USTC

Chapter 27
The C Programming Language

王子磊 (Zilei Wang)
Email: zlwang@ustc.edu.cn
http://vim.ustc.edu.cn

Overview

- 本章从C++的角度对C进行最简单的介绍
  - 如果要使用C，请阅读 K&R 或其它书籍
- C 是 C++ 最相关的，在很多领域是兼容的，因此，已有的C++知识大部分是可用的
- Contents
  - C 和 C++
  - 函数原型
  - printf()/scanf()
  - 数组和字符串
  - 内存管理
  - 宏
  - const
  - C/C++ 互操作

Dennis M. Ritchie

C 和 C++

dmr

ken

bwk

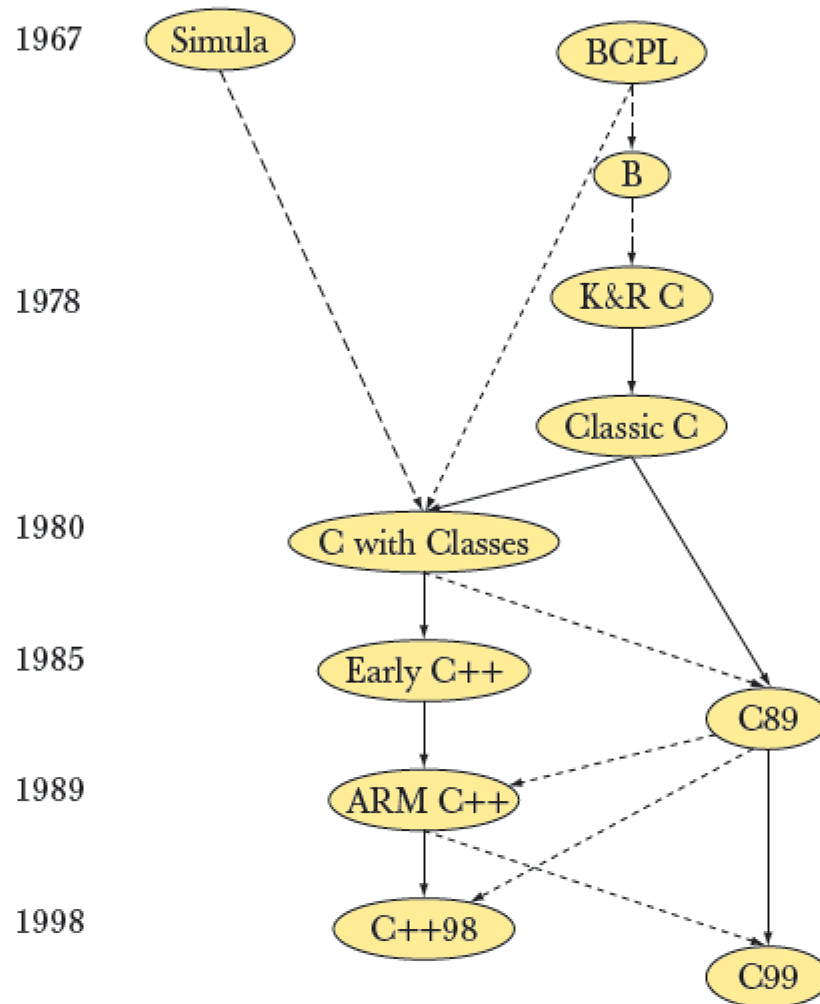
bs

doug

…

- 两者都诞生于新泽西州茉莉山贝尔实验室的计算机科学研究中心

现代 C 和 C++ 是“兄弟”



C和C++

- 在本章中，我们说的“C”是指“ISO C89”
  - 它是目前使用最广泛的一种C定义
    - Classic C基本上已经被取代了(尽管还没有完全取代，奇怪了！)
    - C99还没有广泛应用
- 源码兼容性
  - C基本上是C++的子集
    - 其中一个例外：**int f(int new, int class, int bool);** */* ok in C */*
  - 基本上C和C++中所有构造有相同的含义(语义)
    - 其中一个例外：**sizeof('a')** */* 4 in C and 1 in C++ */*
- 连接兼容性
  - C和C++程序片段能够在单个程序中连接
    - 往往如此
- C++设计为“尽可能的接近C，直到不能再近”
  - 易于两种语言间的转换
  - 两种语言的共存
  - 大多数不兼容性与C++更严格的类型检查有关

C和C++

- 它们都是有ISO标准化委员会定义和控制的
  - 两个分离的委员会
    - 很不幸地导致了不兼容性
  - 使用时有很多支持的实现
  - 比其他任何语言具有更多的平台可用性
- 两者最初的设计目的和当前最重要的应用都是系统级程序设计，例如：
  - 操作系统内核
  - 设备驱动
  - 嵌入式系统
  - 编译器
  - 通信系统

C 和 C++

- 这里，我们
  - 假定你已经了解C++并知道怎么使用它
  - 描述 C 和 C++ 之间的区别
  - 描述如何用 C 提供的特性进行编程
    - 我们的编程理念和技术仍然是一样的，但用来表达我们思想的 工具变化了
  - 描述一些 C 的“陷阱和隐患”
  - 不要从该书中研究所有的细节
    - 兼容性细节是重要的，但基本上是无趣的 ☺

C和C++

- C++是一种偏向于系统编程的通用程序设计语言，它是
  - 一种更好的C
  - 支持数据抽象
  - 支持面向对象程序设计
  - 支持泛型程序设计

C:

- 函数和结构体
- 机器模型(基本类型和操作)
- 编译和链接模型

C 缺少的 (从 C++ 的角度)

- 类和成员函数
  - 使用 struct 和全局函数
- 继承类和虚函数
  - 使用 struct、全局函数和函数
  - 你能够用 C 进行面向对象编程，但不纯粹，你为什么想这样干呢？
  - 你也能够用 C 进行泛型编程，但你为什么想这样干呢？
- 模板和内联函数
  - 使用宏
- 异常
  - 使用错误码、错误返回值等
- 函数重载
  - 给每个函数使用不同的名字
- new/delete
  - 使用 malloc()/free()
- 引用
  - 使用指针
- 常量表达式中的 const
  - 使用宏

C缺少的（从C++的角度）

- 没有类、模板和异常，C不能提供大多数C++的标准库工具
  - 容器
    - **vector**, **map**, **set**, **string**, etc.
    - 使用数组和指针
    - 使用宏（不能类型参数化）
  - STL算法
    - **sort()**, **find()**, **copy()**, …
    - 基本没有替代方案
    - 在可以的地方使用 **qsort()**
    - 自己写或使用第三方库
  - iostreams
    - 使用 stdio: **printf()**, **getch()**, etc.
  - 正则表达式
    - 使用第三方库

C和C++

- 许多有用的代码是用C写的 → 建议
  - 很少语言特性是重要的
    - 原则上，你不需要高级语言，你能够用汇编编写所有的事情(但你为什么想如此呢？)
- 模仿高级编程技术
  - C++直接支持而C不支持
- 使用C++中的C子集
  - 在两种语言下编译保证是一致的
- 调高编译器的警告级别，以获取类型错误
- 对大型程序使用“lint”
  - “lint”是一个一致性检查程序
- C和C++是一样高效的
  - 如果你认为自己看到了区别，请查看一下默认优化器或连接器设置上的区别

函数

- 函数不能重名
- 函数参数类型检查是可选的
- 没有引用类型(因而参数传递没有引用方式)
- 没有成员函数
- 没有内联函数(C 99 除外)
- 有替代的函数定义语法

函数原型(函数参数检查是可选的)

```
/* avoid these mistakes – use a compiler option that enforces C++ rules */

int g(int);             /* prototype – like C++ function declaration */
int h();                /* not a prototype – the argument types are unspecified */

int f(p,b) char* p; char b;    /* old style definition – not a prototype */
{ /* … */ }

int my_fct(int a, double d, char* p)    /* new style definition – a prototype */
{
    f();                /* ok by the compiler! But gives wrong/unexpected results */
    f(d,p);             /* ok by the compiler! But gives wrong/unexpected results */
    h(d);               /* ok by the compiler! But may give wrong/unexpected results */
    f(d);               /* ok by the compiler! But may give wrong/unexpected results */

    g(p);               /* error: wrong type */
    g();                /* error: argument missing */
}
```

Algol60 风格

printf() – 人们最喜欢的一个 C 函数

Format string

```
/* no iostreams – use stdio */
#include<stdio.h>      /* defines int printf(const char* format, …); */

int main(void)
{
    printf("Hello, world\n");
    return 0;
}

void f(double d, char* s, int i, char ch)
{
    printf("double %g string %s int %i char %c\n", d, s, i, ch);
    printf("goof %s\n", i);   /* uncaught error */
}
```

Arguments to be formatted

初学者容易犯的错误

Format strings

Formatting characters

scanf( ) 及相似函数

```
/* the most popular input functions from <stdio.h>: */
int i = getchar();    /* note int, not char;
                         getchar() returns EOF when it reaches end of file */
p = gets();           /* read '\n' terminated line into char array pointed to by p */

void f(int* pi, char* pc, double* pd, char* ps)
{   /* read into variables whose addresses are passed as pointers:  */
    scanf("%i %c %g %s", pi, pc, pd, ps);
    /* %s skips initial whitespace and is terminated by whitespace  */
}
int i; char c; double d; char s[100]; f(&i, &c, &d, s);  /* call to assign to i, c, d, and s */
```

- 永远不要使用 gets() 或 scanf(“%s”)!
  - 认为它们是“有毒”的
  - 它们是许多安全问题的来源
  - 很容易设置溢出并容易利用
  - 使用 getchar()

printf( ) 和 scanf( ) 不是类型安全的

```
double d = 0;
int s = 0;
printf("d: %d , s: %s\n", d, s);    /* compiles and runs
                                        the result might surprise you */
```

"s" for "string"

"d" for "decimal", not "double"

- 尽管易于出错，但 printf() 对内建类型是方便的
- printf() 格式不能扩展到用户自定义类型
  - e.g. 对 My_type 类型值不能%M
- 小心：用户提供格式字符串的 printf () 是一种黑客工具

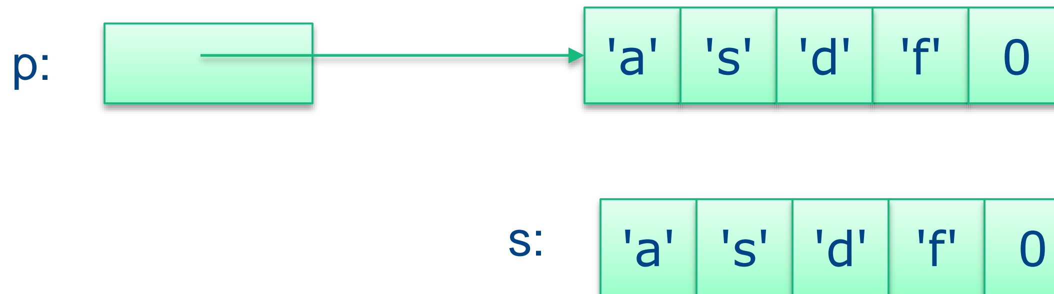
数组和指针

- 差不多就是C++中定义的那样
- 在C中，你基本上总要使用它们
  - 因为没有 **vector, map, string**, etc.
- 记住
  - 一个数组不知道它自身的长度
  - 没有数组赋值
    - 使用 **memcpy()**
  - C风格字符串是一种以0结尾的字符数组

C 风格字符串

❖ 在 C 中，一个字符串(称之为 C 字符串或 C 风格字符串)是一个以 0 结尾的字符数组

```
char* p = "asdf";
char s[ ] = "asdf";
```



C 风格字符串

- 比较字符串

```
#include <string.h>
if (s1 = = s2) {              /* do s1 and s2 point to the same array?
                                    (typically not what you want) */
}

if (strcmp(s1,s2) = = 0) { /* do  s1 and s2 hold the same characters? */
}
```

- 获取字符串的长度

```
int lgt = strlen(s);        /* note: goes through the string at run time
                               looking for the terminating 0 */
```

- 拷贝字符串

```
strcpy(s1,s2); /* copy characters from s2 into s1
                  be sure that s1 can hold that many characters */
```

C 风格字符串

- 字符串拷贝函数 strcpy() 是典型的 C 函数 (可在 ISO C 标准库中找到)
- 除非你理解了下面的实现，否则不要声称自己懂 C 语言：

```
char* strcpy(char *p, const char *q)
{
    while (*p++ = *q++);
    return p;
}
```

常用的考试题…

- 相关解释请参见 K&R 或 TC++PL

标准函数库

- **<stdio.h>** **printf(), scanf()**, etc.
- **<string.h>** **strcmp()**, etc.
- **<ctype.c>** **isspace()**, etc.
- **<stdlib.h>** **malloc()**, etc.
- **<math.h>** **sqrt()**, etc.

- 警告：默认情况下，Microsoft 会尽力让你使用更安全的、但不标准的函数来替代不安全的 C 标准库函数

自由存储：malloc()/free()

```
#include<stdlib.h>

void f(int n) {
    /* malloc() takes a number of bytes as its argument */
    int* p = (int*)malloc(sizeof(int)*n);        /* allocate an array of n ints */
    /* … */
    free(p);        /* free() returns memory allocated by malloc() to free store */
}
```

自由存储：malloc()/free()

- 很少的编译时检查

  /* *malloc*() *returns a* ***void****. *You can leave out the cast of malloc(), but don't* */

  double* p = malloc(sizeof(int)*n);   /* *probably a bug* */

- 很少的运行时检查

  int* q = malloc(sizeof(int)*m); /* *m ints* */

  for (int i=0; i<n; ++i) init(q[i]);

- 没有初始化和清除操作
  - malloc() 不调用构造函数
  - free() 不调用析构函数
  - 记住编写你自己的 init() 和 cleanup()
- 没有什么方法能够保证自动清除
- 在 C++ 程序中不要使用 malloc()/free()
  - new/delete 一样快，且几乎总是更好的选择

没有转换的 malloc()

- C/C++ 在实际代码中的主要不兼容性
  - 非类型安全
  - 历史上准标准 C 的兼容性特性
- 总是有争议的
  - 不必如此 IMO （in my opinion）

```
void* alloc(size_t x);           /* allocate x bytes
                                    in C, but not in C++, void* converts to any T* */
void f (int n)
{
    int* p = alloc(n*sizeof(int));                  /* ok in C; error in C++ */
    int* q = (int*)alloc(n*sizeof(int)); /* ok in C and C++ */
    /* … */
}
```

void*

- 为什么在 C 中 void* 能够转换为 T*，而在 C++ 中不行？
  - C 需要它来转换 **malloc()** 的结果（你更省劲）
  - C++ 不需要：使用 **new**
- 为什么从 **void*** 到 **T*** 的转换不是类型安全的？

```
void f()
{
  char i = 0;
  char j = 0;
  char* p = &i;
  void* q = p;
  int* pp = q;          /* unsafe, legal C; error in C++ */
  *pp = -1; /* overwrite memory starting at &i */
}
```

注释

- // 注释被 Bjarne Stroustrup 从 C 的祖先 BCPL 引入到 C++ 中，这时你会讨厌键入 /* ... */ 注释
- // 注释已经被大多数 C 语言接受，包括最新的 ISO 标准 C (C99)

const

```
// in C, a const is never a compile time constant
const int max = 30;
const int x;           // const not initialized: ok in C (error in C++)

void f(int v)
{
    int a1[max];       // error: array bound not a constant (max is not a constant!)
    int a2[x];         // error: array bound not a constant (here you see why)
    switch (v) {
    case 1:
            // ...
    case max:          // error: case label not a constant
            // ...
    }
}
```

用宏代替 const

```
#define  max 30

void f(int v)
{
    int a1[max];     // ok
    switch (v) {
    case 1:
        // ...
    case max:        // ok
        // ...
    }
}
```

小心使用宏

```
#include "my_header.h"
// ...
int max(int a, int b) { return a>=b?a:b; }        // error: "obscure error message"
```

- 恰巧上一页中我们的头文件 my_header.h 保护了宏 max，因此编译器看到的将是

  ```
  int 30(int a, int b) { return a>=b?a:b; }
  ```

- 不要埋怨地怀疑它！
- 在流行的头文件中有成千上万个宏
- 总是采用 ALL_CAPS 类的名字来定义宏，例如：

  ```
  #define MY_MAX 30
  ```

  除了宏不要在其他任何地方使用 ALL_CAPS 的名字
- 不幸地，不是每个人都遵从这个 ALL_CAPS 约定

C/C++ 互操作

- 能够工作，因为共享连接模型
- 能够工作，因为简单对象的共享模型
  - 内建类型和 structs/classes
- 优化/高效
  - 不需要场景下的重新格式化/转换

C++ 调用 C

❖ 使用 extern "C" 来告诉 C++ 编译器使用 C 的调用惯例

```
// calling C function from C++:

extern "C" double sqrt(double);    // link as a C function

void my_c_plus_plus_fct()
{
        double sr = sqrt(2);
        // ...
}
```

C 调用 C++

- 对 C 编译器**不需要**特别的动作

```
/* call C++ function from C: */

int call_f(S* p, int i);  /* call f for object pointed to by p with argument  i */
struct S* make_S(int x,  const char* p);  /* make  S( x,p) on the free store */

void my_c_fct(int i)
{
        /* ... */
        struct S* p = make_S(17, "foo");
        int x = call_f(p,i);
        /* ... */
}
```

单词计数实例 (C++ version)

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

int main()
{
    map<string,int> m;
    string s;
    while (cin>>s) m[s]++;    // use getline() if you really want lines
    for(map<string,int>::iterator p = m.begin(); p!=m.end(); ++p)
            cout << p->first << " : " << p->second << "\n";
}
```

单词计数实例(C version)

```
// word_freq.c
// Walter C. Daugherity

#include <stdio.h>
#include <stdlib.h>
 #include <string.h>

#define MAX_WORDS 1000 /* max unique words to count */
#define MAX_WORD_LENGTH 100

#define STR(s) #s                    /* macros for scanf format */
#define XSTR(s) STR(s)

typedef struct record{
      char word[MAX_WORD_LENGTH + 1];
      int count;
} record;
```

数据结构定义

单词计数实例 (C version)

```
int main()
{
    // ... read words and build table ...
    qsort(table, num_words, sizeof(record), strcmp);
    for(iter=0; iter<num_words; ++iter)
        printf("%s  %d\n",table[iter].word,table[iter].count);
    return EXIT_SUCCESS;
}
```

单词计数实例 (most of main)

```
record table[MAX_WORDS + 1];
int num_words = 0;
char word[MAX_WORD_LENGTH + 1];
int iter;
while(scanf("%" XSTR(MAX_WORD_LENGTH) "s", word) != EOF) {
    for(iter = 0; iter < num_words && strcmp(table[iter].word, word); ++iter);
    if(iter == num_words) {  //  not find
        strncpy(table[num_words].word, word, MAX_WORD_LENGTH + 1);
        table[num_words++].count = 1;
    }
    else table[iter].count++;    // find
    if(num_words > MAX_WORDS){
        printf("table is full\n");
        return EXIT_FAILURE;
    }
}
```

单词计数实例 (C version)

- 注解
  - 是通用的 C 风格 (不是 BS 写的)
  - 它太长和复杂了！(我的第一反应 – BS)
  - 你不需要任何高级复杂的语言特性！ (不是我的注解 – BS)
  - IMHO (In My Humble Opinion) 对使用 C 不是个大问题
    - 并不是一个非典型应用，但不是底层系统编程
  - 除了在 C++ 中，它也应该“C++ 化”，qsort() 的参数应该转换为合理的类型
    - (int (*)(const void*, const void*)) strcmp
  - 那些宏做了什么？
  - 最大能够达到 MAX_WORD 单词
  - 不要处理超过 MAX_WORD_LENGTH 长度的单词
  - 先读后排序
    - 显然要比通用 C++ 版本慢 (C++使用了一个 map)

更多信息

- Kernighan & Ritchie: The C Programming Language
  - 经典
- Stroustrup: TC++PL, Appendix B: Compatibility
  - C/C++ 的不兼容性，在作者 BS 的主页上
- Stroustrup: Learning Standard C++ as a New Language.
  - 风格和技术比较
  - www.research.att.com/~bs/new_learning.pdf
- 还有很多复习书：www.accu.org