07-04-Language

Created on 20190819.

Last modified on 2023 年 4 月 10 日.

目录

Chapter 1 Introduction

主要包括

Chapter 2　形式化方法（形式验证）

形式语义、代数规范、范畴论、类型论和重写技术等。

2.1　形式语义

Chapter 3　语义理论（自然语言、程序语言）

3.1　计算语义

Chapter 4　类型理论

Chapter 5　多值逻辑

Chapter 6　模糊逻辑

Chapter 7　推理技术

Chapter 8　编译原理

【词法分析】根据词法规则，对源程序逐个字符扫描，识别出一个个“单词”符号，词汇检查。【语法分析】根据语法规则将单词符号序列分解成各类语法单位，如“表达式”“语句”和“程序”等。语法规则就是各类语法单位的构成规则，主要针对结构的检查。【语义分析】分析各语法结构的含义，检查源程序是否包含语义错误，主要针对句子含义的检查。

语法分析和语义分析交替进行；完成一步语法分析，接着进行语义分析。

编译器后端：代码优化【目的：使机器码短小、质量高、访问存储单元次数少】、机器码生成。

Chapter 9　编程范式

9.1　并发编程

9.2　函数式编程

9.3　命令式编程

9.4　逻辑编程

9.5　面向对象编程

9.5.1　UML

united modeling language.

通信图中，：冒号前面是对象名，后面是类名，之间连线上是消息。

模块结构图的主要组成有：模块、调用、数据、控制信息和转接符号

9.5.2　Principles

Open for extension, closed for modification. 对扩展开放，对修改封闭。类的改动通过增加而不是修改代码实现。Single Responsibility. 类对外提供一种功能。模块只能有一种被修改的理由。Dependence inversion. 依赖接口。业务层与实现层都向协议层靠拢。interface segegation. 一个接口提供一种功能。Liskov substitution. 抽象类出现的地方都可以用其实现类替换。composite or aggregate reuse. 优先使用组合，因为继承时父类的修改会影响子类，对象组合会降低依赖关系。law of demeter 一个对象应对其他对象尽量少的了解。

9.5.3　设计模式

观察者模式：具体的观察者将自己注册到事件，具体的事件知道自己的观察者、体现类对扩展开放，对修改关闭。

9.5.4 Others

```
http://api.open-notify.org/astros.json

https://openweathermap.org/current
```

建议把 xy push 到局部 vector 中，把局部 vector push 到 vec 中

Notepad++ 里 Run 可以设置用.bat 来编译运行，bat 里面要先 cd 到其目录

penetration test, using attack tools.

进程因为创建而就绪，因为调度而运行，因为 IO 事件而阻塞，IO 事件完毕或时间片用完回到就绪状态，运行完成后消亡。

c++ 看 gcc-cp 目录

它是个编译器，也可以画图写别的 pythonjava 之类的

我都没有研究过，我需要找点资料好好学习一下，个人觉着 c 用起来更像是地址的别名，fortran 的更像是地址的别名的别名，两个的比较应该找一些开源的转换库看看，

```
 using namespace std;在局部使用挺好的
using FP = void (*) (int, const std::string&); //Replace using typedef
```

伸缩型数组，在 c 中比存指针成员的好处是 1）不占 1 个地址的存储空间，2）最适合制作动态 buffer，因为可以直接就把 buffer 的结构体和缓冲区一起分配。

```
struct MyStruct
{
    int   i;
    float  j;
    double  arr[0];  //伸缩型数组成员
    //double arr[];  //这样声明也可以
};
pf = malloc(sizeof(MyStruct) + N*sizeof(double));/*只分配一块内存*/
```

c# 生态还是局限

假如一种语言不允许运行期动态分配内存的话，确实只要有 stack 就够了，内存的数据区只划分成两大块，一块用于独立于过程的数据（编译、连接时已知大小），剩下的全部做为一个 stack。C、C++ 这样的允许动态分配内存的语言，当然还是要有一个 heap 的，这样内存就划分成三个区域了。在引入线程概念之前，确实如你所想的，除了运行时动态确定大小的数据必须放在 heap 里面，其他都可以放在 stack 里，我们可以把 stack 设置的足够大。然而，后来线程出现了。每个线程有自己独立的 stack，一个程序可以创建成百上千个线程，那么 stack 的大小也就只能是一个很小的值，内存才不会被用光。在 windows 系统里，每个线程的 stack 默认只有 1MB，LINUX 默认 8MB，这样，stack 就成了紧缺资源，必须非常有节制地使用，体量比较大的数据都要由程序员手工分配到 heap 里去并手工管理其生命期。

java 半编译半解释

反码：符号位不变，其余位翻转。正整数的补码：是原码；负整数的补码：是反码 +1；注意运算可能溢出。

```
  a = 3*5,a*4;//60
  typedef int MyINT;
  using MyINT = int;

  decltype(i) j = 2;//the type of j is the same as i.
initializer_list
```

每一个练习封装成一个函数收集在一个.hpp 里面，在一个 main 里面调用相应的练习函数

我认为核心还是数据，设计类的方法要围绕着数据来，公开出去的方法要维护好隐藏起来的成员变量

量子加密 RSA

原来 stl 的模板采用的那叫静态多态，编译期间翻译，所以模板的声明实现分离在头文件里实例一份就行了

Chapter 10　程序设计语言的设计与实现

Chapter 11　程序设计语言学习

程序 = 算法 + 数据 = 对象 + 事件。

11.1　CPP

11.1.1　Introduction

History

C++11　1）并行编程：（Native Concurrency），通过内存模型、线程、原子操作；2）泛型编程：统一初始化表达式、auto、decltype、移动语义；3）系统编程：constexpr、POD；4）支持库构建：内联命名空间、集成构造、右值引用。

Books

Bjarne Stroustrup 推荐要掌握的的 5 种语言：C++、Java、Python、Ruby 和 JavaScript

c++ primer v5, the author is the creator of the first c++ compiler.

the c++ programming language v4, the author is the creator of the c++.

the c++ standard library: a tutorial and reference.

代码规范

修改注释，例如 —[[修改的代码]]——

11.1.2　数据类型

基础数据类型

```
整形:
42, short, 2bytes, [-32768,32767]
42, int, 4bytes
42l, 42L, long,l, L, 4bytes
42ll, 42LL, long long, ll,LL
```

```
无符号整形:
unsigned short, 2bytes, [0,65535]
42u, unsigned int, 4bytes
42LU, 42uL, unsigned long, 4bytes


__int8 // 8位整形
__int16 // 16位整形
__int32 // 32位整形
__int64 // 64位整形

进制:
42 // 10
042 // 8
0xF8 or 0XF8 // 16

浮点:
1.2f, 1.2F, float,f,F, 4bytes
1.2, 1., double, 8bytes
1.2L, long double

科学计数法:
3e2
1.2E-2

字符型:
char; 'a'=97; 'A'=65, 1byte
unsigned char
L'a', wchar_t

'\n' 换行
'\r' 回车
'\t' 制表
'\v' 垂直制表
'\b' 退格
'\a' 蜂鸣
'\\' 反斜杠
'\'' 单引号
'\'''    双引号
'\ddd'    3位8进制
'\xdd'    2位16进制

bool
```

自定义数据类型

```
struct
```

```
class

union

enum; enum class
```

数组

长度固定且不可变。

```
一维数组:
int a[3]; // 3个int, 下标从0开始
int a[3] = {10,20}; //前2个数初始化
int a[3]{10,20,30};
int a[]{10,20,30};


int arr[][2] = {1,2,3,4,5,6}; // 最后一个维度可忽略。在使用的时
候左边的下标要尽量少变化, 以加快访问速度。

char name[] = {'J', 'a', 's', 'o', 'n'}; // length =5
char name[] = "Jason"; // length = 6


【字符串函数】
strcpy( str1, str2 ); // 把str2 复制到str1。需要自己保证str1 的长度足够容纳str2 的内容

strlen( 字符数组名 );//包括里面的空格, 但是不包括里面的字符串结束标志

strcat( str1, str2 ); // 连接字符串.字符数组2 中的字符串连接到字符数组1 中字符串的后面

if( strcmp( str1, str2 ) == 0 ) // 判断两个字符串是否相同


【指针】
int *p1, *p2, val; // pointer p1, p2; int val
```

内存分区

栈，堆，全局/静态存储区，常量存储区。

不同 cpp 中使用同一个变量，在全局变量前加上 extern。extern 关键字只是声明而不是定义一个全局变量。

全局静态变量的作用域仅限于定义这个变量的源文件，而不是整个程序。

free store VS heap

堆是操作系统维护的一块内存。自由存储是 C++ 中通过 new 与 delete 动态分配和释放对象的抽象概念。

Free Store The free store is one of the two dynamic memory areas, allocated/freed by new/delete. Object lifetime can be less than the time the storage is allocated; that is, free store objects can have memory allocated without being immediately initialized, and can be destroyed without the memory being immediately deallocated. During the period when the storage is allocated but outside the object's lifetime, the storage may be accessed and manipulated through a void* but none of the proto-object's nonstatic members or member functions may be accessed, have their addresses taken, or be otherwise manipulated.

Heap The heap is the other dynamic memory area, allocated/freed by malloc/free and their variants. Note that while the default global new and delete might be implemented in terms of malloc and free by a particular compiler, the heap is not the same as free store and memory allocated in one area cannot be safely deallocated in the other. Memory allocated from the heap can be used for objects of class type by placement-new construction and explicit destruction. If so used, the notes about free store object lifetime apply similarly here.

11.1.3 语句

语句和表达式

```
语句：以分号结束
语句块：{}
表达式：操作数+运算符
只有1个三目运算符：条件运算符，"?:"
逗号运算符：从左向右计算，取最右边部分的值

位运算：
~，按位非
<<，左移
^，按位异或
&，按位与
|，按位或
```

运算符优先级

```
1：LR
() 括号、函数调用
[] 数组
```

```
-> 指向成员
. 成员

2: RL
! 逻辑非
~ 取反
++
--
+ 正号
- 负号
* 指针
(type) 类型转换
sizeof 字节操作符

3: LR
* 乘法
/ 除法
\% 取模

4: LR
+
-

5: LR
>>
<<

6: LR
<, <=
>, >=

7: LR
==
!=

8: LR
&

9: LR
^

10: LR
|

11: LR
&&

12: LR
||
```

```
13: RL
?:

14: RL
=
+=,-=,*=,/=,\%=
&=,^=,|=
<<=,>>=

15: LR
, 逗号
```

类型转换

隐式类型转换：double 转为 int 舍去小数；char 转为 int 取对应的 ASCII 码。

控制流程

```
   描述方法:
伪码
流程图
UML(顺序图、活动图)


【分支】
if(expression) B;
if(expression) B else C;

switch(expression)
{
   case 常量1:
      A;
      break;
   case 常量2:
      B;
      break;

   default:
      C;
      break;
}


【循环】
while(expression) A; //先判断，再执行
while(expression) {...} //先判断，再执行
```

```
do A; while(expression); //先执行1次，再判断
do {...} while(expression); //先执行1次，再判断

for(int i=0;i<10;++i){std::cout << i << std::endl;}


【循环控制】
break; // 用于switch和循环，退出1层
continus; //用于循环，结束当前循环，进入下一次循环


【流程跳转】
AAAAA: // 标签
goto AAAAA; // 直接到标签处运行
```

11.1.4 细节

sizeof

程序存储分布有三个区域：栈、静态和动态。

sizeof 操作符，计算的是对象在栈上的投影体积。

不管指针指向的内容在什么地方，sizeof 得到的都是指针的栈大小

C++ 中对引用的处理比较特殊；sizeof 一个引用得到的结果是 sizeof 一个被引用的对象的大小。

```
   关于sizeof的两个精巧的宏实现。
非数组的sizeof:
#defne _sizeof(T) ( (size_t)((T*)0 + 1) )

数组的sizeof:
#define array_sizeof(T) ( (size_t)(&T+1) - (size_t)(&T) )
```

Reference

The reference is ofen used as parameters of a function. rreference is a const pointer

```
// const T& or const T cannot be passed to T&
 …
int var(1);
int &ref = var; //ref is always the second name of var.
…
int& func(const double& iVal,...){...}
func() = 3; // We can use the function just as a variable
```

Const

```
// const T* can be transformed to T* using (T*)
 …
int var(1);
const int* pCanNotModifyVar = &var; //const T* p CANNOT modify the variable it points to.

const int& num = 10;//ok
int& num = 10;//wrong
```

const object function can only be called by const object.

```
void testFuncAdd20210718_2()
{
    const int a = 10;
    int* pConstModifier = (int*)&a;// compiler finds and allocates memory
    const int *q = &a;
    {
        *pConstModifier = 20;
    }
    std::cout <<a <<std::endl;//10
    std::cout <<*pConstModifier <<std::endl;//20
    std::cout << (&a == pConstModifier ) <<std::endl;//1
    std::cout << (q == pConstModifier ) <<std::endl;//1
}
```

IO

dec,hex, oct

11.1.5 RTTI 与异常处理

错误：可预见，知道如何处理；

异常：不可预见，不知道如何处理。

RTTI，运行时类型信息，在类层次结构中漫游，可向上、向下、平行转换，可实现反射、高级调试等功能。

被抛出的变量称为异常对象，因为跨域，所以生命周期需要跨域。

异常处理：安全的 longjmp 机制。

```
void function(void) throw(something){//...}

throw "abcde";

class MyException1 : public exception{};
class MyException2{};
```

```
throw MyException1();

try{ // 可能产生异常的代码}
catch( type1 id1 ){ // 错误处理}
catch( type2 id2 ){ // 错误处理}
catch(...){ //捕获全部异常}



catch( int & n)
{
   n = 2;
   throw; // 重新抛出n, n=2
}
```

构造函数中抛出异常会产生不完整的类，永远不会调析构。解决方法：做一个有 T* 数据成员的模板类 A，我们创建的类 B 中的 int* pVal 要在 B 的构造中 new，现在改为存 A<int>pVal，在 B 的构造中 new，这样 new 委托给了类 A，在我们的类 B 的构造中 throw 也不会导致内存泄露。

栈展开

栈展开（stack unwinding），沿嵌套函数调用链继续向上，直至为异常找到一个 catch 子句。

抛出异常时，暂停当前函数的执行，开始查找匹配的 catch 子句。首先检查 throw 身是否在 try 块内部，如果是，检查与该 try 块相关的 catch 子句，看其中是否有一个与被抛出对象相匹配。如果找到匹配的 catch 子句，就处理异常；如果找不到，就退出当前函数（释放当前函数的内存并撤销局部对象），并且继续在调用函数中查找。

11.1.6 Function

完成形参的构造（如调用类的拷贝构造等）之后，再进入函数体内。

```
以下3个声明是等效的:
void function( int *p );
void function( int a[] );
void function( int a[100] ); // 数组长度没有意义，所以a[10]同a[100]是等效的
```

默认参数

函数定义中可以为最右边连续若干参数有省却值, 可以用于扩充函数参数时减少对原有代码的修改。

函数声明时没有声明形参，在函数定义时可声明形参；不同 cpp 中可对于同一个头文件中的函数声明不同的形参。如头文件中有 a,b,c 三个参数；可实现 a,b,c=0; a,b=0,c=0 两个版本的函数。

inline

We use Inline to tell Compiler to do copy-paste. The function is short and called many times. 调用时间少了，可执行文件体积增大

oveload

重载。Functions with the same name but the variables are different.We donot care the returned type.

函数地址

通常将函数 A 的地址传到函数 B 中，这样实现动态改变 B 的行为，称为回调。例如排序算法中，把元素关系比较函数传入比较函数。

```
void f(){}
f和&f都表示函数f的地址。

int (*pf)( int a, int b); // 定义函数指针变量pf
typedef int ( * pF ) ( int a, int b ); // 定义函数指针类型pF
```

11.1.7 Object

1) Do not set data member public. 2) Try the best to use reference as the io of functions. 3) Use const as most as possible. 4) use initial list in constructor. 结构化程序设计没有封装和隐藏的概念，导致数据结构与函数之间相互的关系、函数之间的调用关系不明确。把数据结构和对其进行的操作方法放在一起。

stack object, also called auto object. static object will be created once, it will not be delete until the program ends. global object, outside every , its scale is the whole program.

类的实现也可以直接在.h 中。

Basic

给出了构造函数，编译器不默认提供无参构造函数。复制构造函数，只有 1 个

```
C（const C& iC);
C();//临时对象，下一个语句就析构了。
```

返回值是类 A 的实例时，函数返回时调用 A 的复制构造函数

声明时可给出变量的初始值，后面可在构造函数中重新赋值。

解决菱形继承，在第一级继承时使用 virtual:public, 这样后面每次实例化时要调用虚基类的构造函数。

```
//类型转换构造函数
C（int iIn)
C c1 = 12, c2;
c2 = 9;//error
c2 = C(9); // ok
```

```
//When we new a class:
C* pC = new C(1,2);
// The compiler transformed into 3 steps:

void* mem = operator new(sizeof C);//Inside it calls malloc(n);
pC = static_cast<C*>(mem);
pC->C::C(1,2);

// When delete a class:
delete pC;
// It turns into:
C::~C(pC);//Clear the data.
operator delete(pC);//Insid it calls free(pC); //free the memory.
```

参数对象消亡时调用析构函数；函数返回时是生成临时对象返回，在临时对象调用的那条语句之后，临时对象消亡，调用析构。

When we do something in constructor, we need to make sure the function works well in copy constructor.

在构造函数的初始化列表中对类的成员类进行初始化,，顺序是类中成员声明顺序。

Decoration

没有使用成员变量的方法，可以通过空的类指针调用，相当于翻译后传了一个空的 this 指针。static 方法也可以空类指针调用，static 方法参数中没有 this 指针

static 和 global 在 main 结束后按入栈的顺序析构。

sizeof doesn't contain static variables.

friend function, a class must declare WhichClass::WhichFunction is its friend, so that the specific function can access its private members. Also the same as friend class.

friend relationship cannot be passed, or be inherited.

每个对象的空间中都有 this 指针。（x）

```
提示编译器不生成默认复制构造函数：A(CONST A&)=delete;
提示编译器提供默认构造函数 A()=default;
默认情况一旦写了构造函数，编译器就不生成无参的默认构造函数了。
委托构造函数 A():A(0,0,0){}
```

const function cannot modidfy class members. final, means this function or class cannot be inherited. override, compiler will check the function, and if it is not override a function in its base, it tells you the error. Outsied, a const class can only use const functions.

关系

Composition，复合，A has a B。类 A 有 1 个 B 的成员变量。deque 读作 dek，queue，读作 q。构造时由内而外；析构时由外而内。

Delegation，委托，类 A 有 1 个 B 的指针的成员变量。Composition by reference。pimpl，pointer to implementation。Handle, body。隔离了接口与具体的实现。编译防火墙。reference counting。共享。copy on write，当多人共享，其中某类想要改变共享的类时，拷贝一份，是他脱离共享。

Inheritance, is a, 继承。子类的对象中有父类的成分。父类的析构函数需要是 virtual 的。非虚函数，不希望 derived class override 的函数。虚函数，希望 derived class override 的函数。通过子类对象调用父类的虚函数，这延缓了执行逻辑，设计模式叫做 template method。Application framework 的常用手法，例如打开文件的流程。pure virtual function, derived class 一定要 override。

```
(*(this->vptr)[n])(this);
```

1 份数据，多个窗口，或多种表达形式。

11.1.8 Template

sortable, comparable, assignable.

Data type int is a comparable modle.

Iterators connects the containers with algorithms.

模板函数

```
template <typename T, int size> // 函数模板参数，包括类型参数T 和数值参数size
int find( T (&ary)[ size ], T var ); // 定义查找函数：在T 类型数组ary 中查找元素var

typedef int(*pf)(int(&)[3],int); //指向该函数模板实例的函数指针

参数 可接收类型举例（不完全）
const T 左值、右值
T* 指针、double a[]型
const T* 左值取地址


template<typename T> class Base{};
template<typename T> class Derived:Base<T>{};
template<typename T> void f(Base<T> obj){}// f可以传递基类实例、子类实例。
```

同名函数的匹配优先级：非模板函数 > 最特化的模板函数> 一般模板函数

11.1.9 Memory

new是先分配memory（调用operator new,内部调用malloc返回void*，再static_cast转到类指针），再调用构造函数

delete先调用析构函数，再调用operator delete（内部调用free）释放内存

Dynamic

```
T* p = new T;
if(p){
    delete p;
    p = nullptr;
}
int arraySize(6);
T* p = new T[arraySize];
if(p){
    delete[] p;
    p = nullptr;
}
```

11.1.10 STL

Contianer

```
Sequence and reversible:
list

Sequence and random access:
vector, deque

Associative and reversible:
    set, multiset, map, multimap

    array, , forward_list,

unordered_set, unordered_multiset, unordered_map, unordered_multimap
```

11.2 HTML

11.2.1 Basic

Html, 超文本标志语言。前端做的多。asp aspx jsp php

用 adobe 的 Dw 开发。

11.3 JAVA

11.3.1 Basic Grammer

```
DOS command
cd\ //back to the root
md xxx // make directory
rd xxx // remove directory
del test.txt // delete the file
rd /s /q testFolder
```

Chapter 12 Else