When and where it exists?

C 程 序 设 计

C Programming

12

存储类、链接和内存管理

理论课程

厦門大學 XIAMEN UNIVERSITY

信息学院 黄 炜

（国家示范性软件学院） 博士，副教授

School of Informatics Dr. Wei Huang

知识框架

- 作用域
- 链接
- 存储时期
- 存储类
- 随机函数

内容纲要

1	作用域、链接和存储时期
2	存储类
3	随机函数
4	伪随机数
5	动态内存分配

基本概念

- 存储类：按变量和函数的作用域和生存期划分类别
- 作用域：变量和函数名在代码中的使用范围
- 存储时期：变量在内存中保留的时间
- 链接：变量和函数名是否可以被其它范围使用
- 变量的作用域和链接一起标明程序的哪些部分可以用变量名使用该变量

作用域

- 代码块作用域 (block scope)
  - 默认在声明语句之后且在声明语句的代码块内使用

```
double blocky(double cleo)
{
    double patrick = 0.0;
    int i;
    for (i = 0; i < 10; i++)
    {
        double q = cleo * i; // start of scope for q
        ...
        patrick *= q;
    } // end of scope for q
    ...
    return patrick;
}
```

走出函数，变量就不起作用了

走出花括号，变量就不起作用了

走出花括号，变量就不起作用了

作用域

- 文件作用域 (file scope)
  - 在所有函数之外声明的变量或函数，在该文件内使用

```
#include <stdio.h>
int units = 0; /* a variable with file scope */
void critic(void);
int main(void)
{
    ...
}
```

作用域

- 函数原型作用域 (function prototype scope)
  - 使用变长数组参量时

```
void use_a_VLA(int n, int m,int ar[n][m]);
```

- 函数作用域 (function scope)
  - 函数中的goto标签对于函数内部可见

链接

- 空链接 (no linkage)
  - 具有代码块作用域或函数原型作用域的变量
  - 由其所在代码块或函数原型所私有
- 外部链接 (external linkage)
  - 文件作用域变量具有内部或外部链接
  - 外部链接的变量可以在多文件程序的任何地方使用
- 内部链接 (internal linkage)
  - 只能在当前文件的任何地方使用

存储时期

- 静态存储时期 (static storage duration)
  - 文件作用域的变量具有静态存储时期
    - 运行声明语句时分配内存；退出程序前不释放。
  - 注意：标注 static 不表示静态存储时期，而是链接类型
- 自动存储时期 (auto storage duration)
  - 代码块作用域的变量一般具有自动存储时期
    - 进入代码块时分配内存；退出则释放。

内容纲要

1	作用域、链接和存储时期
2	存储类
3	随机函数
4	伪随机数
5	动态内存分配

存储类

- 5种存储类

存储类	时期	作用域	链接	声明方式
自动	自动	代码块	空	代码块内
寄存器	自动	代码块	空	代码块，使用关键字register
具有外部链接的静态	静态	文件	外部	所有函数之外
具有内部链接的静态	静态	文件	内部	所有函数之外，使用关键字static
空链接的静态	静态	代码块	空	代码块内，使用关键字static

*自动变量 (auto)

- 声明自动变量：auto
  - 显式：`auto <类型> <变量名>`
  - 隐式：默认值 `<类型> <变量名>`
  - 内层定义名称与外层定义相同时，内层自动覆盖外层
- 初始化
  - 自动变量不会自动初始化，需自行初始化

```c
// hiding.c -- variables in blocks
#include <stdio.h>
int main()
{
    int x = 30;       // original x
    printf("x in outer block: %d at %p\n", x, &x);
    {
        int x = 77;  // new x, hides first x
        printf("x in inner block: %d at %p\n", x, &x);
    }
    printf("x in outer block: %d at %p\n", x, &x);
    while (x++ < 33) // original x
    {
        int x = 100; // new x, hides first x
        x++;
        printf("x in while loop:
    }
    printf("x in outer block: %d
    return 0;
}
```

```
x in outer block: 30 at 0x7fff47543720
x in inner block: 77 at 0x7fff47543724
x in outer block: 30 at 0x7fff47543720
x in while loop: 101 at 0x7fff47543724
x in while loop: 101 at 0x7fff47543724
x in while loop: 101 at 0x7fff47543724
x in outer block: 34 at 0x7fff47543720
```

```c
// forc99.c -- new C99 block rules
#include <stdio.h>
int main()
{
    int n = 8;
    printf("   Initially, n = %d at %p\n", n, &n);
    for (int n = 1; n < 3; n++)
        printf("      loop 1: n = %d at %p\n", n, &n);
    printf("After loop 1, n = %d at %p\n", n, &n);
    for (int n = 1; n < 3; n++)
    {
        printf(" loop 2 index n = %d at %p\n", n, &n);
        int n = 6;
        printf("      loop 2: n = %
        n++;
    }
    printf("After loop 2, n = %d at
    return 0;
}
```

```
   Initially, n = 8 at 0x7fffbd8421fc
      loop 1: n = 1 at 0x7fffbd842204
      loop 1: n = 2 at 0x7fffbd842204
After loop 1, n = 8 at 0x7fffbd8421fc
 loop 2 index n = 1 at 0x7fffbd842200
      loop 2: n = 6 at 0x7fffbd842204
 loop 2 index n = 2 at 0x7fffbd842200
      loop 2: n = 6 at 0x7fffbd842204
After loop 2, n = 8 at 0x7fffbd8421fc
```

*寄存器变量 (register)

- 声明寄存器变量：register
  - 显式：`register <类型> <变量名>`
  - 位置：

```
int main(void)
{
    register int count;
```

```
void macho(register int count)
{
    ...
```

  - 请求变量存储在寄存器或高速内存中，提供高速访问
    - 可声明的类型和数量有限，请求不到也没办法
  - 不可以取内存地址（寄存器在CPU里）

代码块作用域的静态变量(static)

- 声明静态变量：static
  - 显式：
    ```
    static <类型> <变量名>
    ```
  - 位置：函数参量不能使用static
  - 注意：静态是指内存位置静态，不是其值静态
    - 不随着函数运行结束而消失和初始化
- 初始化：如果声明时未初始化，则初始化为0。
  - 先声明后赋值，每次运行重新赋值
    ```
    static int val; val=0;
    ```
  - 声明时赋值，只在第一次运行赋值
    ```
    static int val=0;
    ```

```c
/* loc_stat.c -- using a local static variable */
#include <stdio.h>
void trystat(void);
int main(void)
{
    int count;
    for (count = 1; count <= 3; count++)
    {
        printf("Here comes iteration %d:\n", count);
        trystat();
    }
    return 0;
}
void trystat(void)
{
    int fade = 1;
    static int stay = 1;
    printf("fade = %d and stay = %d\n", fade++, stay++);
}
```

```
Here comes iteration 1:
fade = 1 and stay = 1
Here comes iteration 2:
fade = 1 and stay = 2
Here comes iteration 3:
fade = 1 and stay = 3
```

外部链接的静态变量 (extern)

- 声明外部变量：extern
  - 显式：
    - 定义声明：`<类型> <变量名>`
    - 引用声明：`extern <类型> <变量名>`
  - 位置
    - 在一个文件的全局变量区域做定义声明
    - 在用到该全局变量的其他文件里做引用声明声明
- 初始化
  - 应显式初始化，且应使用常量赋值。

```c
/* Example 1 */
int Hocus;
int magic();

int main(void)
{
    extern int Hocus; // Hocus declared external
    ...
}

int magic()
{
    extern int Hocus; // same Hocus as above
    ...
}
```

```c
/* Example 2 */
int Hocus;
int magic();

int main(void)
{
    extern int Hocus; // Hocus declared external
    ...
}

int magic()
{
    // Hocus not declared but is known
    ...
}
```

```c
/* Example 3 */
int Hocus;
int magic();

int main(void)
{
    int Hocus; // Hocus declared, is auto by default
    ...
}

int Pocus;

int magic()
{
    auto int Hocus; // local Hocus declared automatic
    ...
}
```

```c
/* global.c  -- uses an external variable */
#include <stdio.h>
int units = 0;          /* an external variable      */
void critic(void);

int main(void)
{
    extern int units;  /* an optional redeclaration */

    printf("How many pounds to a firkin of butter?\n");
    scanf("%d", &units);
    while ( units != 56 )
        critic();
    printf("You must have looked it up!\n");

    return 0;
}
```

```c
void critic(void)
{
    /* optional redeclaration omitted */
    printf("No luck, my friend. Try again.\n");
    scanf("%d", &units);
}
```

```
How many pounds to a firkin of butter?
32↲
No luck, my friend. Try again.
56↲
You must have looked it up!
```

外部链接的静态变量(extern)

- 根据“内覆盖外”的原则

```
int traveler = 1; // external linkage
static int stayhome = 1; // internal linkage
int main()
{
        extern int traveler; // use global traveler
        extern int stayhome; // use global stayhome
        ...
```

  - 只有使用多文件构成的程序时，内部和外部链接的区别才显得重要

```c
/* global1.c  -- uses an external variable */
#include <stdio.h>
external int units;          /* an external variable      */

void critic(void);
int main(void)
{
    extern int units;  /* an optional redeclaration */
    printf("How many pounds to a firkin of butter?\n");
    scanf("%d", &units);
    while ( units != 56)
        critic();
    printf("You must have looked it up!\n");
    return 0;
}
```

此处若对units赋初始值，则链接错误。

```c
/* global2.c  -- uses an external variable */
#include <stdio.h>
int units = 0;          /* an external variable      */

void critic(void)
{
    /* optional redeclaration omitted */
    printf("No luck, my friend. Try again.\n");
    scanf("%d", &units);
}
```

存储类说明符

- 存储类说明符
  - auto, register, static, extern和typedef
    - typedef与内存存储无关
  - 在一个声明类中只能使用一个上述说明符

```c
// parta.c --- various storage classes
// compile with partb.c
#include <stdio.h>
void report_count();
void accumulate(int k);
int count = 0;        // file scope, external linkage

int main(void)
{
    int value;        // automatic variable
    register int i;   // register variable

    printf("Enter a positive integer (0 to quit): ");
    while (scanf("%d", &value) == 1 && value > 0)
    {
        ++count;      // use file scope variable
        for (i = value; i >= 0; i--)
            accumulate(i);
```

```
        printf("Enter a positive integer (0 to quit): ");
    }
    report_count();
    return 0;
}

void report_count()
{
    printf("Loop executed %d times\n", count);
}
```

```
Enter a positive integer (0 to quit): 6↲
loop cycle: 1
subtotal: 21; total: 21
Enter a positive integer (0 to quit): 9↲
loop cycle: 2
subtotal: 45; total: 66
Enter a positive integer (0 to quit): a↲
Loop executed 2 times
```

```c
// partb.c -- rest of the program
// compile with parta.c
#include <stdio.h>
extern int count;        // reference declaration, external linkage
static int total = 0;    // static definition, internal linkage
void accumulate(int k); // prototype

void accumulate(int k)  // k has block scope, no linkage
{
    static int subtotal = 0;  // static, no linkage

    if (k <= 0)
    {
        printf("loop cycle: %d\n", count);
        printf("subtotal: %d; total: %d\n", subtotal, total);
        subtotal = 0;
    }
```

```
    else
    {
        subtotal += k;
        total += k;
    }
}
```

存储类和函数

- 外部函数（默认）

```
double gamma();
```

  - 可以在其它文件中使用该函数（应先声明后使用）

- 内部（静态）函数

```
static double gamma();
```

  - 只能在同一文件的范围内使用

- 函数如果不声明静态则为外部的（推荐用这个）

内容纲要

1	作用域、链接和存储时期
2	存储类
3	随机函数
4	伪随机数
5	动态内存分配

随机数

- 随机数
  - 专门的随机试验的结果
- 伪随机数：用算法计算出均匀分布的随机数序列
  - 但具有随机数的统计特征，如均匀性、独立性等。
- 伪随机数发生器（PRNG）
  - 种子为一个整数，用于确定伪随机数发生器的初始状态
  - 当前状态运算得到新伪随机数，随后更新为新状态

随机数

- 伪随机数发生器的种子
  - 常用当前机器时间作为种子，增加游戏的趣味性
  - 种子相同的发生器，生成的伪随机数序列相同
    - 经常用于测试时能重现上一次运行结果
- 相关函数

作用	原型	头文件
设置随机数种子	`void srand(unsigned int seed);`	`<stdlib.h>`
返回0~32767的随机数	`int rand(void);`	`<stdlib.h>`
获取当前时间	`time_t time(time_t *destTime);`	`<time.h>`

掷骰子

自制伪随机数发生器

```c
/* rand0.c -- produces random numbers                */
/*                uses ANSI C portable algorithm    */
static unsigned long int next = 1;  /* the seed  */

unsigned int rand0(void)
{
    /* magic formula to generate pseudorandom number */
    next = next * 1103515245 + 12345;
    return (unsigned int) (next/65536) % 32768;
}
```

```c
/* r_drive0.c -- test the rand0() function */
/* compile with rand0.c                    */
#include <stdio.h>
extern unsigned int rand0(void);

int main(void)
{
    int count;

    for (count = 0; count < 5; count++)
        printf("%d\n", rand0());

    return 0;
}
```

```
16838
5758
10113
17515
31051
```

```c
/* s_and_r.c -- file for rand1() and srand1()    */
/*                uses ANSI C portable algorithm */
static unsigned long int next = 1;  /* the seed  */

int rand1(void)
{
    /* magic formula to generate pseudorandom number */
    next = next * 1103515245 + 12345;
    return (unsigned int) (next/65536) % 32768;
}

void srand1(unsigned int seed)
{
    next = seed;
}
```

```c
/* r_drive1.c -- test rand1() and srand1() */
/* compile with s_and_r.c                  */
#include <stdio.h>
#include <stdlib.h>
extern void srand1(unsigned int x);
extern int rand1(void);

int main(void)
{
    int count;
    unsigned seed;

    printf("Please enter your choice for seed.\n");
    while (scanf("%u", &seed) == 1)
    {
        srand1(seed);    /* reset seed */
        for (count = 0; count < 5; count++)
            printf("%d\n", rand1());
```

```
        printf("Please enter next seed (q to quit):\n");
    }
    printf("Done\n");
    return 0;
}
```

```
Please enter your choice for seed.
23↲
26833
20599
12880
14586
14632
Please enter next seed (q to quit):
122↲
22657
5465
25328
17789
22663
Please enter next seed (q to quit):
q↲
Done
```

掷骰子

伪随机数发生器库函数

```c
/* diceroll.c -- dice role simulation */
/* compile with manydice.c           */
#include "diceroll.h"
#include <stdio.h>
#include <stdlib.h>             /* for library rand()   */

int roll_count  = 0;            /* external linkage     */

static int rollem(int sides)  /* private to this file */
{
    int roll;

    roll = rand() % sides + 1;
    ++roll_count;               /* count function calls */

    return roll;
}
```

```
int roll_n_dice(int dice, int sides)
{
    int d;
    int total = 0;
    if (sides < 2)
    {
        printf("Need at least 2 sides.\n");
        return -2;
    }
    if (dice < 1)
    {
        printf("Need at least 1 die.\n");
        return -1;
    }
    for (d = 0; d < dice; d++)
        total += rollem(sides);

    return total;
}
```

```c
//diceroll.h
extern int roll_count;

int roll_n_dice(int dice, int sides);
/* manydice.c -- multiple dice rolls                    */
/* compile with diceroll.c                              */
#include <stdio.h>
#include <stdlib.h>              /* for library srand() */
#include <time.h>                /* for time()          */
#include "diceroll.h"            /* for roll_n_dice()   */
/* and for roll_count  */
int main(void)
{
    int dice, roll;
    int sides;
    int status;

    srand((unsigned int) time(0)); /* randomize seed      */
    printf("Enter the number of sides per die, 0 to stop.\n");
```

```
while (scanf("%d", &sides) == 1 && sides > 0 )
{
    printf("How many dice?\n");
    if ((status =scanf("%d", &dice)) != 1)
    {
        if (status == EOF)
            break;              /* exit loop              */
        else
        {
            printf("You should have entered an integer.");
            printf(" Let's begin again.\n");
            while (getchar() != '\n')
                continue;      /* dispose of bad input */
            printf("How many sides? Enter 0 to stop.\n");
            continue;          /* new loop cycle         */
         }
    }
    roll = roll_n_dice(dice, sides);
```

```
        printf("You have rolled a %d using %d %d-sided dice.\n",
               roll, dice, sides);
        printf("How many sides? Enter 0 to stop.\n");
    }
    printf("The rollem() function was called %d times.\n",
           roll_count);                    /* use extern variable */

    printf("G

    return 0;
}
```

```
Enter the number of sides per die, 0 to stop.
4↲
How many dice?
6↲
You have rolled a 17 using 6 4-sided dice.
How many sides? Enter 0 to stop.
9↲
How many dice?
23↲
You have rolled a 114 using 23 9-sided dice.
How many sides? Enter 0 to stop.
0↲
The rollem() function was called 29 times.
GOOD FORTUNE TO YOU!
```

内容纲要

1	作用域、链接和存储时期
2	存储类
3	随机函数
4	伪随机数
5	动态内存分配

动态内存分配

- 分配内存（内存版本）：malloc()
  - 作用：在堆中找到指定大小的连续区域并返回其首地址
  - 原型：
    ```
    void *malloc(size_t num_bytes);
    ```
    - 第一个参数为需要开辟的字节数（不是元素个数）
    - 返回类型：void*表示未确定类型的指针
  - 用法示例
    ```
    p = (int *)malloc(sizeof(int) * 60);
    ```
- 包含头文件
  ```
  #include <malloc.h>
  ```

动态内存分配

- 注意事项
  - 应判断返回值是否为空（NULL）
    - 返回值如果为NULL，说明内存中没有符合条件的连续空间
  - 新开辟的空间应先置零
    - 头文件：<memory.h> 或 <string.h>

```
void *memset(void *dest, int c, size_t count);
```

  - 空间开辟后不再使用时应及时释放内存（free()函数）

动态内存释放

- 释放内存：free()
  - 作用：释放指定首地址开始的由malloc()开辟的内存空间
  - 原型：
```
void free(void *memblock);
```
    - 第一个参数为需要释放的地址（当时开辟的首地址）
  - 示例：
```
free(p);
```
- 注意事项
  - malloc开辟后的地址应妥善保存，不用时应前逐个释放
  - 重要性：开辟的空间如不及时释放将造成内存泄露

动态内存分配

- 分配内存（字符串版本）：calloc()
  - 返回的是char *型，而非返回void *型
  - 它将空间置零（有些系统，浮点数并非全零表示零）
- 释放内存（字符串版本）：cfree()

分配内存：malloc()和free()

- 分配内存：malloc()
  - 作用：在堆里找到一个大小合适的块并返回其内存首地址
  - calloc()：旧版返回的是char *型新版返回void*型
    - 它将空间置零（有些系统，浮点数并非全零表示零）
- 释放内存：free()
  - 作用：释放指定首地址开始的由malloc()开辟的内存空间
  - 重要性：如不释放将造成内存泄露

```c
/* dyn_arr.c -- dynamically allocated array */
#include <stdio.h>
#include <stdlib.h> /* for malloc(), free() */
int main(void)
{
    double * ptd;
    int max;
    int number;
    int i = 0;
    puts("What is the maximum number of type double entries?");
    if (scanf("%d", &max) != 1)
    {
        puts("Number not correctly entered -- bye.");
        exit(EXIT_FAILURE);
    }
    ptd = (double *) malloc(max * sizeof (double));
    if (ptd == NULL)
    {
        puts("Memory allocation failed. Goodbye.");
```

```
        exit(EXIT_FAILURE);
    }
    /* ptd now points to an array of max elements */
    puts("Enter the values (q to quit):");
    while (i < max && scanf("%lf", &ptd[i]) == 1)
        ++i;
    printf("Here are your %d entries:\n", number = i);
    for (i = 0; i < number; i++)
    {
        printf("%7.2f ", ptd[i]);
        if (i % 7 == 6)
            putchar('\n');
    }
    if (i % 7 != 0)
        putchar('\n');
    puts("Done.");
    free(ptd);
    return 0;
}
```

```
What is the maximum number of
type double entries?
52↲
Enter the values (q to quit):
35↲
99↲
2↲
q↲
Here are your 3 entries:
  35.00   99.00    2.00
Done.
```

分配内存：动态内存 vs 变长数组

- 动态内存分配
  - 程序员自己开，自己关
  - 开关未必要在同一个函数内
- 变长数组
  - 系统开，系统关
  - 不推荐使用
- 普通数组
  - 不能超过栈大小（默认8M）

存储类与动态内存分配

- 设想程序将内存分为三个部分
  - 存放外部链接、内部链接和空链接的静态变量
    - 编译时知道所需内存数量，整个程序期间可以用，程序结束时终止
  - 存放自动变量
    - 进入变量定义的代码块时产生，退出代码块时终止
    - 此内存为栈：先进先出。变量创建时顺序加入，消亡时反序消除
  - 存放动态分配的内存
    - 在内存中为碎片状，程序访问动态内存比堆栈内存要慢

```c
//  where.c  -- where's the memory?
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int static_store = 30;
const char * pcg = "String Literal";
int main()
{
    int auto_store = 40;
    char auto_string[] = "Auto char Array";
    int * pi;
    char * pcl;

    pi = (int *) malloc(sizeof(int));
    *pi = 35;
    pcl = (char *) malloc(strlen("Dynamic String") + 1);
    strcpy(pcl, "Dynamic String");
```

```
    printf("static_store: %d at %p\n", static_store,
&static_store);
    printf("  auto_store: %d at %p\n", auto_store,
&auto_store);
    printf("          *pi: %d at %p\n", *pi, pi);
    printf("  %s at %p\n", pcg, pcg);
    printf(" %s at %p\n", auto_string, auto_string);
    printf("  %s at %p\n", pcl, pcl);
    printf("   %s at %p\n", "Quoted String", "Quoted String");
    free(pi);
    free(pcl);
    return 0;
}
```

```
static_store: 30 at 0x601058
  auto_store: 40 at 0x7fff5848d8cc
          *pi: 35 at 0x2093010
  String Literal at 0x400834
 Auto char Array at 0x7fff5848d8e0
  Dynamic String at 0x2093030
   Quoted String at 0x4008a2
```

ANSI C的类型限定词

- 类型限定词const
  - 声明为常量
- 在文件中共享const数据要小心
  - 全局变量暴露了数据
  - 在一个文件使用const，另一个文件使用extern const
  - 将常量放置于头文件（.h）内
    - 应声明为static const

ANSI C的类型限定词

- 类型限定词volatile
  - 语法同const
  - 编译器优化：说明编译器应将volatile的变量放置于寄存器中，而非内存中
  - 允许const+volatile（顺序不重要）
    - 因为const不被程序改变，但仍可以被其他东西改变（例如：硬件时钟）

ANSI C的类型限定词

- 类型限定词restrict
  - 语法同const
  - 编译器优化：说明访问该指针使用唯一方式，于是编译器可以用等价语句进行合并
- 旧限定词新位置
  - 告诉编译器：可以用该信息优化代码
  - 告诉用户：只能用特定要求的参数

C 程 序 设 计

C Programming

12

谢谢观看

理论课程

厦門大學

XIAMEN UNIVERSITY

信息学院 黄 炜

（国家示范性软件学院） 博士，副教授

School of Informatics Dr. Wei Huang