};
```

20. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□

21. □□□□□□□□□□□□□□□□□□□□□□□□□

22. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

第十六章 模板和泛型编程

1. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□

2. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

3. □□□□□□□□ template □□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
template <typename T>
int compare(const T &v1, const T &v2)
{
```

```
    if (v1 < v2) return -1;
    if (v2 < v1) return 1;
    return 0;
}
```

4. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

5. □□□□□□□□□ class □ typename □□□□□□□□ class T □□□ T □□□□ □□□□□ class □ typename □□□□□

6. □□□□□□□□□□□□□□□□□□□□□ inline□□□□□□□□□□□□ □□□□□□□□□□□□□□□□ template □□□

7. □□□□□□□□□□□□□□□□□□□□□□

8. □□□□□□□□□□□□□□□□ template □□□□□□□□□□□ Queue □□□ □□□□□ Type □□□□□□□□

```
template <class Type> class Queue {
public:
    Queue ();
    Type &front ();
    const Type &front () const;
    void push (const Type &);
    void pop();
    bool empty() const;
private:
// ...
};
```

9. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
Queue<int> qi;
Queue< vector<double> > qc;
Queue<string> qs;
```

10. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□

```
template <class T> int compare(const T&, const T&) ;
```

11. □□□□□□□□ class □ typename □□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□

12. □□□□□□□□□□□□□□□□□□□□□□□□ int □□□□□□□□□□□□ □□□□□□□□□□□□□□

13. □□□□□□□□□□□□□□□□□□□□□□□□□□

14. □□□□□□□□□□□□□□

- □□□□□□ const □□□
- □□□□□□□□□ < □□□

15. □□□□□□□□□□□□□□□□□□□□

16. □

```
Queue<int> qi;
Queue<string> qs;
```

17. □

```
int main()
{
    compare(1, 0);
    compare(3.14, 2.7);
    return 0;
}
```

18. export □

19. □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

- □ □ □ □ □ □ template □ □ □ □ □ □ □ □ □ □ □ □ □
- □ □ □ □ □ □ □ □ □ □ □ □ □
- □ □ □ □ □ □ □ □ □ □ □ □

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ Queue □ □ □ □ □ □ □ □ □ □ □ □ □

```
template <class T> ret-type Queue<T>::member-name
```

20. □

- □

```
template <class Type> class Bar {
    friend class FooBar;
    friend void fcn();
};
```

- □

```
template <class Type> class Bar {
    template <class T> friend class Foo1;
    template <class T> friend void templ_fcn1(const T&);
};
```

- □

```
template <class T> class Foo2;
template <class T> void templ_fcn2(const T&);
template <class Type> class Bar {
    friend class Foo2<char*>;
    friend void templ_fcn2<char*>(char* const &);
```

```
};
```

21. □□

```
template <class Type> class Queue {
public:
      template <class It>
      Queue(It beg, It end):
            head(0), tail(0) { copy_elems(beg, end); }
      template <class Iter> void assign(Iter, Iter);
private:
      template <class Iter> void copy_elems(Iter, Iter);
};
```

22. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

23. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
template <class T> template <class Iter>
void Queue<T>::assign(Iter beg, Iter end)
{
      destroy();
      copy_elems(beg, end);
}
```

24. □□□□□□□□□□□□□□□ static □□□

```
template <class T> class Foo {
public:
      static std::size_t count() { return ctr; }
private:
      static std::size_t ctr;
};
```

25. □□□□□□□ static □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ static □□□□□□□ □□□□□□□□□□□□□□□□□

```
template <class T>
size_t Foo<T>::ctr = 0;
```

26. □□□□□template specialization□□□

- □□□ template □□□□□□□□□□□<>□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□
- □□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□template□

27. □□□□□□□

```
template <>
int compare<const char*>(const char* const &v1,
                         const char* const &v2)
{
    return strcmp(v1, v2);
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
template <typename T>
int compare(const T &v1, const T &v2)
{
    if (v1 < v2) return -1;
    if (v2 < v1) return 1;
    return 0;
}
```

□□

27. □□□

```
template<>
int compare<const char*>(const char* const&,
                         const char* const&);
```

28. □□□□□□□

```
template<> class Queue<const char*> {
public:
    void push(const char*);
    void pop() {real_queue.pop();}
    bool empty() const {return real_queue.empty();}
    std::string front() {return real_queue.front();}
    const std::string &front() const
        {return real_queue.front();}
private:
    Queue<std::string> real_queue;
};
```

29. □□□□□□□□□□□□□□□□□□□ template<> □□□

30. □□□

```
template <class T1, class T2>
class some_template {
};
template <class T1>
class some_template<T1, int> {
};
```

第十七章 用于大型程序的工具

1. auto_ptr□□□□□□□

2. auto_ptr□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □ auto_ptr□□□□□□□□□□□□□□□□□□□□□□□□

3. □□□□ auto_ptr□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□

```
void f()
{
    auto_ptr<int> ap(new int(42));
}
```

4. auto_ptr□□□□□□□□□□□□□□□

5. □□□□□□□□□□ auto_ptr□□□□□□□ new□□□□□□□□□□□□

```
auto_ptr<int> pi(new int(1024));
```

□□□□□ pi□□□□□ new□□□□□□□□□□□□□□ new□□□□□□□ □□□ 1024□

6. auto_ptr□□□□□□□□□□□*□□□□□□□□->□□□□□□□□□ auto_ptr □□□□□□□□□□□□□□□□□□□□□□□□□□□□ auto_ptr□□□ auto_ptr□□□□□□□□□□□□□ auto_ptr□□□□□□□□□□□□□□□□ □□□□□

7. □□□ auto_ptr□□□□□ auto_ptr□□□□□□□□□□□ auto_ptr□□□□□□ □□□□□□□□□□□□□□□□ auto_ptr□□□□□□□□□□□□□□□□ auto_ptr□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□

8. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□

9. □□□□□□□□□auto_ptr□□□□□□□□□□□□□□□

```
auto_ptr<int> p_auto;
```

10. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 0□ □□□□□□□□□ auto_ptr□□□

```
if (p_auto.get())
    *p_auto = 1024;
```

11. auto_ptr□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □ auto_ptr□□□

```
p_auto = new int(1024);              // error
```

□□□□□□□ reset□□□□□□□□

```
if (p_auto.get())
    *p_auto = 1024;
else
    p_auto.reset(new int(1024));
```

12. □ □ □ □ □ □ auto_ptr □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

- □ □ □ □ auto_ptr □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
- □ □ □ □ □ □ □ □ auto_ptrs □ □ □ □ □ □ □ □ □
- □ □ □ □ auto_ptr □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
- □ □ □ auto_ptr □ □ □ □ □ □ □ □ □

13. □ □ □ □ □ □ □ □ □ □ namespace □ □ □ □ □ □ □ □ □ □ □ □ □

```
namespace cplusplus_primer {
	class Sales_item { /* ... */};
	Sales_item operator+(const Sales_item&,
	                     const Sales_item&);
	class Query {
	public:
		Query(const std::string&);
		std::ostream &display(std::ostream&) const;
	};
	class Query_base { /* ... */};
}
```

14. □

15. □

16. □ □ std □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ using □

```
using cplusplus_primer::Query;
```

17. □ □ □ □ □ □ □ □ □ □ □ □

18. □ namespace □ □ □ □ □ □ □ □ namespace □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

```
#include <iostream>
#include <memory>
using  namespace std;

namespace {
int i=0;
}

int main() {
cout << i << endl;
}
```

19. □

20. □□□

21. □□□

```
class Panda : public Bear, public Endangered {
};
```

22. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□

23. □□□

24. □□□ Bear □□ Endangered □□□□□□□ print □□□□ □□ Panda □□□□ □□□□□□

```
ying_yang.print(cout);
```

□□□□□□□□□□□□□□Panda □□□□□□□□□□□□□ print □□□□□ □□□□□□□□□□□□□□□□□□□□□□□ Panda □□□□ print□□□□□□□□ □□□□

25. □□□□□□□ print □□

26. □□□□□□□□□□□□□□□□□

```
ying_yang.Endangered::print(cout);
```

27. □□□

28. □□□□□□□□□□□□□ virtual □□□□□□

```
class istream : public virtual ios { ... };
class ostream : virtual public ios { ... };
```

29. □□□□□□ virtual □□□ □□□□□□□□□□□□□□□□□□□□□□□

30. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

31. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

32. □□□ □□

第十八章 特殊工具与技术

1. C++ □□□□□□□□□□□□□□□□□□□□□□

- allocator □
- □ □ □ □ □ operator new □ operator delete□ □

2. allocator □
3. allocator □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ allocator □ allocator □ □ □ □ □ □ □ construct □ destroy □ □ □ □ □ □ □ □ □ □ □
4. □ □ □ operator □ □ □ □ operator=□ □ □ □ operator new □ operator delete □ □ □ □ □ □ new □ delete □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ new □ delete □ □ □ □ □ □ □
5. □ □ □ □ □ □ □ □ □ □ RTTI□ □
6. □ □ □ □ □ □ □ □ □ □ □ RTTI□

- typeid □
- dynamic_cast □

7. □
8. □ □ □ □ dynamic_cast □ dynamic_cast □ □ □ □ □ □ □ □ □ □ □ □ □ ——□ □ □ □ 0 □ □ □ □ □ □ □ □ □
9. □ □ □ □ □ □ □ □ □ □ □ □ dynamic_cast □ dynamic_cast □ □ □
10. typeid □ □ □ □ □ □

```
typeid(e)
```

□ □ e □ □ □ □ □ □ □ □ □ □ □ □ □

11. □ static □ □ □static □ static □ □ □static □ □ □ □ □ □ □ □ □ □
12. □

- □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ const□
- □ □ □ □ □
- □ □ □ □ □ □ □

□ □ □ □ □ □ □ □ □ get □ □ □ Screen □ □ □ □ □ □ □ □ □ □ □ □ □ □

```
char (Screen::*)() const
```

□ □ □ □ □ □ Screen □ □ const □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ char □ □ □ □ □ □ □ get □

```
char (Screen::*pmf)() const = &Screen::get;
```

□ □ □ □ □ □ □ □ □ □ get □ □ □ □ □ □ □ □ □ □

```
char (Screen::*pmf2)(Screen::index, Screen::index) const;
pmf2 = &Screen::get;
```

13. □ □ □ □ □ □ □ □ □ □ . □ ->□ .* □ -> □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

□ □ □ □ □ □ □ □ □ □

- □ □ □ □ □ □ □ □ □ □ □ .*□ □ □ □ □ □ □ □ □ □ □ □
- □ □ □ □ □ □ □ □ □ □ ->*□ □ □ □ □ □ □ □ □ □ □ □ □

14. □

15. □ □ □ □ □ □ □ □ □ □ □ □ union □ union □ union □

16. □

```
union TokenValue {
    char cval;
    int ival;
    double dval;
};
```

17. □ □ □ □ □ □ □ □ □ union □ □ □ □ □ public □ □ □ union □ union □

18. □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ union □ 12.4.5 □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ union □ first_token □ □ □ □ □ □ □ cval □ □ □ □ □ □

```
TokenValue first_token = {'a'};
TokenValue last_token;
TokenValue *pt = new TokenValue;
```

19. □ □ □ □ □ □ □ □ □ □ □ □ □ □ . □ ->□ □ □ union □ □ □ □ □ □ □ □

```
last_token.cval = 'z';
pt->ival = 42;
```

20. □ static □ □ □ □ □ □ □ □ □ □ □ □ □ □

21. ::□ ::bind □ □ □ □ □ □ □ □ □ □ □ □ □ bind □ □ □

22. □ static □

```
int a, val;
void foo(int val)
{
    static int si;
    enum Loc { a = 1024, b };
    class Bar {
    public:
```

```
        Loc locVal;                  // ok: uses local type name
        int barVal;
        void fooBar(Loc l = a)       // ok: default argument is Loc::a
        {
            barVal = val;            // error: val is local to foo
            barVal = ::val;          // ok: uses global object
            barVal = si;             // ok: uses static local object
            locVal = b;              // ok: uses enumerator
        }
    };

}
```

23. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□

```
typedef unsigned int Bit;
class File {
    Bit mode: 2;
    Bit modified: 1;
    Bit prot_owner: 3;
    Bit prot_group: 3;
    Bit prot_world: 3;
    // ...
};
```

24. □□□□□□□□□□□□□□□□□□□□

```
void File::write()
{
    modified = 1;
    // ...
}
void File::close()
{
    if (modified)                    // modified □□ 22 □□□□
    // ... save contents
}
```

25. □□□□□□□□□□□□□□□□□□□□□

```
enum { READ = 01, WRITE = 02 }; // File modes
int main() {
    File myFile;
    myFile.mode |= READ;            // set the READ bit
    if (myFile.mode & READ)         // if the READ bit is on
        cout << "myFile.mode READ is set\n";
}
```

26. □□□ volatile □□□□□□□□□□□□□□□□□□□□□□□□□

27. □□ const □ volatile □□□□□□□□□□□□□□□□□□□□□□□□□
volatile □□□□□□□□□□□

28. □□□□□□□□□ volatile □□□□□□□□□□□□ volatile □□□□□ volatile □□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□

```
class Foo {
public:
      Foo(const volatile Foo&);
      Foo& operator=(volatile const Foo&);
      Foo& operator=(volatile const Foo&) volatile;            // volatile □□□□
};
```

29. □□□□□ extern "C"

30. C++ □□□□□□□□□□□ C++ □□□□□□□□

31. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□

```
extern "C" size_t strlen(const char *);
extern "C" {
      int strcmp(const char*, const char*);
      char *strcat(char*, const char*);
}
```

32. □□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□ C++ □□□

```
extern "C" double calc(double dparm) { /* ... */ }
```

33. □□□□□□□□□□□□□□□□□□□□extern "Ada"□ extern "FORTRAN" □□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□ C □□□□□□□□□□□

34. C++ □□□□□□□□□□ C□