When and where does it exist?

C语言程序设计

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

(特色化示范性软件学院) 博士,副教授

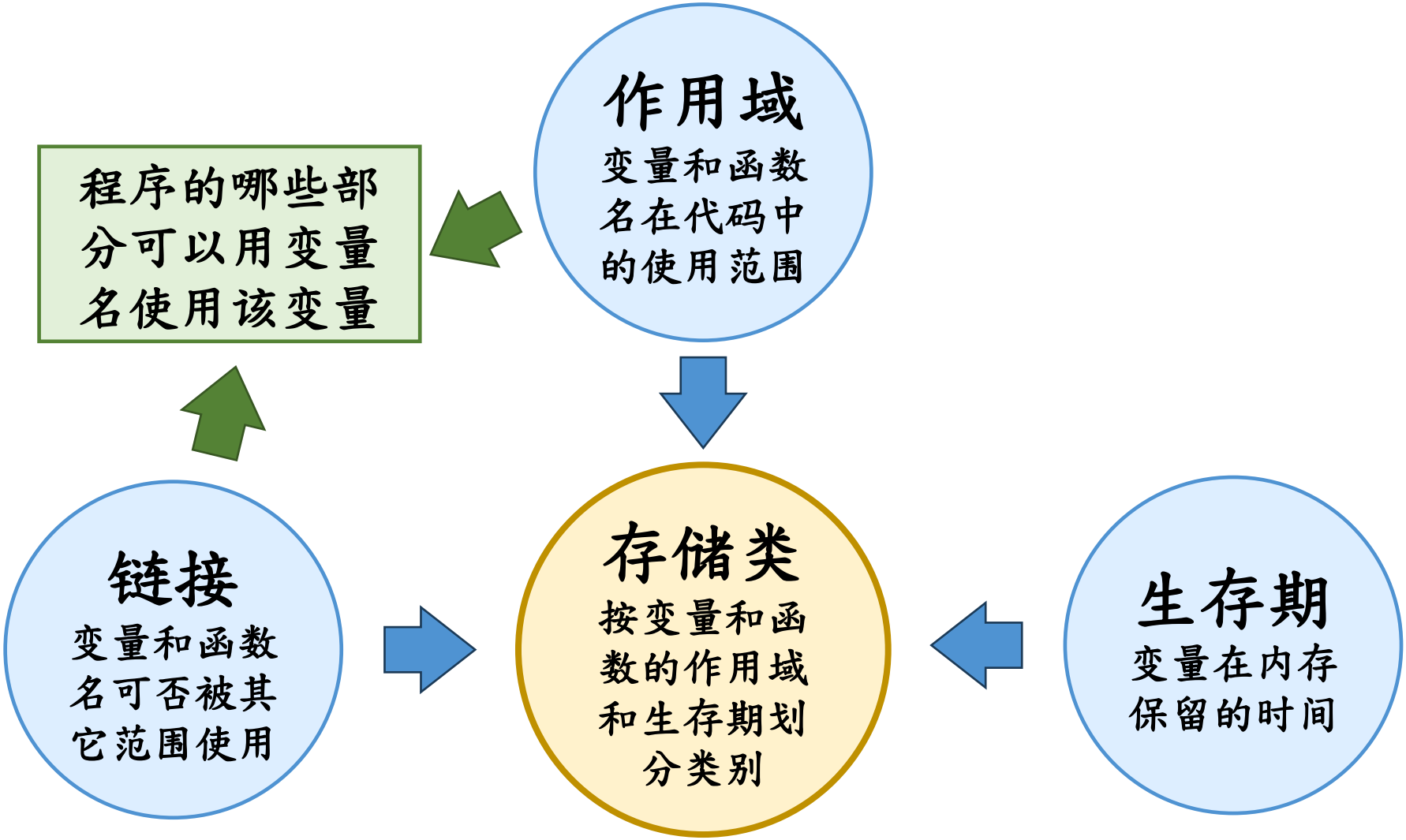
School of Informatics Dr. Wei Huang

内容要点

- 作用域
- 链接
- 存储时期
- 存储类
- 随机函数

目录

基本概念

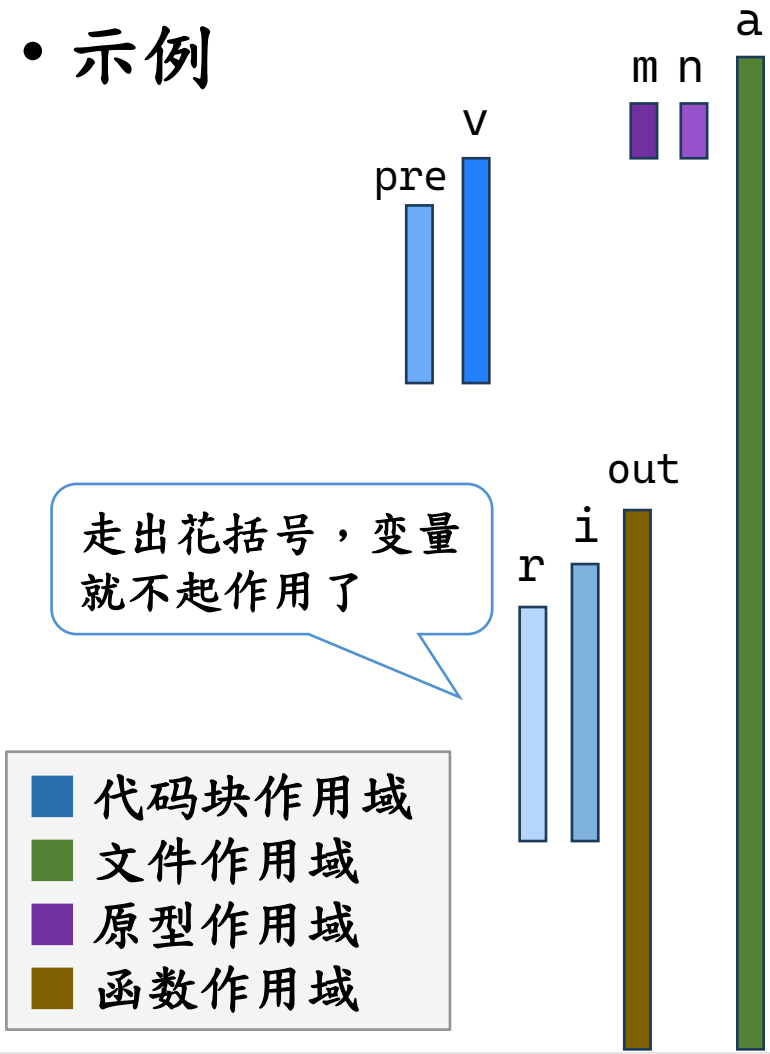

作用域
变量和函数名在代码中的使用范围
程序的哪些部分可以用变量名使用该变量
链接
变量和函数名可否被其它范围使用
存储类
按变量和函数的作用域和生存期划分类别
生存期
变量在内存保留的时间

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
- 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
- 函数原型作用域 (function prototype scope)
  - 函数声明语句中出现的参量，在声明语句内使用
- 函数作用域 (function scope)
  - 函数中的标签对于函数内部可见，外部不可goto进来

作用域

- 示例




```c
#include <stdio.h>
int a = 0;
void sum(int m, int n, int r[m][n]);
int amplify(int* v) {
    int pre = *v;
    *v *= a;
    return pre;
}
void sum(int x, int y, int b[x][y]) {
    // omitted...;
}
int main() {
    for (int i = 0; i < 10; i++) {
        int r = amplify(&i);
        a = a + i - r;
        if (i % 5 == 3)
            goto out;
    }
out:
    printf("%d", a);
    return 0;
}
```

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)
  - 外部链接的变量可以在多文件程序的任何地方使用
  - 文件作用域变量具有内部或外部链接
- 内部链接 (internal linkage)
  - 只能在当前文件的任何地方使用

链接

- 示例


```c
#include <stdio.h>
extern int s = 0;
void print() {
    printf("%d\n", s);
}
```

```c
#include <stdio.h>
static int a = 0;
int s = 0;
int amplify(int* v) {
    int pre = *v;
    *v *= a;
    return pre;
}
int main() {
    for (int i = 0; i < 10; i++) {
        int r = amplify(&i);
        a = a + i - r;
    }
    printf("%d", a);
    return 0;
}
```

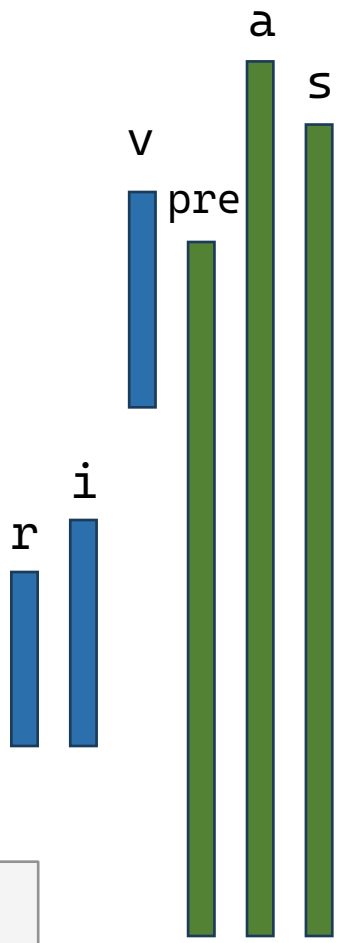
- 空 链 接：v, pre, i, r
- 外部链接：s
- 静态链接：a

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。

存储时期

- 示例




```
#include <stdio.h>
static int a = 0;
int s = 0;
int amplify(int* v) {
    static int pre = *v;
    *v *= a;
    return pre;
}
int main() {
    for (int i = 0; i < 10; i++) {
        int r = amplify(&i);
        a = a + i - r;
    }
    printf("%d", a);
    return 0;
}
```

自动存储时期
静态存储时期

目录

存储类

- 5种存储类

作用域	链接	自动生存期	静态生存期
代码块	空链接	1. 自动类 2. 寄存器类	3. 空链接的静态类
文件	内部链接	/	4. 具有内部链接的静态类
	外部链接	/	5. 具有外部链接的静态类

存储类	声明语句示例
1. 自动存储类	`auto int a;` （限函数内）
2. 寄存器存储类	`register int r;` （限函数内）
3. 空链接的静态存储类	`static int sn;` （限函数内）
4. 具有外部链接的静态存储类	`int e;` （限全局变量）
5. 具有内部链接的静态存储类	`static int si;` （限全局变量）

自动类

- 声明自动类变量
  - 位置：函数内，包括函数参量列表

说明	格式	示例
显式声明	auto <类型> <变量名>	auto int a;
隐式声明	<类型> <变量名>	int a;

  - 作用域：代码块；链接：空链接（代码块内）
  - 生存期：自动存储时期（函数结束时销毁）
- 规则
  - 内层定义名称与外层定义相同时，内层自动覆盖外层
  - 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: %d at %p\n", x, &x);
    }
    printf("x in outer blo
    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 = %d at %p\n", n, &n);
        n++;
    }
    printf("After loop 2,
    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
```

寄存器类

扩展内容

- 声明寄存器类变量
  - 位置：函数内，包括函数参量列表

说明	格式	示例
显式声明	register <类型> <变量名>	register int r;

  - 作用域：代码块；链接：空链接（代码块内）
  - 生存期：自动生存期（函数结束时销毁）
- 规则
  - 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类

- 声明代码块作用域的静态类变量
  - 位置：函数内，不包括函数参数列表

说明	格式	示例
显式声明	static <类型> <变量名>	static int r;

  - 作用域：文件；链接：空链接（代码块内）
  - 生存期：静态生存期（程序结束时销毁）

注意：静态是指内存位置的静态，不是值的静态

- 规则
  - 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
```

外部链接的静态类

扩展内容

- 声明外部链接的静态类变量
  - 位置：文件内，所有函数之外

说明	格式	示例
定义声明	<类型> <变量名>	int r;
引用声明	extern <类型> <变量名>	extern int r;

  - 作用域：文件；链接：外部链接（外部代码可用）
  - 生存期：静态生存期（程序结束时销毁）
- 规则
  - 在一个文件的全局变量区域做定义声明
  - 使用该全局变量的其他文件里做引用声明

```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;
}

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!
```

内部链接的静态类

扩展内容

- 声明内部链接的静态类变量
  - 位置：文件内，所有函数之外

说明	格式	示例
定义声明	static <类型> <变量名>	static int r;
引用声明	extern <类型> <变量名>	extern int r;

  - 作用域：文件；链接：内部链接（外部代码不可用）
  - 生存期：静态生存期（程序结束时销毁）
- 规则
  - 在一个文件的全局变量区域做定义声明
  - 限本文件该用到该全局变量的函数做引用声明（不必要）

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

自动	寄存器	静态	外部	类型定义
auto	register	static	extern	typedef

- typedef与内存存储无关
- 在一个声明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 pos
    }
    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

分类	外部函数	内部函数
声明格式	<类型> <函数名>(<参量列表>);	static <类型><函数名>(<参量列表>);
声明示例	double gamma();	static double gamma();
使用范围	其它文件可用	限同一文件范围使用
默认值	是	不是

目录

动态内存分配

malloc

功能	动态内存分配	
格式	void* malloc(size_t *size*);	
参数	*size*	无符号整数。开辟内存空间的长度，单位：字节。 可使用：sizeof(<类型>) * <数量>
返回值	成功	指向新分配内存开头的指针
	失败	NULL
示例	p = (int *)malloc(sizeof(int) * 6);	
头文件	stdlib.h	
说明	1. 如果*size*为0，则返回值未定义。 2. 指针不使用时，应使用free()释放，避免内存泄漏。	

动态内存分配

calloc

功能	动态内存分配，并初始化为0。	
格式	void* calloc(size_t *num*, size_t *size*);	
参数	*num*	对象数目。
	size	每个对象的长度，单位：字节。
返回值	成功	指向新分配内存开头的指针
	失败	NULL
示例	p = (int *)calloc(6, sizeof(int));	
头文件	stdlib.h	
说明	1. 如果*num*或*size*为0，则返回值未定义。 2. 因为对齐需求，分配的字节数不必等于 *num***size*。 3. 分配存储中的所有字节初始化为0。 4. 指针不使用时，应使用free()释放，避免内存泄漏。	

内存填充

memset

功能	内存填充	
格式	void *memset(void *dest, int ch, size_t count);	
参数	*dest*	指向要填充的对象的指针
	ch	填充字节值
	count	要填充的字节数
返回值	成功	指针*dest*的副本。
示例	memset(p, 0, 6);	
头文件	string.h	
说明	无。	

动态内存分配

- 动态内存分配后的操作
  - 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（memset）
  - 空间不再使用时应释放内存（free()）

动态内存释放

free

功能	归还之前已经分配的内存。	
格式	void free(void* *ptr*);	
参数	*ptr*	指针。之前由malloc()、calloc()、realloc()或aligned_alloc()分配的空间。
返回值	成功	无
示例	free(p);	
头文件	stdlib.h	
说明	1. 如果*ptr*为NULL，函数不进行操作；*ptr*不等于之前分配的任何空间，或该空间已归还，行为未定义。 2. 归还空间后，该空间被分配给其他变量前，通过该指针访问内存，行为未定义。	

```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

分类	内存管理方式	空间限制
动态内存	程序员手动开辟和释放	堆空间
变长数组	系统按生存周期管理	堆空间
定长数组（全局）	系统按生存周期管理	堆空间
定长数组（局部）	系统按生存周期管理	栈空间

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

扩展内容

- 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

扩展内容

- 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

扩展内容

- 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

目录

随机数

- 随机数：专门的随机试验的结果
- 伪随机数：用算法计算得到均匀分布的序列
  - 仅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：一个整数，用于确定伪随机数发生器的初始状态
    - 相同种子生成的随机数序列相同
    - 一般使用当前时间作为随机数种子，增加随机性
  - 状态：当前状态运算得到新伪随机数，随后更新为新状态
- 作用：测试程序增加多样性、游戏增加趣味性

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>`

设置伪随机数发生器的种子

srand

功能	设置伪随机数发生器的种子。	
格式	void srand(unsigned *seed*);	
参数	*seed*	无符号整数。种子值，用于决定初始状态。
返回值	成功	无
示例	srand(2024);	
头文件	stdlib.h	
说明	1. 随机数生成器应该只设置一次种子，即在程序开始处，在所有 rand() 调用前设置。不应重复播种。每次希望生成新一批随机数时，再重新设置种子。 2. 一般以 time(0) 的结果作为种子。 3. 相同种子在不同机器生成的随机数不同。	

获取伪随机数

rand

功能	获取伪随机数。	
格式	int rand();	
参数	无	
返回值	成功	0与RAND_MAX间包含边界的随机整数值。
示例	int r = rand();	
头文件	stdlib.h	
说明	1. srand()前调用rand()，其行为如同srand(1)后调用。 2. RAND_MAX 由实现定义，一般不少于32767。 3. 生成的伪随机数并无质量保证，不可用于严格的伪随机数生成需求，如加密。	

获取当前时间

time

功能	获取当前时间。	
格式	time_t time(time_t *arg);	
参数	*arg*	指针。指向存储时间对象的指针，或空指针。
返回值	成功	返回time_t对象的当前日历时间，如*arg*不为空，则同时存入*arg*。
	失败	(time_t)(-1)
示例	int t = time(0);	
头文件	time.h	
说明	1. 日历时间在time_t中的编码是未指定的，但多数系统遵循POSIX规定，返回保有从纪元开始至今秒数的整数类型值。 2. time_t为32位有符号整数，许多以往的实现会在 2038 年出错。	

```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;
}
```

掷骰子
自制伪随机数发生器

```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;
}
```

掷骰子
自制伪随机数发生器，以及留有设置种子的函数

```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 choic
    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
（此处省略数行）
14632
Please enter next seed (q to quit):
122↲
22657
（此处省略数行）
22663
Please enter next seed (q to quit):
q↲
Done
```

```c
/* diceroll.c -- dice role simulation */
/* compile with mandydice.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;
}
```

掷骰子
伪随机数发生器库函数

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

int roll_n_dice(int dice, int sides);
```

```c
/* 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 roll
           roll_coun
    printf("GOOD FOR
    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!
```

C语言程序设计

C Programming

12

谢谢观看

理论课程

厦門大學

XIAMEN UNIVERSITY

信息学院 黄 炜

(特色化示范性软件学院) 博士,副教授

School of Informatics Dr. Wei Huang