HCC 技术沙龙・第八期 / HCC Tech Salon, 8th

你所不知道的 C 语言

The C Programming Language That You Don't Know

Junde Yhi <lmy441900@live.com>

C 语言简史

- CppReference.com
  - BCPL→NB→C
  - K&R C: 最初定稿版本
  - ANSI C (**C89**) / ISO **C90**: 第一个标准化版本
  - ISO C95, **C99**, **C11**, C17, ***C2x****: 新标准

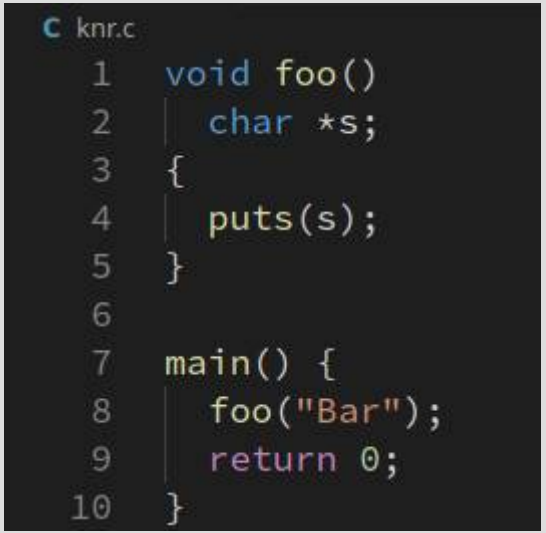
* 尚未发布

C 标准

- K&R C （史前事实标准）
  - *The C Programming Language, First Edition* (1978)
- C89 / C90 (1989)
- C99 (1999)
- C11 (2011)
- C2x (2021?)

C 标准： K&R C

- 撰写函数的方式略有不同
  - 参数列表外置
  - 默认返回整形 (int)
- *The C Programming Language* 的第二版和 C89 几乎没有区别
  - 甚至是很不错的参考书
  - 005.133/K47/1988



C 标准： ANSI C89 / ISO C90

- 变量声明必须紧跟在代码块起始之后
- 为了保持绝对的兼容性， Dr. Philippe Meunier 在讲义中只使用 C89 / C90
  - 这样就算是上古系统也能顺利编译

c89.c

```c
int main(void) {
  int i = 3;
  int j = 4;

  printf("i = %d, j = %d\n", i, j);

  {
    int sum = i + j;
    printf("sum = %d\n", sum);
  }

  return 0;
}
```

C 标准： C99

- 日常使用的 C 语言标准；一般情况下
  - 我们依赖（学了）一部分 C99 才制定的语法
  - 我们也几乎没有见到（没学）另一部分
    - 并且这些语法非常有用
- 现代编译器都支持 C99 ， MSVC (Visual Studio) 除外
  - MSVC 对 C 语言的支持一直处于落后状态

C 标准： C99 — 单行注释

- 从 C99 开始， C 语言支持从 C++ 学来的单行注释
  - 换句话说，在不支持 C99 的编译器中，只能使用多行注释 (/* */)
    - 并且至今仍**不允许嵌套**

c99-comments.c

```c
int main(void) {
  // This is a single line comment.
  puts("Hello world!");

  /* This is a multi-
     line comment. */
  return 0;
}
```

C 标准： C99 — 循环变量初始化

c89-for-init.c

```c
int main(void) {
  int sum = 0;

  int i;
  for (i = 1; i <= 100; i++)
    sum += i;

  printf("%d\n", sum);

  return 0;
}
```

C89

c99-for-init.c

```c
int main(void) {
  int sum = 0;

  for (int i = 1; i <= 100; i++)
    sum += i;

  printf("%d\n", sum);

  return 0;
}
```

C99

C 标准： C99 — 布尔值

- C 语言从来都没有布尔值
  - 0 就是 False，其它值都是 True
- C99 向 C++ 学习了布尔值，但并没有什么人用
  - `#include <stdbool.h>`
  - 它本质仍然是 8 位整数
  - 容易和 C89 代码造成不兼容

c99-bool.c

```c
#include <stdbool.h>

bool foo(_Bool b) {
    if (b)
        return false;
    else
        return true;
}

int main(void) {
    if (foo(false))
        return 0;
    else
        return 1;
}
```

C 标准： C99 — 指定成员初始化

c89-designated-init.c

```c
struct Foo {
  int a;
  int b;
};

int main(void) {
  struct Foo foo;
  foo.a = 3;
  foo.b = 4;

  /* Or
  struct Foo foo = {3, 4};
  */

  printf("a = %d, b = %d\n", foo.a, foo.b);

  return 0;
}
```

C89

c99-designated-init.c

```c
struct Foo {
  int a;
  int b;
};

int main(void) {
  struct Foo foo = {
    .a = 3,
    .b = 4,
  };

  printf("a = %d, b = %d\n", foo.a, foo.b);

  return 0;
}
```

C99

C 标准： C99 — 随处声明变量

c89.c

```c
int main(void) {
  int i = 3;
  int j = 4;

  printf("i = %d, j = %d\n", i, j);

  {
    int sum = i + j;
    printf("sum = %d\n", sum);
  }

  return 0;
}
```

C89

c99-decl.c

```c
int main(void) {
  int i = 3;
  int j = 4;

  printf("i = %d, j = %d\n", i, j);

  int sum = i + j;
  printf("sum = %d\n", sum);

  return 0;
}
```

C99

C 标准： C99 — 复合字面量

- 用于方便地就地定义结构体类型变量

c99-compound-literal.c

```c
struct Foo {
  int a;
  int b;
};

int main(void) {
  struct Foo foo = {0, 0};
  printf("a = %d, b = %d\n", foo.a, foo.b);

  foo = (struct Foo) {.b = 4, .a = 3};
  printf("a = %d, b = %d\n", foo.a, foo.b);

  return 0;
}
```

C 标准： C99 — 变长数组成员

- 一个看起来并没有什么技术含量的标准操作，用于方便直接伸缩结构体大小
  - 例如一个带长度信息的字符串结构体类型就可以直接获取长度而不需要使用 `strlen` 动态计算
- 比较少用

c99-flexible-array-member.c

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

typedef struct {
  size_t size; // <- Must have other members
  char data[]; // <- Array at the end
} String;

int main(void) {
  String *str = malloc(sizeof(String) + 32);
  strcpy(str->data, "Hello world!");
  str->size = 13;

  printf("String %s has length %zu\n",
    str->data, str->size);

  free(str);

  return 0;
}
```

C 标准： C99 — 变长数组

- C99 允许数组在声明时指定一个变量作为长度，一旦这个变量被赋值那么数组的长度就会变成这个值
- 直接声明的数组存储在栈内存中而非堆内存 (malloc)
  - 这就让程序暴露在爆栈的风险之中
  - **不推荐使用**

c99-vla.c

```c
#include <stdio.h>

int main(void) {
  int n = 0;
  int a[n];

  puts("How long do you want?");
  scanf("%d", &n);

  a[n - 1] = 114514;
  printf("%d\n", a[n - 1]);

  return 0;
}
```

C 标准： C99 — 写在中间

- C99 还有很多特性，在此不一一介绍
  - 变长参数列表
  - 新关键词： `restrict`, `long long`, `inline`
  - 复数类型 (`_Complex`)
  - ……
- ~~`stdio.h` 警察~~细心的同学会发现我有时候并没有 `#include <stdio.h>`
  - Fun Fact: 因为这些函数都在缺省链接的标准 C 函数库内，所以不 include 顶多带来一些编译器警告
  - 关于预处理器、编译器、汇编器和链接器的知识，请选修 COMP3173 Compiler Construction

C 标准： C11

- 针对编程安全 (Safe) 在各方面增强了 C 语言
  - 包括 Visual Studio 频繁警告的 *_s 函数也出自 C11 （稍后提及）
- 我甚至没有完全吃透 C11 新增的内容
  - 但有一些是用到了的

C 标准： C11 — 移除 gets

- 求你们不要再用 gets 了，编译器都把你们惯坏了

c11-gets.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  char *line = malloc(16);
  puts("Please input your name:");
  gets(line);
  printf("Your name is %s\n", line);
  free(line);
  return 0;
}
```

Tilix

1: lmy441900@yhi-latitude-13:~/repositories/tcpltydn

```
lmy441900@yhi-latitude-13 [ tcpltydn@ ] $ gcc -std=c11 -g -o c11-gets c11-gets.c
c11-gets.c: In function ‘main’:
c11-gets.c:7:3: warning: implicit declaration of function ‘gets’; did you mean ‘fgets’? [-Wimplicit-function-declaration]
    7 |   gets(line);
      |   ^~~~
      |   fgets
/bin/ld: /tmp/ccknfuzJ.o: in function `main':
/home/lmy441900/repositories/tcpltydn/c11-gets.c:7: warning: the `gets' function is dangerous and should not be used.
lmy441900@yhi-latitude-13 [ tcpltydn@ ] $
```

C 标准： C11 — 匿名结构 / 联合体

c99-anon-struct-union.c

```c
#include <stdio.h>

struct Foo {
  int bar;
  union _baz {
    int   ibaz;
    char *sbaz;
  } baz;
};

int main(void) {
  struct Foo foo = {.bar = 3, .baz.ibaz = 4};
  printf("%d / %d\n", foo.bar, foo.baz.ibaz);

  struct Foo oof = {.bar = 4, .baz.sbaz = "kksk"};
  printf("%d / %s\n", oof.bar, oof.baz.sbaz);

  return 0;
}
```

C99

c11-anon-struct-union.c

```c
#include <stdio.h>

struct Foo {
  int bar;
  union {
    int   ibaz;
    char *sbaz;
  };
};

int main(void) {
  struct Foo foo = {.bar = 3, .ibaz = 4};
  printf("%d / %d\n", foo.bar, foo.ibaz);

  struct Foo oof = {.bar = 4, .sbaz = "kksk"};
  printf("%d / %s\n", oof.bar, oof.sbaz);

  return 0;
}
```

C11

C 标准： C11 — 静态断言

- 用于在编译时判断一些条件来保证程序对编译环境的假设是否有保证
  - 在此之前，要么不管，要么用黑魔法
- 编写认真考虑跨平台问题的软件时需要用到
  - 每个平台、每款编译器提供的环境都不太一样
  - 稍后将提及

c11-static-assert.c

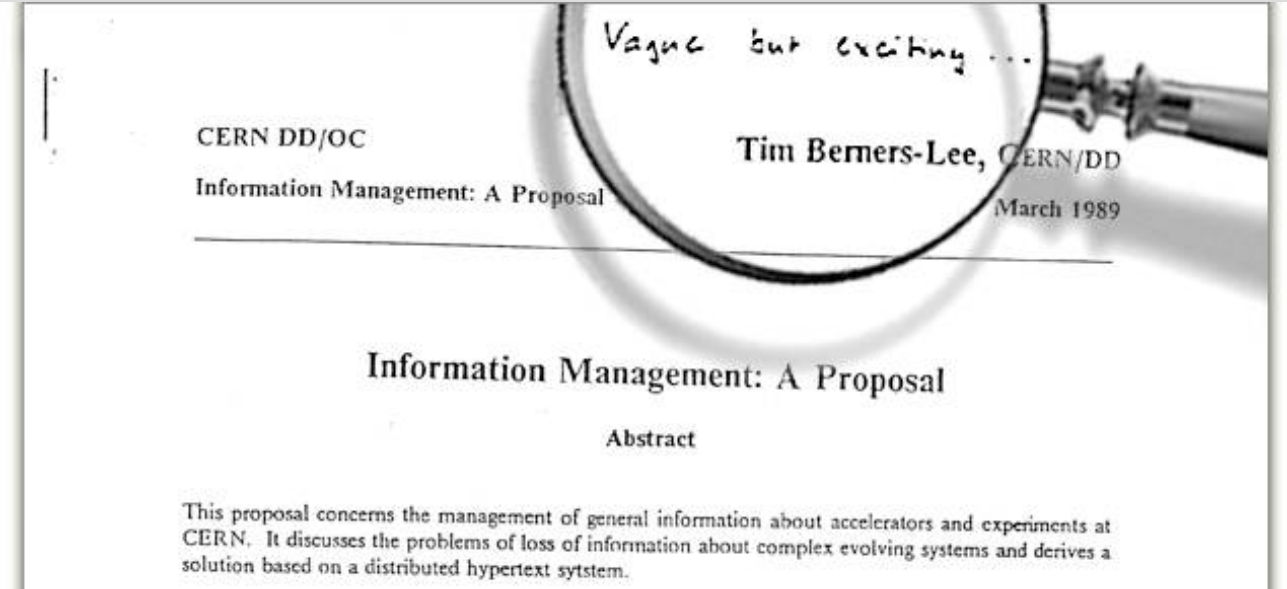
```c
#include <assert.h>
#include <stdbool.h>

int main(void) {
  // Dynamic assertion
  assert(true == false);

  // Static assertion (C11)
  static_assert(
    true == false,
    "true is not false"
  );

  return 0;
}
```

C 标准： C11 — 附件 K

- i.e. **边界检查函数** (*_s)
- CppReference.com

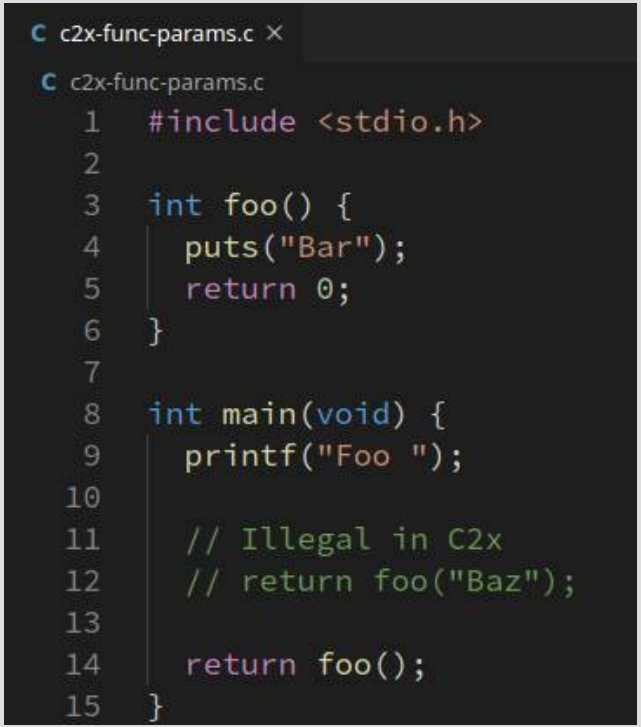
C 标准： C2x

- 尚未发布；预计在 2021 年发布



C 标准： C2x — 无参函数

- 在 C2x 之前，函数参数列表兼容 K&R C
  - 即无形参函数代表函数接受任意数量实参
- 又一个向 C++ 学习的特性



C 标准： C2x — 属性标记

- 又一个从 C++ 学来的语法
- 用于替代原本由编译器各自实现的 `__attribute__` 宏以使语法更加干净
  - 一般用来方便编译器作优化

c2x-attr.c

```c
#include <stdio.h>

[[deprecated]]
int foo() {
  puts("Foo");
  return 1;
}

int main() {
  int f = foo(); // Warning here
  switch (f) {
    case 0:
      [[fallthrough]];
    case 1:
      puts("Bar");
      // Warning here
    default:
      puts("Baz");
      break;
  }

  return 0;
}
```

C 标准： C2x — 新标准库函数

- 从 POSIX 学来的
- 将会提升生活质量的函数：
  - **`strdup` / `strndup`**
  - `memccpy`

c2x-strdup.c

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

static const char const *s =
  "Hello world!";

int main() {
  // This is illegal
  // s[4] = ' ';
  puts(s);

  char *s_dyn = strdup(s);
  s_dyn[4] = ' ';
  puts(s_dyn);
  free(s_dyn);

  return 0;
}
```

C 标准： C2x — 新标准库函数

- 从 POSIX 学来的
- 将会提升生活质量的函数：
  - `strdup` / `strndup`
  - **`memccpy`**

c2x-memccpy.c

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

static const char const *s =
  "Hello world!";

int main() {
  char *hello = malloc(6);
  if (!hello) return 1;

  memccpy(hello, s, ' ', 6);
  hello[4] = '!';

  puts(hello);

  free(hello);
  return 0;
}
```

C 标准： C2x — 单参静态断言

c11-static-assert.c

```c
#include <assert.h>
#include <stdbool.h>

int main(void) {
  // Dynamic assertion
  assert(true == false);

  // Static assertion (C11)
  static_assert(
    true == false,
    "true is not false"
  );

  return 0;
}
```

C11

c2x-static-assert.c

```c
#include <assert.h>
#include <stdbool.h>

int main(void) {
  // Dynamic assertion
  assert(true == false);

  // Static assertion (C2x)
  static_assert(true == false);

  return 0;
}
```

C2x

C 魔法

- C 宏 (Macro)，特别是宏函数 (Macro Function) 这一特性使得 C 比我们想象中的要高层次
  - 基于此我们可以“生成”一些代码
    - 借助预处理器
- 语言不是范式，范式不是语言（稍后提及）

X Macro

- 用于批量生成重复代码
  - 先定义一个宏函数
  - 然后定义一组调用了 X 这个宏函数的数据
  - 然后在需要配合这组数据生成代码的地方，依次
    - 定义需要的宏函数名为 X
    - 写下这组数据的名字
    - 去定义 X，防止误用

x-macro.c

```c
#include <stdio.h>

#define PRINT_TWO_SUMS(a, b, s) \
  { \
    printf("%d + %d = %d\n", a, b, s); \
  }

#define TWO_SUMS \
  X(0, 0, 0) \
  X(0, 1, 1) \
  X(1, 0, 1) \
  X(1, 1, 2)

int main(void) {
#define X PRINT_TWO_SUMS
TWO_SUMS
#undef X

  return 0;
}
```

```
lmy441900@yhi-latitude-13 [ tcpltydn@ ] ! gcc -g -o x-macro x-macro.c
lmy441900@yhi-latitude-13 [ tcpltydn@ ] $ ./x-macro
0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 2
```

X Macro

- X Macro 这一名字源于数据组内调用的函数名 X （如图所示）
  - 最早可溯源到 1960 年代
  - https://en.wikipedia.org/wiki/X_Macro
- Don’t Repeat Yourself (DRY) :)

x-macro.c

```c
#include <stdio.h>

#define PRINT_TWO_SUMS(a, b, s) \
  { \
    printf("%d + %d = %d\n", a, b, s); \
  }

#define TWO_SUMS \
  X(0, 0, 0) \
  X(0, 1, 1) \
  X(1, 0, 1) \
  X(1, 1, 2)

int main(void) {
#define X PRINT_TWO_SUMS
TWO_SUMS
#undef X

  return 0;
}
```

```
lmy441900@yhi-latitude-13 [ tcpltydn@ ] ! gcc -g -o x-macro x-macro.c
lmy441900@yhi-latitude-13 [ tcpltydn@ ] $ ./x-macro
0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 2
```

用 C 面向对象编程

- 面向对象是一种编程范式，不是语言
  - 因此每种高级语言都可以或多或少实现面向对象
- C 语言语法上没有面向对象元素，所以我们只能模拟它
  - 结构体 (struct) 模拟
  - Glib / GObject

用 C 面向对象编程：结构体模拟

- 对象的本质：一些数据（成员）和一些附属于其上的方法（函数）
  - 由于 C 语言中函数是指针（！），做到这一点没有什么大问题

图太长，放不下

(c-oop-sim.c)

用 C 面向对象编程： GObject

- GNOME* 人认为如果我们用 C 语言写一个面向对象系统，然后让所有语言都绑定到 C 语言，不就世界大同了吗?
  - 于是有了一个成熟的 C 语言面向对象编程系统
- GNOME 配套的 GLib 基础工具库和 GTK 图形库都基于 GObject

c-oop-gobject.c

```c
#include <gtk/gtk.h>

int main(void) {
  gtk_init(NULL, NULL);

  GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(window), "Hello world!");
  g_signal_connect(window, "destroy", gtk_main_quit, NULL);
  gtk_widget_show_all(window);
  gtk_main();

  return 0;
}
```

使用 GTK 创建一个简单的窗口

* https://www.gnome.org/

用 C 函数式编程

- TL;DR: 可以，但过于玄学
- 概括：
  - 高阶函数 → 函数指针
  - map, reduce / fold, foreach, ... → 函数指针
  - 闭包 → 结构体模拟
  - Monad → https://segmentfault.com/a/1190000012435966

整形 (int) 不定长

未定义行为

编译器魔法优化

- https://godbolt.org

写在最后

- 截图中代码已上传，可供参考
  - https://github.com/lmy441900/tcpltydn