



第10章

程序结构

局部变量

- 在函数体内声明的变量：

```
int sum_digits(int n)
{
    int sum = 0;
    /* local variable */

    while (n > 0) {
        sum += n % 10;
        n /= 10;
    }


    return sum;
}
```

局部变量缺省性质-时空属性

- 时间：自动存储期限
 - 存储空间在函数被调用时“自动”分配，函数返回时自动回收。
 - **eg.** 教室、机房、机器
- 空间：块作用域
 - 作用域从变量声明处一直到所在函数体的末尾。
- C99：
 - 不要求在函数开始处声明变量，所以局部变量的作用域可能非常小：

```
void f(void)
{
```

- ```
 ...
 int i;
 ...
}
```



# 静态局部变量

- **static** 局部变量声明
- 静态存储期限：拥有永久的存储空间：
  - 存储空间及值在整个程序执行期间都会保留
  - eg, 宿舍
- **Example:**

```
void f(void)
{
 static int i;
 ...
}
```



下次调用f  
时i还在

- 仍然具有块作用域特性，对于其它函数是不可见的。

# 外部变量

- 声明在任何函数体外，也被称为全局变量。归属：整个程序
- 向函数传递信息的最常用方法：
  - 参数传递
- 还可通过外部变量与函数通信

# 函数访问局/外部变量

```
int m, n;
int average_global() {
 return (m+n)/2; }
int average(int a, int b) {
 return (a+b)/2; }
int main() {
 int x,y,z;
 //准备x,y,m,n等
 z=average(x,y);
 z=average_global();
 return 0;
}
```

# 外部变量性质

- 静态存储期限：
  - 空间及值永久保存
- 文件作用域：
  - 从被声明的点一直到所在文件的末尾。



# 用外部变量实现栈

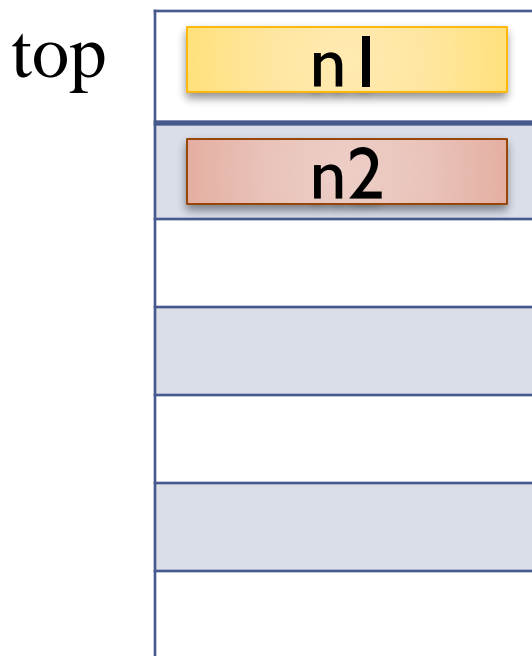
- 栈结构上类似数组
  - 存储相同类型的多个数据项。
- 但操作受限：
  - 只能访问栈顶：不能测试或修改不在栈顶的数据项，
  - **filo**：先入后出，弹夹，电梯
  - **fifo**：先入先出，打饭队列，自动扶梯：队列**queue**
  - 压栈（**Push**）：在栈顶加入一个数据项
  - 出栈（**Pop**）：从栈顶取走（删除）一个数据项





# 示例：用外部变量实现栈

- `int contents[N], top=0;`
- 访问：用`top`访问，后移动`top`
  - `push: contents[top++];`
  - `pop: contents[--top]`



```
push n1: contents[top]=n1;
 top++;
push n2: contents[top]=n2;
 top++;
pop n2: top--;
 a=contents[top];
pop n1: top--;
 a=contents[top];
```

# 温故而知新——变量总结

- 局部变量
  - 自动存储期限:
  - 块作用域:
- 静态局部变量
  - 静态存储期限
  - 块作用域
- 外部变量
  - 静态存储期限:
  - 文件作用域



# 用栈跟踪程序

- 堆栈记录函数活动记录-函数返回地址及函数执行环境

```
int main(void) {
```

```
...
```

```
 a();
```

```
...}
```

```
void a(void) {
```

```
...
```

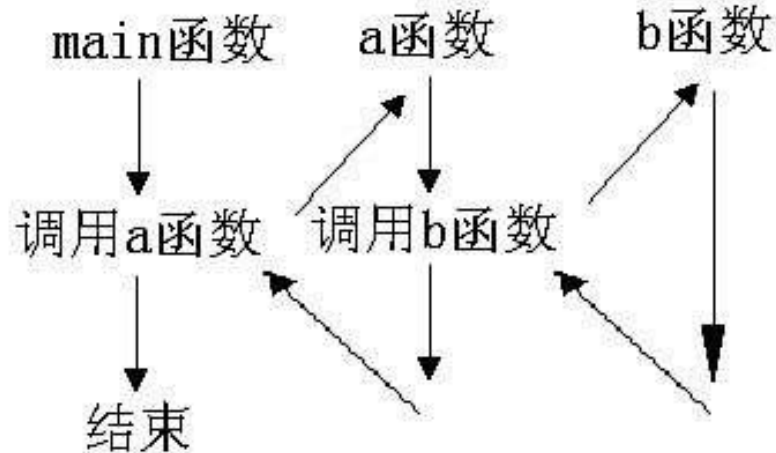
```
 b();
```

```
...}
```

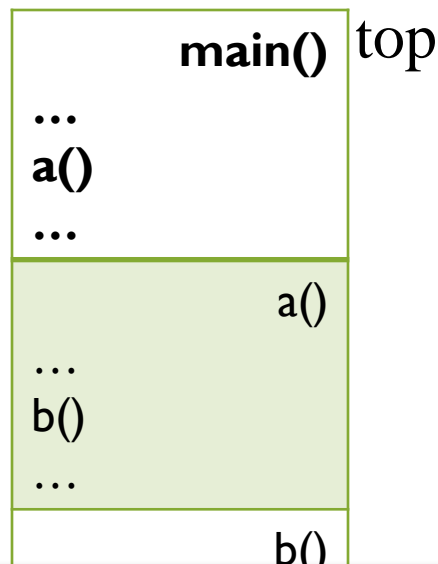
```
void b(void) {
```

```
...
```

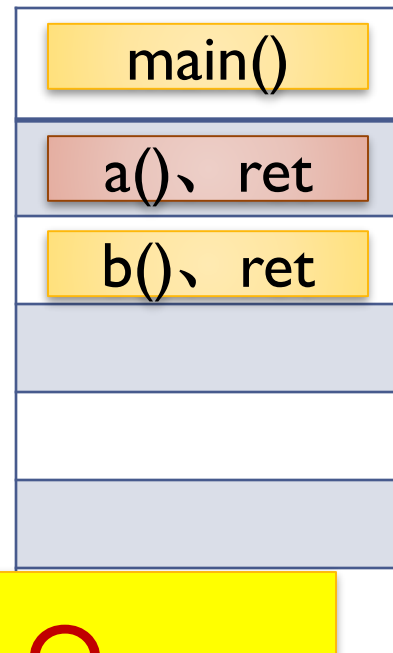
```
...}
```



pc



stack



函数返回地址：FILO

# 示例：用外部变量实现栈

- 栈处理：
  - 压、弹、清空、判空,判满等
  - 用函数实现，都需以栈及栈顶为参数
- 定义栈及栈顶为外部变量
  - 所有栈处理函数都可以见
  - 减少参数传递

```
#define STACK_SIZE 100 //堆栈长度
int contents[STACK_SIZE]; //外部声明堆栈
int top = 0;
void make_empty(void) //清空
{
 top = 0;
}
bool is_empty(void) //判空
{
 return top == 0;
}
bool is_full(void) //判满
{
 return top == STACK_SIZE;
}
```

```
void push(int i) //压入i
{
 if (is_full())
 stack_overflow();
 else
```

如果堆栈不在外部声明:

```
void make_empty(int content[], int len, int top);
bool is_empty(int content[], int len, int top);
bool is_full(int content[], int len, int top);
void push(int i, int content[], int len, int top);
int pop(int content[], int len, int top);
```

```
}
```

# 外部变量的利与弊

- 利：

- 多个函数共享一个变量，避免都传参
- 少数几个函数需要共享大量变量，少传参

- 弊：

- 改变外部变量（类型），需确定每个使用其的函数，以确认影响
- 外部变量被赋了错误的值，可能很难确定出错的函数
- 很难在其它程序中复用依赖于外部变量的函数

- 多数情况下，函数间通信：参数传递比外部变量共享更好



# 外部变量的利与弊

- 避免在不同函数中为不同的目的使用同一个外部变量，如：
  - 几个函数都用变量*i*控制for循环，  
正确做法：函数各自声明局部*i*
  - 程序顶部声明（外部变量），容易产生误导：这些变量*i*的使用是彼此相关
- 确保外部变量名有意义；
  - 局部变量名并不总有意义，如for循环控制变量*i*。

# 外部变量的利与弊

- 把应为局部的变量声明为外部变量可能导致的错误。如，显示10 × 10 星形程序：

```
int i;
void print_one_row(void) {
 for (i = 1; i <= 10; i++)
 printf("*");
}
void print_all_rows(void) {
 for (i = 1; i <= 10; i++) {
 print_one_row();
 printf("\n");
 }
}
```

i=1 调用print\_one\_row  
，退出后i=11

- 结果print\_all\_rows仅输出一行星号，两个for公用一个i

# 程序：猜数游戏

- 自动产生一个1-100间的随机数，提示用户尝试用尽可能少的次数猜出这个数：

```
Enter guess: 55
Too low; try again.
Enter guess: 65
Too high; try again.
Enter guess: 60
Too high; try again.
Enter guess: 58
You won in 4 guesses!
```

```
Play again? (Y/N) y
```

```
A new number has been chosen.
```

```
Enter guess: 78
Too high; try again.
Enter guess: 34
You won in 2 guesses!
```

```
Play again? (Y/N) n
```

# 程序：猜数游戏

- 程序需完成如下几个任务：
  - 初始化随机数发生器
  - 秘密选择一个随机数
  - 通过交互让用户猜值
- 上述任务各通过一个函数处理
- 秘密数声明为外部变量，方便后两函数处理

# guess.c

```
/* Asks user to guess a hidden number */
```

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

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

```
#include <time.h>
```

```
#define MAX_NUMBER 100
```

```
/* external variable */
```

```
int secret_number;
```

出入通道

```
/* prototypes */
```

```
void initialize_number_generator(void);
```

```
void choose_new_secret_number(void);
```

```
void read_guesses(void);
```

出通道

入通道

```
int main(void)
{
 char command;
 printf("Guess the secret number between 1
and %d.\n\n", MAX_NUMBER);
 initialize_number_generator();

 do{
 choose_new_secret_number();
 printf("A new number has been chosen.\n");
 read_guesses();
 printf("Play again? (Y/N) ");
 scanf(" %c", &command);
 printf("\n");
 } while (command == 'y' || command == 'Y');

 return 0;
}
```

```
void initialize_number_generator(void)
{
 srand((unsigned) time(NULL));
}
```

```
void choose_new_secret_number(void)
{
 secret_number = rand() % MAX_NUMBER + 1;
}
```



```
void read_guesses(void)
{
 int guess, num_guesses = 0;

 for (;;) {
 num_guesses++;
 printf("Enter guess: ");
 scanf("%d", &guess);
 if (guess == secret_number) {
 printf("You won in %d guesses!\n\n",
num_guesses);
 return;
 } else if (guess < secret_number)
 printf("Too low; try again.\n");
 else
 printf("Too high; try again.\n");
 }
}
```

# 程序块

- 复合语句:

`{ statements }`

- 包含声明的复合语句:

`{ declarations  
statements }`

- 这种类型的复合语句为：程序块

- 大括号包含的自带声明和语句的程序段

# 程序块

- 程序块中声明的变量，程序块局部变量：
  - 自动存储期限：进入程序块分配，退出程序块收回。
  - 具有块作用域，不能在程序块外被引用。
- 程序块中的变量可以被声明为 `static`，使之具有静态存储期限。

# 程序块

- 函数体、选择分支或循环体都是一个程序块
- 在函数体内部，构造更小程序块，在其中声明变量，其好处：
  - 避免只在小范围或临时使用的变量与其它变量混淆；
  - 减少名字冲突：可在不同程序块中使用相同标识符（同名的变量）；
  - 避免函数体起始位置的声明与只是临时使用的变量相混淆；
  - 减少名字冲突
- C99允许在程序块的任何位置声明变量

# 作用域

- 作用域：标识符作用范围
- 相同（同名）标识符可以有不同的含义
- 作用域规则确定同名标识符含义：
  - 最新声明规则（覆盖）：在程序块内声明标识符时，如果此标识符已经可见（存在），新声明会临时“覆盖”（隐藏）旧声明
  - 在程序块结束后，旧标识符重新恢复到以前的含义。
  - 类比，指定C班长

# 作用域规则

- 声明1：静态存储期限和文件作用域；
- 声明2：块作用域形式参数；
- 声明3：块作用域自动变量；
- 声明4：块作用域自动变量。

```
int i ; /* Declaration 1 */

void f(int i) /* Declaration 2 */
{
 i = 1;
}

void g(void)
{
 int i = 2; /* Declaration 3 */
 if (i > 0) { /* Declaration 4 */
 int i ;
 i = 3;
 }
 i = 4;
}

void h(void)
{
 i = 5;
}
```

# 构建C程序（单文件程序）

- C程序的构成要素：
  - 预处理指令，如： `#include` 或 `#define`
  - 类型定义//`typedef` .....
  - 外部变量声明
  - 函数原型
  - 函数定义
- 顺序没有过多限制
  - 预处理指令执行到才起作用
  - 类型名定义后才允许使用
  - 变量在声明后才可以使用
  - 函数调用前声明或定义：C99强制规定



# 构建C程序（单文件程序）

- 构建程序常用顺序：
  - `#include` 指令
  - `#define` 指令
  - 定义类型 `typedef int bool`
  - 声明外部变量
  - 声明除 `main` 之外的函数原型
  - 定义 `main` 函数
  - 定义其它函数

# 构建C程序

- 在函数定义前加一个注释框：好编程习惯：
  - 函数名
  - 定义该函数的目的
  - 每个参数的含义
  - 如果有返回值，应当给出描述信息
  - 列出函数的副作用（如修改了外部变量的值）

# 程序：给一手牌分类

- 读取5张牌作为一手，按照给出的规则进行分类（优先级从高到低）
  - 同花顺（5张牌花色相同且牌的分值顺序相连）
  - 四条（四张牌具有相同分值）
  - 葫芦（三张牌同分加一对）
  - 同花（5张牌花色相同）
  - 顺子（5张牌的分值顺序相连）
  - 三条（三张牌具有相同分值）
  - 两对
  - 一对（两张牌具有相同分值）
  - 高分（其它类型的牌型）

# 程序：给一手牌分类

- 扑克表示

- 花色：梅花clubs、方块diamonds、红桃hearts或黑桃spades
- `const char suit_code[] = {'c', 'd', 'h', 's'};`
- 点数：2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K或A)
- `const char rank_code[] = {'2', '3', '4', '5', '6', '7', '8', '9', 't', 'j', 'q', 'k', 'a'};`

- 给牌：用户输入花色、点数

- 非法或输入同一张牌两次，忽略该输入，提示出错，请求用户重新输入
- 如果用户输入0，则程序终止

# 程序：给一手牌分类

- 与程序的会话示例如下：

```
Enter a card: 2s
Enter a card: 5s
Enter a card: 4s
Enter a card: 3s
Enter a card: 6s
Straight flush
Enter a card: 8c
Enter a card: as
Enter a card: 8c
Duplicate card; ignored.
Enter a card: 7c
Enter a card: ad
Enter a card: 3h
Pair
```

# 程序：给一手牌分类

```
Enter a card: 6s
Enter a card: d2
Bad card; ignored.
Enter a card: 2d
Enter a card: 9c
Enter a card: 4h
Enter a card: ts
High card
```

```
Enter a card: 0
```

# 程序：给一手牌分类

- 该程序完成如下三个任务：
  - 读入一手5张牌： `read_cards`
  - 分析对子、顺子等情况：  
`analyze_hand`
  - 显示对该手牌的分类信息：  
`print_result`
- 函数实现上面具体任务，`main`循环调用上述函数，怎么循环
  - 输入0结束，`read_cards`读入，在其中退出程序，`how?`
  - `main`中无限循环： `for(;;)`



# 程序：给一手牌分类

- 函数间需共享大量信息，外部变量实现函数间通信
  - `read_cards`将牌相关信息存储外部变量
  - `analyze_hand`检查外部变量，并分析
  - 结果存储到另外的外部变量，便于 `print_result`使用。

# 程序概览——编写程序框架

```
/* #include directives go here */

/* #define directives go here */

/* declarations of external variables go here */

/* prototypes */
void read_cards(void);
void analyze_hand(void);
void print_result(void);
```

# 程序：给一手牌分类

```
/******
 * main: Calls read_cards, analyze_hand, and
 * print_result repeatedly. _
******/
int main(void)
{
 for (;;) {
 read_cards();
 analyze_hand();
 print_result();
 }
}
/******
 * read_cards: Reads the cards into external variables;
 * checks for bad cards and duplicate cards. *
******/
void read_cards(void)
{
 ...
}
```

# 程序：给一手牌分类

```
/* **** */
* analyze_hand: Determines whether the hand contains a *
* straight, a flush, four-of-a-kind, *
* and/or three-of-a-kind; determines the *
* number of pairs; stores the results into *
* external variables. *
* **** */
void analyze_hand(void)
{
 ...
}

/* **** */
* print_result: Notifies the user of the result, using *
* the external variables set by *
* analyze_hand. *
* **** */
void print_result(void)
{
 ...
}
```

# 程序：给一手牌分类

- 怎么分析？

| 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | J | Q  | K  | A  |
|---|---|---|---|---|---|---|---|----|---|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8  | 9 | 10 | 11 | 12 |
|   |   |   |   |   |   |   |   |    |   |    |    |    |

- 需要两个数组：num 和 rank[13]

| C | D | H | S |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
|   |   |   |   |

的数量  
的数量

- 分值编码：0到12

- '2' '3' '4' '5' '6' '7' '8' '9' '10' 'J' 'Q' 'K' 'A'

| 'c' | F   | F | F | F | F | F | ... |  |  |  |  |  |  |
|-----|-----|---|---|---|---|---|-----|--|--|--|--|--|--|
| 'd' | ... |   |   |   |   |   |     |  |  |  |  |  |  |
| 'h' |     |   |   |   |   |   |     |  |  |  |  |  |  |
| 's' |     |   |   |   |   |   |     |  |  |  |  |  |  |

# 程序：给一手牌分类

- `read_cards`和`analyze_hand`都需要访问
  - `num_in_rank`和`num_in_suit`
  - 外部变量。
- `card_exists`仅被`read_cards`函数使用
  - 局部变量。
- 规则：
  - 仅在必要时才将变量声明为外部变量
    - 函数间共享的数据项多，数据量大

# poker.c

```
/* Classifies a poker hand */
```

```
#include <stdbool.h> /* C99 only */
```

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

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

```
#define NUM_RANKS 13
```

```
#define NUM_SUITS 4
```

```
#define NUM_CARDS 5
```

```
/* external variables */
```

```
int num_in_rank[NUM_RANKS] = {0};
```

```
int num_in_suit[NUM_SUITS] = {0};
```

```
bool straight, flush, four, three;
```

```
int pairs; /* can be 0, 1, or 2 */
```



```
/* prototypes */
void read_cards(void) ;
void analyze_hand(void) ;
void print_result(void) ;

/*****
 * main: Calls read_cards, analyze_hand, and print_result *
 * repeatedly. *
 *****/
int main(void)
{
 for (;;) {
 read_cards() ;
 analyze_hand() ;
 print_result() ;
 }
}
```

```

/*****
* read_cards: Reads the cards into the external *
* variables num_in_rank and num_in_suit; *
* checks for bad cards and duplicate cards. *
*****/
void read_cards(void)
{
 bool card_exists[NUM_RANKS][NUM_SUITS] =
 {false};
 char ch, rank_ch, suit_ch; //点数、花色
 int rank, suit; //数组下标, 记录点数、花色
 bool bad_card; //记录多种无效牌
 int cards_read = 0; //有效牌数

```

```

while (cards_read < NUM_CARDS) {
 bad_card = false;
 printf("Enter a card: ");
 rank_ch = getchar();
 switch (rank_ch) {
 case '0': exit(EXIT_SUCCESS);
 case '2': rank = 0; break;
 case '3': rank = 1; break;
 case '4': rank = 2; break;
 case '5': rank = 3; break;
 case '6': rank = 4; break;
 case '7': rank = 5; break;
 case '8': rank = 6; break;
 case '9': rank = 7; break;
 case 't': case 'T': rank = 8; break;
 case 'j': case 'J': rank = 9; break;
 case 'q': case 'Q': rank = 10; break;
 case 'k': case 'K': rank = 11; break;
 case 'a': case 'A': rank = 12; break;
 default: bad_card = true;
 }
}

```

```

suit_ch = getchar();
switch (suit_ch) {
 case 'c': case 'C': suit = 0; break;
 case 'd': case 'D': suit = 1; break;
 case 'h': case 'H': suit = 2; break;
 case 's': case 'S': suit = 3; break;
 default: bad_card = true;
}
while ((ch = getchar()) != '\n')
 if (ch != ' ') bad_card = true;
 //过滤牌后无效输入
if (bad_card)
 printf("Bad card; ignored.\n");
else if (card_exists[rank][suit])
 printf("Duplicate card; ignored.\n");
else {
 num_in_rank[rank]++;
 num_in_suit[suit]++;
 card_exists[rank][suit] = true;
 cards_read++;
}
}
}

```

```

void analyze_hand(void)
{
 int num_consec = 0; //存连牌张数
 int rank, suit;
 straight = false;
 flush = false;
 four = false;
 three = false;
 pairs = 0;
 /* check for flush */
 for (suit = 0; suit < NUM_SUITS; suit++)
 if (num_in_suit[suit] == NUM_CARDS)
 flush = true;

```

| C | D | H | S |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
|   | 5 |   |   |

```

/* check for straight */
rank = 0;
while (num_in_rank[rank] == 0) rank++;
 //扫描第一张牌（最低点数）
for (; rank < NUM_RANKS && num_in_rank[rank] > 0;
rank++)
 num_consec++; //统计连续出现的点数
if (num_consec == NUM_CARDS) {
 straight = true;
 return;
}
/* check for 4-of-a-kind, 3-of-a-kind, and pairs */
for (rank = 0; rank < NUM_RANKS; rank++) {
 if (num_in_rank[rank] == 4) four = true;
 if (num_in_rank[rank] == 3) three = true;
 if (num_in_rank[rank] == 2) pairs++;
}
}

```

| 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | J | Q  | K  | A  |
|---|---|---|---|---|---|---|---|----|---|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8  | 9 | 10 | 11 | 12 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0  | 0 | 0  | 0  | 0  |

```

/*****
* print_result: Prints the classification of the hand,
* based on the values of the external
* variables straight, flush, four, three,
* and pairs.
*****/
void print_result(void)
{
 if (straight && flush) printf("Straight flush");
 else if (four) printf("Four of a kind");
 else if (three &&
 pairs == 1) printf("Full house");
 else if (flush) printf("Flush");
 else if (straight) printf("Straight");
 else if (three) printf("Three of a kind");
 else if (pairs == 2) printf("Two pairs");
 else if (pairs == 1) printf("Pair");
 else printf("High card");

 printf("\n\n");
}

```