标准类型库和string、vector

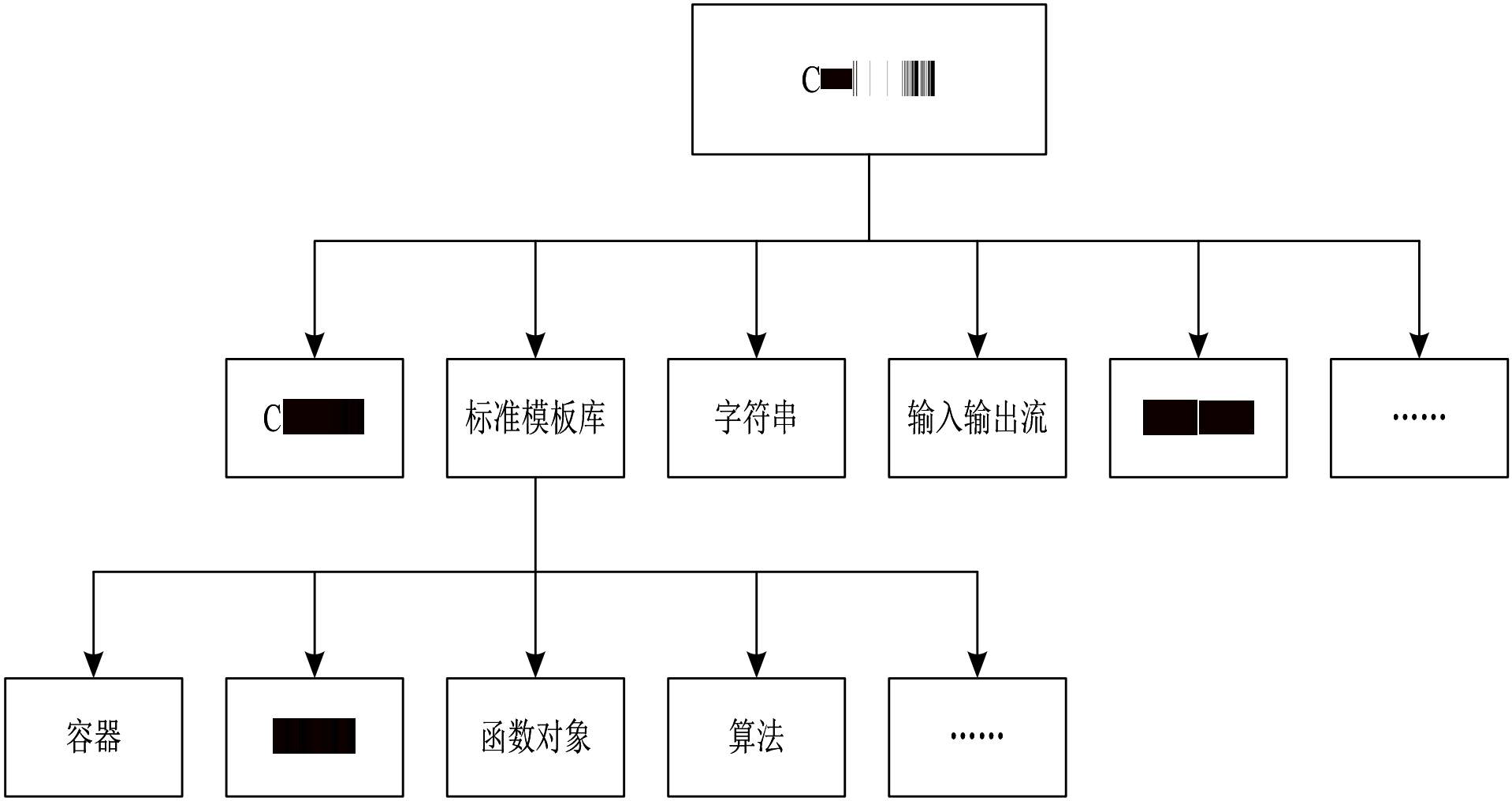
- 标准库的组织结构
- 命名空间
- 标准库的string类型
- 标准库的vector类型

概念和特点

- C++ Standard Library is a collection of classes and functions, which are written in the core language and part of the C++ ISO Standard itself
- Features of the C++ Standard Library are declared within the std namespace

标准库的组织结构

- C++标准库由多个组件构成。按照它们的功能，可以进行如下分类：



C标准库

- <cassert>
- <cctype>
- <cerrno>
- <cfloat>
- <climits>
- <cmath>
- <csetjmp>
- <csignal>
- <cstdlib>
- <cstddef>
- <cstdarg>
- <cstdio>
- <cstring>
- <ctime>
- <cwchar>
- <cwctype>

1、无“.h”作为后缀；
2、以字符“c”作为前缀；
3、属于命名空间“std”

- 标准模板库（STL，Standard Template Library）：这是C++标准库的核心部分，提供了很多标准类型的容器以及操作这些容器的算法。整个STL由容器、迭代器、函数对象和算法等组件构成。
- 字符串（string）：字符串类型的对象，提供了很多对字符串的操作，如字符串的读写、连接等。
- 输入/输出流（I/O Stream）：对输入/输出流进行操作。
- 异常处理：可以定义错误和异常并提供操作的方法。

标准库的string类型

- C++ 语言定义了几种基本类型：字符型、整型、浮点型等。
- C++ 还提供了可用于自定义数据类型的机制，标准库正是利用这些机制定义了许多更复杂的类型，比如string、vector 等。
- string 类型支持长度可变的字符串，vector 可用于保存一组指定类型的对象

标准库的string类型

- 基本数据类型：这些类型表示数值或字符的抽象，并根据其具体机器表示来定义。
- 抽象数据类型：
- 1、（高级）表示了更复杂的概念；
- 2、（抽象）我们在使用时不需要关心它们是如何表示的，只需知道这些抽象数据类型支持哪些操作就可以了。

标准库的string类型

- 标准库 string 类型的目的就是满足对字符串的一般应用。
- 用户程序要使用 string 类型对象，必须包含相关头文件。如果提供了合适的 using 声明，那么编写出来的程序将会变得简短些：

```
#include <string>
using namespace std;
```

string对象的定义和初始化

- 定义一个空的string对象。
  - 例如： string s1;
- 用一个字符串定义和初始化string对象。
  - 例如： string s2(“This is a C++ program!”);
- 用一个已有的string对象来定义另一个string对象。
  - 例如： string s3(s2);

string对象的赋值

- 可以把一个string对象赋值给另一个string对象，还可以把字符数组赋给string对象，也可以给string对象赋值一个字符。
- 例如：

```
string s1,s2,s3;
s1=”This is a string”;
s2=s1;
s3=’A’;
```

string对象的读写

- string对象可以同基本类型变量一样使用iostream标准库中的输入/输出操作符进行读写操作。
- 例如：

```
string s1;
cin>>s1;
cout<<s1;
```

string对象的读写

- 在读取字符串时，系统会以空格、回车、换行作为读取结束的标志。
- 对字符串对象使用getline函数可以完整的读取字符串。
- getline函数的格式：

  getline(输入流对象，string对象)
- getline函数将换行符作为读取结束的标志。

string对象的连接

- 连接string对象可以使用“+”和“+=”运算符。
- “+”运算符可以将一个string对象、一个字符串或是一个字符连接到一个string对象后面。

```
int main()
{
   string s1,s2;
   s1="C++";
   s2=s1+" program"+"\n";
   s1+="\n";
   cout<<s1;
   cout<<s2;
   return 0;
}
```

程序运行的结果：

```
C++

C++ program
```

string对象的比较

- string对象支持所有的关系运算符。
- string对象之间、string对象与字符数组之间都可以进行比较。
- 比较时，英文字符按照A~Z、a~z升序、大写字母大于小写字母的顺序比较，如果是汉字则按照拼音的a~z升序顺序比较。

```
string s1,s2;
s1="hand";
s2="head";
if(s1>s2)
  cout<<s1<<">"<<s2<<endl;
if(s2<"hear")
  cout<<s2<<"<"<<"hear"<<endl;
if(s2!="help")
  cout<<s2<<"!="<<"help"<<endl;
```

string对象的长度

- string对象的长度实际是指string对象中字符的个数。
  - 求string对象中字符的个数可以通过操作函数size()或length()实现。
- 如果要判断string对象中是否有字符，或者说判断string对象是否为空，那么可以通过操作函数empty()进行判断。
  - empty函数的返回值为布尔值，如果为空则返回true，否则返回false。

```
string s1;
s1="student";
cout<<s1.size()<<endl;
cout<<s1.length()<<endl;
if(!s1.empty())
  cout<<"Good"<<endl;
else
  cout<<"null"<<endl;
```

获取string对象中的字符

- 在string对象中可以使用下标操作符[ ]或操作函数at( )检索字符串中的某个字符。
- string对象的下标从0开始。

```
string s1("student");
cout<<s1[2]<<endl;
cout<<s1.at(5)<<endl;
```

获取string对象的子串

- 在string对象中可以使用substr()函数来取得一个子串。substr()函数的格式为：

  substr(m,n);

- 函数中的第一个参数m是指子串在原字符串中的起始位置，第二个参数n是指子串的长度。
- 例如：

```
string str1="Telephone";
string str2=str1.substr(0,4);
cout<<str2<<endl;
```

标准库的vector类型

- vector对象在一个线性列表中存储数据元素。其中的数据按照存储的先后顺序进行排列。
- vector对象是一个动态数组，同时vector允许程序员随时对其中的数据进行访问、增加、删除的操作。
- vector类型还提供了对序列中元素的随机访问。在运行时vector可以动态改变自身的大小以便容纳任何数目的元素。
- 如果在程序中要使用vector类型，需要包含头文件：

```
#include <vector>
```

vector对象的定义和初始化

- 定义一个空的vector对象。
  例如：vector<string> vec1;
- 定义vector的大小。
  例如：vector<int> vec2(10);
- 定义vector的大小，并且每个元素的初值为1。
  例如：vector<int> vec3(10，1);

vector对象的定义和初始化

- 使用已有的vector对象或数组创建新的vector对象。
  例如：

```
vector<int> vec4(vec3);
int a[5]={1,2,3,4,5};
vector<int> vec5(a,a+5);
vector<int> vec6(a+1,a+4);
```

- 在使用数组初始化vector对象时需要使用数组的首地址和要复制的最后一个元素的下一个地址来实现。

向vector中添加元素

- 向vetcor添加元素时需要使用push_back()函数，该函数的作用是将一个值作为一个新的元素添加到vector对象的后面。

  例如：

```
vector<int> vec1;        //创建一个空的vector对象
vec1.push_back(12);      //将12插入到vec1的末尾
vec1.push_back(35);      //将35插入到vec1的末尾
```

vector对象的长度

- vector对象的size( )函数可以返回元素的个数。
- empty( )函数可以判断vector对象是否为空，它的返回值为布尔值，如果为空返回true，否则返回false。

例如：

```
vector<int> vec1(10);        //定义vec2的元素个数为10
cout<<vec2.size()<<endl; //输出vec2中的元素个数
if(vec2.empty())
    cout<<"YES"<<endl;   //如果vec2为空则输出“YES”
```

访问vector中的元素

- 访问vector中的元素是通过元素所在的位置来进行访问的，可以使用下标操作符[]访问，也可以使用at( )函数。例如下面的两个语句是等价的，作用都是输出vec3中的元素值。

  例如：

```
for(i=0;i!=vec3.size( );++i)
   cout<<vec3.at(i)<<endl;
for(i=0;i!=vec3.size( );++i)
   cout<<vec3[i]<<endl;
```

向vector中插入元素

- 使用insert()函数可以在vector对象中的任意位置插入新的元素。

  例如：

  ```
  //在vec4开始的位置插入两个元素，值为'x'
  vec4.insert(vec4.begin( ),2,'x');
  //在vec4第3个元素后面插入两个元素，值为'x'
  vec4.insert(vec4.begin( )+3,2,'x');
  ```

- 在insert()函数中有三个参数：第一个参数为插入的位置，在上面的例子中可以使用begin()函数取得vector对象中的第一个元素的位置；第二个参数是指要插入的元素的个数，第三个参数是要插入的元素的值。

删除vector中的元素

- 使用pop_back()函数可以删除vector对象的最后一个元素。

  例如：

  ```
  vector<int> vec5(5,1);
  vec5.pop_back();
  ```

- 使用erase()函数可以删除任意位置的元素，并且返回下一个元素位置的值。

  例如：

  ```
  vec5.erase(vec5.begin( )+3);          //删除vec5中的第4个元素
  ```