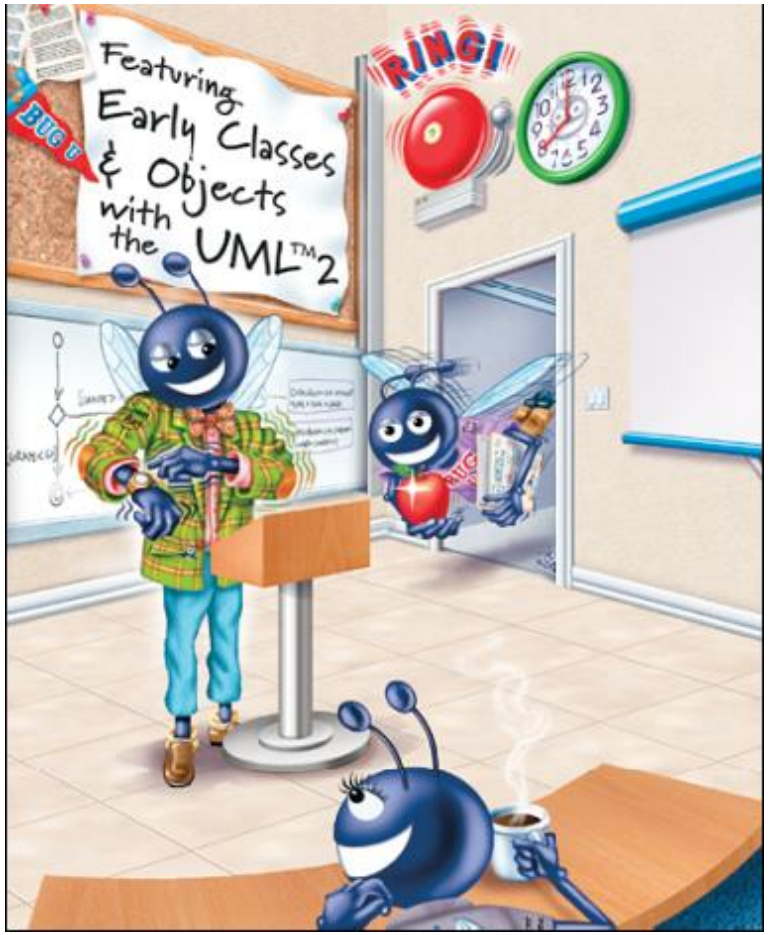
C++程序设计



上节课内容回顾

1. 声明数组、初始化数组、引用数组中的元素
2. 将数组传递给函数
3. 多维数组
4. 使用C++标准库类模板 vector

第七讲 指针和基于指针的字符串

学习目标：

- 指针和引用的异同
- 指针作为参数传递给函数
- 基于指针的 C 风格的字符串
- 指针和数组的关系
- 函数指针

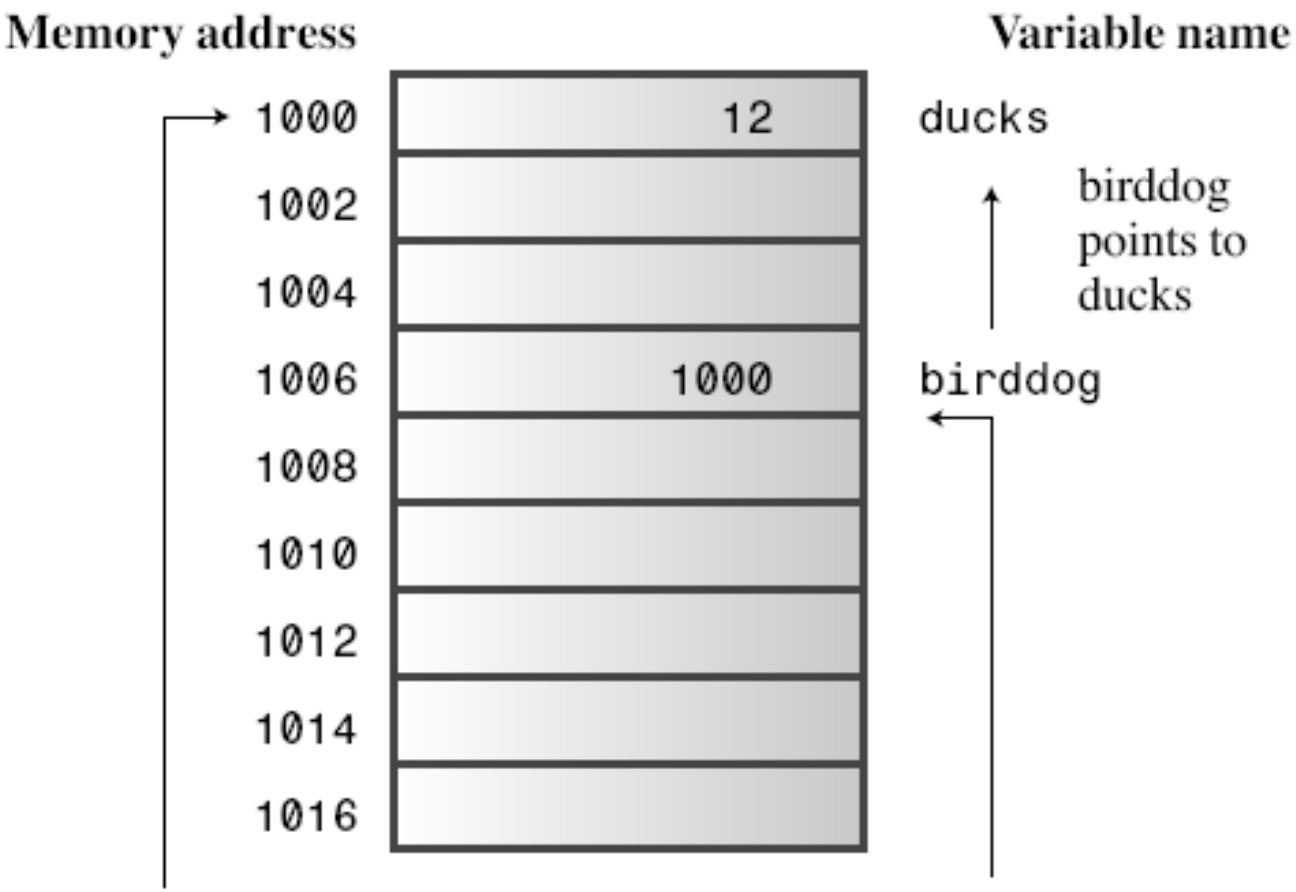
1. Introduction

- **Pointers**
  - 功能强大但难于掌握
  - 可以用来执行 pass-by-reference
  - 可以用来创建和操纵动态数据结构
  - 与数组和字符串有着密切的关系

2. Pointer Variable Declarations and Initialization

- **Pointer variables**
  - 将内存地址作为变量值
    - 通常变量包含特定的值 (直接引用)
    - 指针包含变量的地址值 (间接引用)
- **Indirection**
  - 通过指针来引用变量的值

2. Pointer Variable Declarations and Initialization




```
int ducks = 12;
```

creates ducks variable, stores the value 12 in the variable

```
int *birddog = &ducks;
```

creates birddog variable, stores the address of ducks in the variable

2. Pointer Variable Declarations and Initialization

- **Pointer declarations**
  - * 指示一个变量为指针
    - Example : int *myPtr; //指向 int 变量的指针
    - 多个指针变量的声明：int *myPtr1, *myPtr2;
- **Pointer initialization**
  - 初始化为 0, NULL 或一个地址

2. Pointer Variable Declarations and Initialization

常见编程错误：声明指针变量时，每个指针变量前面都必须加“*”。

良好编程习惯：在指针变量名中包含字母 Ptr能够更清楚地表示这个变量是指针变量。

错误预防技巧：指针初始化是为了防止指向未知的和未经初始化的内存区域。

3. Pointer Operators

- 地址运算符 (&)
  - 返回操作数的地址
  - int y = 5;
    int *yPtr;
    yPtr = &y; //变量 yPtr “指向” y

3. Pointer Operators

- * operator
  - 也称为 indirection operator or dereferencing operator
  - *yPtr 返回 y (因为 yPtr 指向 y)
  - Dereferenced pointer is an *lvalue*
    - *yPtr = 9;

4. Passing Arguments to Functions by Reference with Pointers

- 三种向函数传递参数的方式
  - Pass-by-value
  - Pass-by-reference with reference arguments
  - Pass-by-reference with pointer arguments

4. Passing Arguments to Functions by Reference with Pointers

- 一个函数只能返回一个值
- 使用引用参数向函数传递参数
  - 函数可以修改参数的原始值
    - 可以返回“多个值”

4. Passing Arguments to Functions by Reference with Pointers

- 使用指针参数按引用传递
  - 使用 & 运算符传递参数地址
  - 数组名为数组首地址，故无须再加“&”
  - * 运算符在函数内部用做参数的别名来使用

```
void cubeByReference( int * );

int main()
{
  int number = 5;
  .......
  cubeByReference( &number );
  .......
  return 0;
}

void cubeByReference( int *nPtr )
{
  *nPtr = *nPtr * *nPtr * *nPtr;
}
```

5. Using const with Pointers

- **Principle of least privilege (最低权限原则)**
  - 授予函数足够的权限来完成任务
  - 例如：打印数组元素的函数
    - 数组元素应为 const
    - 数组长度应为 const

5. Using const with Pointers

- **const pointers**
  - Constant pointer to a non-constant int
    - int *const myPtr = &x;
  - Non-constant pointer to a constant int
    - const int *myPtr = &x;
  - Constant pointer to a constant int
    - const int *const Ptr = &x;

5. Using const with Pointers

```
……
// ptr is a constant pointer to an integer that can
// be modified through ptr, but ptr always points to the
// same memory location.
int * const ptr = &x; // const pointer must be initialized

*ptr = 7; // allowed: *ptr is not const
ptr = &y; // error: ptr is const; cannot assign to it a new address
……
```

5. Using const with Pointers

```
// xPtr cannot modify the value of constant variable to which it points
void f( const int *xPtr )
{
   *xPtr = 100; // error: cannot modify a const object
}
```

5. Using const with Pointers

```
……
int x = 5, y;

// ptr is a constant pointer to a constant integer.
// ptr always points to the same location; the integer
// at that location cannot be modified.
const int *const ptr = &x;
*ptr = 7; // error: *ptr is const; cannot assign new value
ptr = &y; // error: ptr is const; cannot assign new address
……
```

6. Pointer Expressions and Pointer Arithmetic

- **Pointer assignment**
  - 同一类型之间的指针可以相互赋值
    - 如果为不同类型，需要使用类型转换运算符
    - 例外：void *（代表任何类型）
      - 无须将指针转换为void *
      - 需要将void *转换为其他类型
      - void 指针不能被 dereferenced

7. Relationship Between Pointers and Arrays

- 数组与指针密切相关
  - 数组名为 constant 指针
  - 指针可以用来进行数组的索引操作

7. Relationship Between Pointers and Arrays

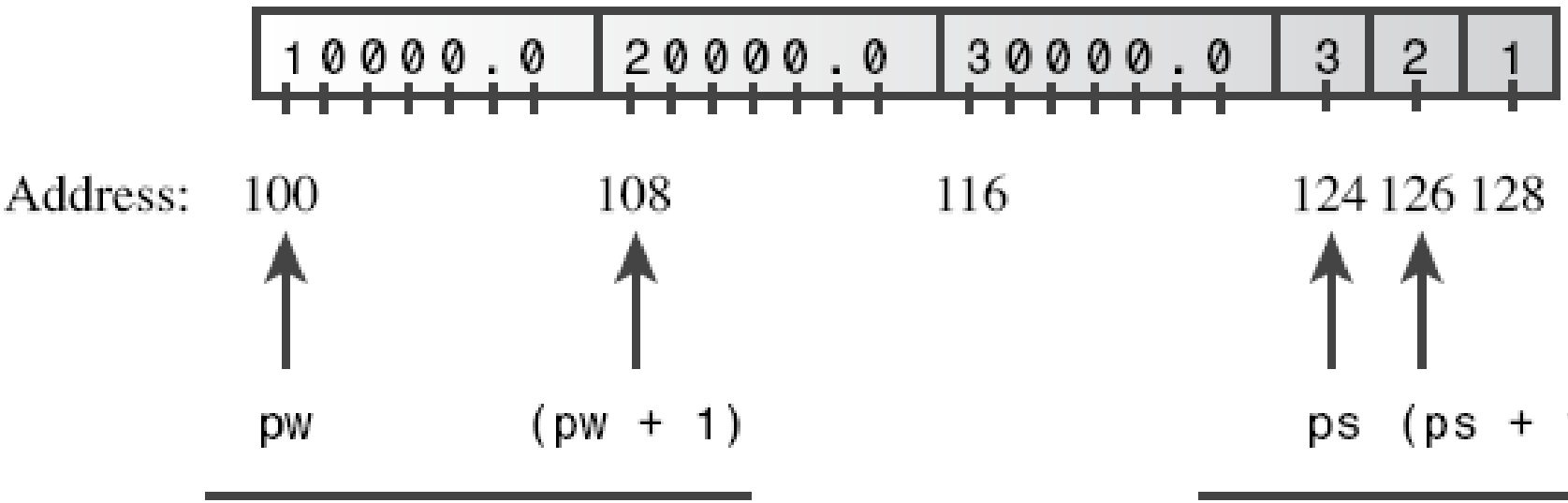
- 使用指针访问数组元素
  - int b[5];
    int *bPtr;
    bPtr = b;
  - b[n] = = *(bPtr + n)
  - &b[3] = = bPtr + 3
  - b[3] = = *(b + 3)
  - b[3] = = bPtr[3]

7. Relationship Between Pointers and Arrays

常见编程错误：尽管数组名是指向数组开头的指针，并且指针可在算术表达式中修改，但是数组名不可以在算术表达式中修改，因为数组名实际上是个常量指针。

7. Relationship Between Pointers and Arrays

```
double wages[3] = {10000.0, 20000.0, 30000.0};
short stacks[3] = {3, 2, 1};
double * pw = wages;
short * ps = &stacks[0];
```



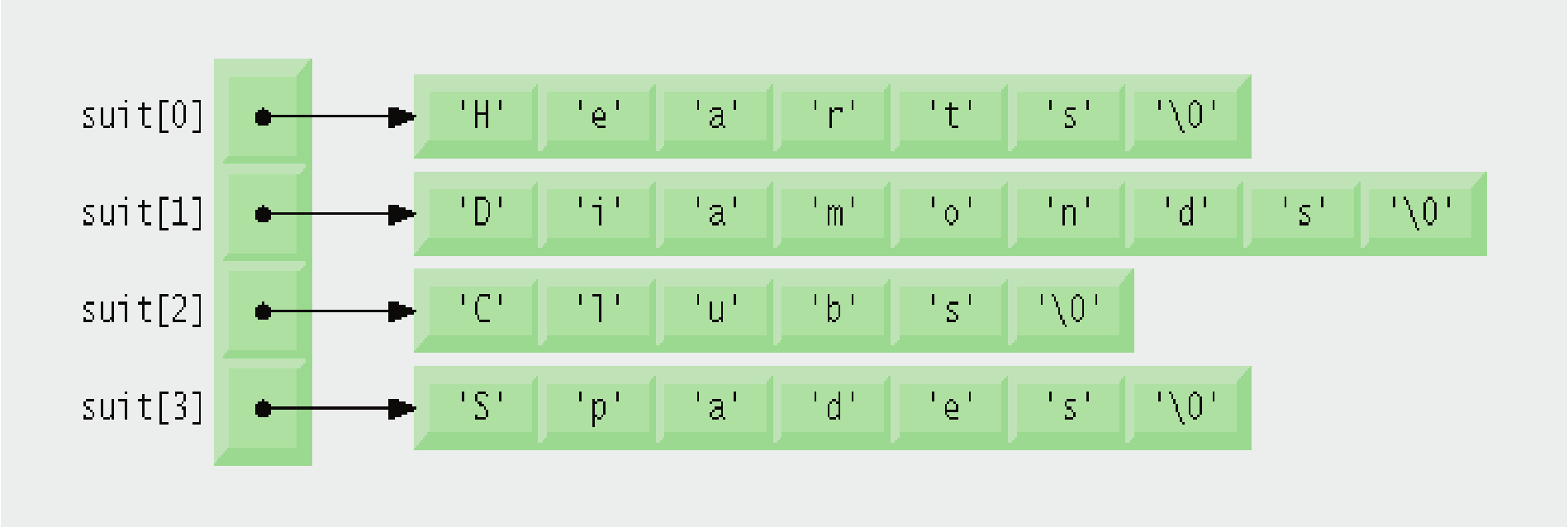
pw points to type *double*, so adding 1 to *pw* changes its value by 8 bytes.

ps points to type *short*, so adding 1 to *ps* changes its value by 2 bytes.

8. Arrays of Pointers

- 数组中可包含指针
  - 通常用来存储字符串数组
    - 例如：
      - const char *suit[4] =
        { "Hearts", "Diamonds", "Clubs", "Spades" };
    - suit 数组元素具有固定长度，但其指向的字符串可以为任意长度

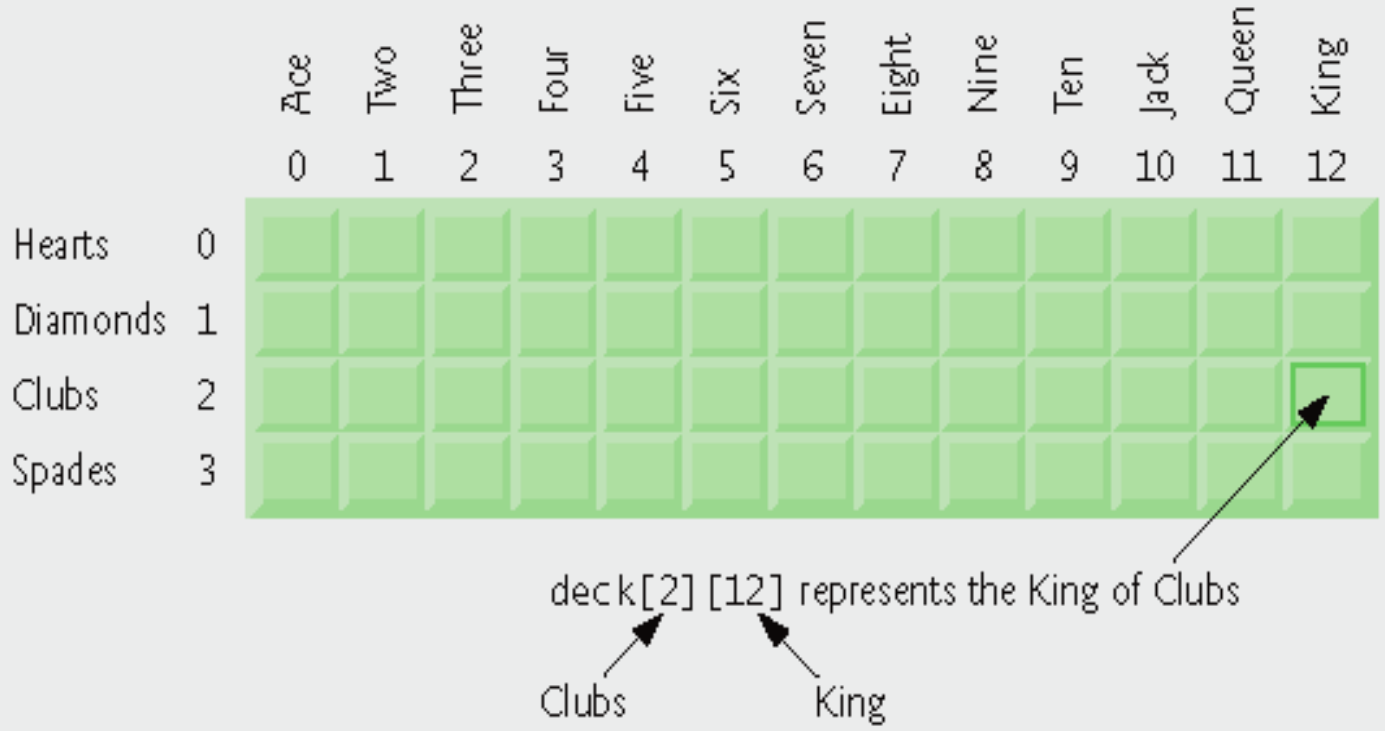
8. Arrays of Pointers



9. Card Shuffling and Dealing Simulation

- 洗牌程序
  - 使用指针数组存放字符串（牌的花色）
  - 用二维数组的第二维索引来表示牌的面额
  - 将 1-52 存放到二维数组中表示发牌的顺序

9. Card Shuffling and Dealing Simulation



9. Card Shuffling and Dealing Simulation

```
// DeckOfCards class definition
class DeckOfCards
{
public:
   DeckOfCards(); // constructor initializes deck
   void shuffle(); // shuffles cards in deck
   void deal(); // deals cards in deck
private:
   int deck[ 4 ][ 13 ]; // represents deck of cards
};
```

```
void DeckOfCards::shuffle()
 {
   int row; // represents suit value of card
   int column; // represents face value of card

   for ( int card = 1; card <= 52; card++ )
   {
      do // choose a new random location until unoccupied slot is found
      {
         row = rand() % 4; // randomly select the row
         column = rand() % 13; // randomly select the column
      } while( deck[ row ][ column ] != 0 ); // end do...while

      deck[ row ][ column ] = card;
   } // end for
} // end function shuffle
```

```
// deal cards in deck
void DeckOfCards::deal()
{
  static const char *suit[ 4 ] =
    { "Hearts", "Diamonds", "Clubs", "Spades" };

  static const char *face[ 13 ] =
    { "Ace", "Deuce", "Three", "Four", "Five", "Six", "Seven",
    "Eight", "Nine", "Ten", "Jack", "Queen", "King" };
```

suit array contains pointers to **char** arrays

face array contains pointers to **char** arrays

```
for ( int card = 1; card <= 52; card++ )
{
   for ( int row = 0; row <= 3; row++ )
   {
      for ( int column = 0; column <= 12; column++ )
      {
         if ( deck[ row ][ column ] == card )
         {
            cout << setw( 5 ) << right << face[ column ]
               << " of " << setw( 8 ) << left << suit[ row ]
               << ( card % 2 == 0 ? '\n' : '\t' );
         } // end if
      } // end innermost for
   } // end inner for
} // end outer for
} // end function deal
```

Cause face to be output right justified in field of 5 characters

Cause suit to be output left justified in field of 8 characters

10. Function Pointers

- 函数指针
  - 包含函数的地址
    - 函数名为函数的起始地址

10. Function Pointers

- **Selection Sort**

8	4	6	9	2	3	1
1	4	6	9	2	3	8
1	2	6	9	4	3	8
1	2	3	9	4	6	8

1	2	3	4	9	6	8
1	2	3	4	6	9	8
1	2	3	4	6	8	9
1	2	3	4	6	8	9

```
// prototypes
void selectionSort( int [], const int, bool (*)( int, int ) );
int main()
 {
   ......
  if ( order == 1 )
  {
     selectionSort( a, arraySize, ascending );
  } // end if
  else
  {
     selectionSort( a, arraySize, descending );
  } // end else part of if...else
  ......
}
```

Pass pointers to functions **ascending** and **descending** as parameters to function **selectionSort**

```
void selectionSort( int work[], const int size, bool (*compare)( int, int ) )
{
  int smallestOrLargest; // index of smallest (or l

  for ( int i =
  {
    smallestOrLargest = i; // first index of remaining vector

    for ( int index = i + 1; index < size;
      if ( !(*compare)( work[ smallestOrLargest ], work[ index ] ) )
        smallestOrLargest = index;

    swap( &work[ smallestOrLargest ], &work[ i ] );
  } // end if
} // end function selectionSort
```

Parentheses necessary to indicate pointer to function

compare is a pointer to a function that receives two integer parameters and returns a **bool** result

Dereference pointer **compare** to execute the function

```
// determine whether element a is less than
// element b for an ascending order sort
bool ascending( int a, int b )
{
   return a < b; // returns true if a is less than b
} // end function ascending

// determine whether element a is greater than
// element b for a descending order sort
bool descending( int a, int b )
{
   return a > b; // returns true if a is greater than b
} // end function descending
```

10. String Manipulation Functions

- <cstring>
  - 操纵字符串数据
  - 比较字符串
  - 字符和字符串查找
  - 字符串分隔

思考题：下面程序的运行结果？为什么？

```
void GetMemory(char *p)
{
     p = (char *)malloc(100);
}

int main()
{
    char *str = NULL;

    GetMemory(str);
    strcpy(str, "hello world");
    cout << str << endl;

    return 0;
}
```

思考题：下面程序的运行结果？为什么？

```
char *GetMemory(void)
{
     char p[] = "hello world";
     return p;
}

int main()
{
    char *str = NULL;

    str = GetMemory();
    cout << str << endl;

    return 0;
}
```

思考题：下面程序的运行结果？为什么？

```
void GetMemory(char **p, int num)
{
     *p = (char *)malloc(num);
}

int main()
{
    char *str = NULL;

    GetMemory(&str, 100);
    strcpy(str, "hello");
    cout << str << endl;

    return 0;
}
```

思考题：下面程序的运行结果？为什么？

```
int main()
{
    char *str = (char *) malloc(100);

    strcpy(str, "hello");
    free(str);

    if(str != NULL)
    {
        strcpy(str, "world");
        cout << str << endl;
    }

    return 0;
}
```

思考题：

- 8.20：修改书中的洗牌和发牌程序，使洗牌和发牌操作采用一个函数（shuffleAndDeal）完成
- 8.36：编写一个程序，输入一行文本，然后利用strtok标记每个单词，按相反顺序输出