基于 Linux 的 C++

第六讲　复合数据类型

提　纲

字　符

数　组

结构体

字　符

字符类型、字符文字与量

定义格式：char ch; const char cch = 'C';

字符文字使用单引号对

实际存储时字符类型量存储字符的对应 ASCII 值

可使用 signed 与 unsigned 修饰字符类型

字符表示的等价性

```
char a = 'A';
char a = 65;
char a = 0101;
char a = 0x41;
```

ASCII 码

	0	1	2	3	4	5	6	7	8	9
0	\0	SOH	STX	ETX	EOT	ENQ	ACK	\a	\b	\t
10	\n	\v	\f	\r	SO	SI	DLE	DC1	DC1	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US	(Space)	!	"	#	$	%	&	'
40	(	)	*	+	,	–	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	DEL		

ASCII 码

控制字符、通信专用字符、可打印字符

回车与换行

Windows：\n\r

Linux：\n

Mac：\r

由来：早期电传打字机每秒打印10个字符，换行需要0.2秒，为保证新传输的字符不丢失，每打印一行需要传递额外两个字符控制打字机换行（打字机走纸一行）和回车（打印头移动到最左端）。移植到计算机上时，传递两个字符有些浪费，因此不同系统的实现策略发生了分歧

字符量的数学运算

编写函数，判断某个字符是否为数字

```
bool IsDigit( char c )
{
  if( c >= '0' && c <= '9' )
    return true;
  else
    return false;
}
```

```
bool IsDigit( char c )
{
  if( c >= 48 && c <= 57 )
    return true;
  else
    return false;
}
```

字符量的数学运算

编写函数，将字符转换为大写字符

```
char ToUpperCase( char c )
{
  if( c >= 'a' && c <= 'z' )
    return c – 'a' + 'A';
  else
    return c;
}
```

标准字符特征库

C 标准字符特征库：ctype.h/cctype

标准字符特征库常用函数

```
bool isalnum( char c );
bool isalpha( char c );
bool isdigit( char c );
bool islower( char c );
bool isspace( char c );
bool isupper( char c );
char tolower( char c );
char toupper( char c );
```

数　组

数组的意义与性质

数组的存储表示

数组元素的访问

数组与函数

多维数组

数组的意义与性质

数组的定义

定义格式：元素类型 数组名称[常数表达式];

示例：int a[8]; /* 定义包含 8 个整数元素的数组 */

特别说明

常数表达式必须是常数和常量，不允许为变量

错误示例：int count = 8; int c[count];

数组元素编号从 0 开始计数，元素访问格式为 a[0]、a[1]、……

不允许对数组进行整体赋值操作，只能使用循环逐一复制元素

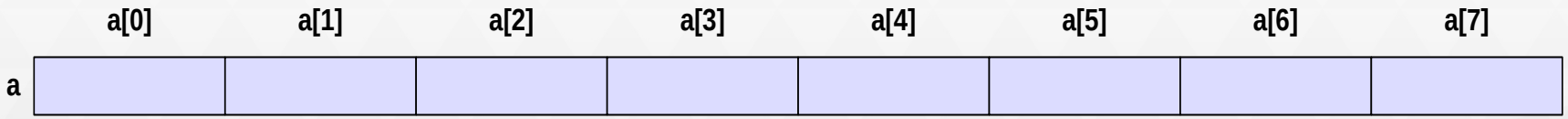
错误示例三：int a[8], b[8]; a = b;

意义与性质

将相同性质的数据元素组织成整体，构成单一维度上的数据序列

数组的存储表示

数组元素依次连续存放，中间没有空闲空间



数组的地址

数组的基地址：数组开始存储的物理位置

数组首元素的基地址：数组首个元素开始存储的物理地址，数值上总是与数组基地址相同

“&”操作符：&a 获得数组的基地址；&a[0] 获得数组首元素的基地址

设数组基地址为 p，并设每个元素的存储空间为 m，则第 i 个元素的基地址为 p + mi

数组元素的初始化

基本初始化格式

定义格式：元素类型 数组名称[元素个数] = { 值1, 值2, 值3, …… };

示例一：int a[8] = { 1, 2, 3, 4, 5, 6, 7, 8 };

初始化时省略元素个数表达式

在全部元素均初始化时可以不列写元素个数，使用 sizeof 操作符可以获得元素个数

示例二：int a[] = { 1, 2, 3, 4, 5, 6, 7, 8 };

```
int num_of_elements = sizeof( a ) / sizeof( a[0] );
```

sizeof(a) 获取数组存储空间大小（以字节为单位），sizeof(a[0]) 获取数组首元素的存储空间大小

数组基本操作示例

编写程序，使用数组存储用户输入的 5 个整数，并计算它们的和

```
#include <iostream>
using namespace std;
int main()
{
  int i, a[5], result = 0;
  for( i = 0; i < 5; i++ )
  {
    cout << "Integer No. " << i + 1 << ": ";
    cin >> a[i];
  }
  for( i = 0; i < 5; i++ )
    result += a[i];
  cout << "The sum of elements of the array is " << result << endl;
}
```

数组与函数

数组元素作为函数实际参数

```
int Add( int x, int y ){  return( x + y ); }

int a[2] = { 1, 2 }, sum;

sum = Add( a[0], a[1] );
```

数组整体作为函数形式参数

数组整体作为函数形式参数

基本格式：返回值类型 函数名称(元素类型 数组名称[]，元素个数类型 元素个数)

示例：void GenerateIntegers(int a[], unsigned int n);

特别说明：作为函数形式参数时，数组名称后的中括号内不需列写元素个数，必须使用单独的参数传递元素个数信息

数组作为函数参数

编写函数，随机生成 n 个位于[lower, upper]区间的整数并保存到数组中

```
void GenerateIntegers( int a[], unsigned int n, int lower, int upper )
{
  unsigned int i;
  Randomize();
  for( i = 0; i < n; i++ )
    a[i] = GenerateRandomNumber( lower, upper );
}
```

数组作为函数参数

直接书写常数：只能操作 8 个元素的数组！

```
void GenerateIntegers( int a[8], int lower, int upper )
{
  unsigned int i;
  Randomize();
  for( i = 0; i < 8; i++ )
    a[i] = GenerateRandomNumber( lower, upper );
}
```

不写常数：不知道元素个数，易出错

```
void GenerateIntegers( int a[], int lower, int upper )
{
  ……
}
```

数组作为函数参数

直接书写变量：错误！

```
/* 元素元素个数不允许为量 */
void GenerateIntegers( int a[n], int lower, int upper )
{
  ……
}
```

调用格式

使用单独数组名称作为函数实际参数，传递数组基地址而不是数组元素值

形式参数与实际参数使用相同存储区，对数组形式参数值的改变会自动反应到实际参数中

```
#define  NUMBER_OF_ELEMENTS  8
const int lower_bound = 10;
count int upper_bound = 99;
int a[NUMBER_OF_ELEMENTS];
GenerateIntegers( a, NUMBER_OF_ELEMENTS, lower_bound, upper_bound);
```

数组与函数：arrmanip.h

编写程序，随机生成8个10～99之间的整数保存到数组中，然后将这些元素颠倒过来

```
void GenerateIntegers( int a[], unsigned int n, int lower, int upper );
void ReverseIntegers( int a[], unsigned int n );
void SwapIntegers( int a[], unsigned int i, unsigned int j );
void PrintIntegers( int a[], unsigned int n );
```

数组与函数：main.cpp

```
#include <iostream>
#include "arrmanip.h"
using namespace std;
#define NUMBER_OF_ELEMENTS 8
const int lower_bound = 10;
const int upper_bound = 99;

int main()
{
  int a[NUMBER_OF_ELEMENTS];
  GenerateIntegers( a, NUMBER_OF_ELEMENTS, lower_bound, upper_bound );
  cout << "Array generated at random as follows: \n";
  PrintIntegers( a, NUMBER_OF_ELEMENTS );
  ReverseIntegers( a, NUMBER_OF_ELEMENTS );
  cout << "After all elements of the array reversed: \n";
  PrintIntegers( a, NUMBER_OF_ELEMENTS );
  return 0;
}
```

■数组与函数：arrmanip.cpp

```
#include <iostream>
#include "random.h"
#include "arrmanip.h"

using namespace std;

void GenerateIntegers( int a[], unsigned int n, int lower, int upper )
{
  unsigned int i;
  Randomize();
  for( i = 0; i < n; i++ )
    a[i] = GenerateRandomNumber( lower, upper );
}
```

■数组与函数：arrmanip.cpp

```
void ReverseIntegers( int a[], unsigned int n )
{
  unsigned int i;
  for( i = 0; i < n / 2; i++ )
    SwapIntegers( a, i, n - i - 1 );
}
void SwapIntegers( int a[], unsigned int i, unsigned int j )
{
  int t;
  t = a[i],  a[i] = a[j],  a[j] = t;
}
void PrintIntegers( int a[], unsigned int n )
{
  unsigned int i;
  for( i = 0; i < n; i++ )
    cout << setw(3) << a[i];
  cout << endl;
}
```

多维数组

多维数组的定义

定义格式：元素类型 数组名称[常数表达式1] [常数表达式2]…;

示例一：int a[2][2]; /* 2×2 个整数元素的二维数组 */

示例二：int b[2][3][4]; /* 2×3×4 个整数元素数组 */

特别说明：同单维数组

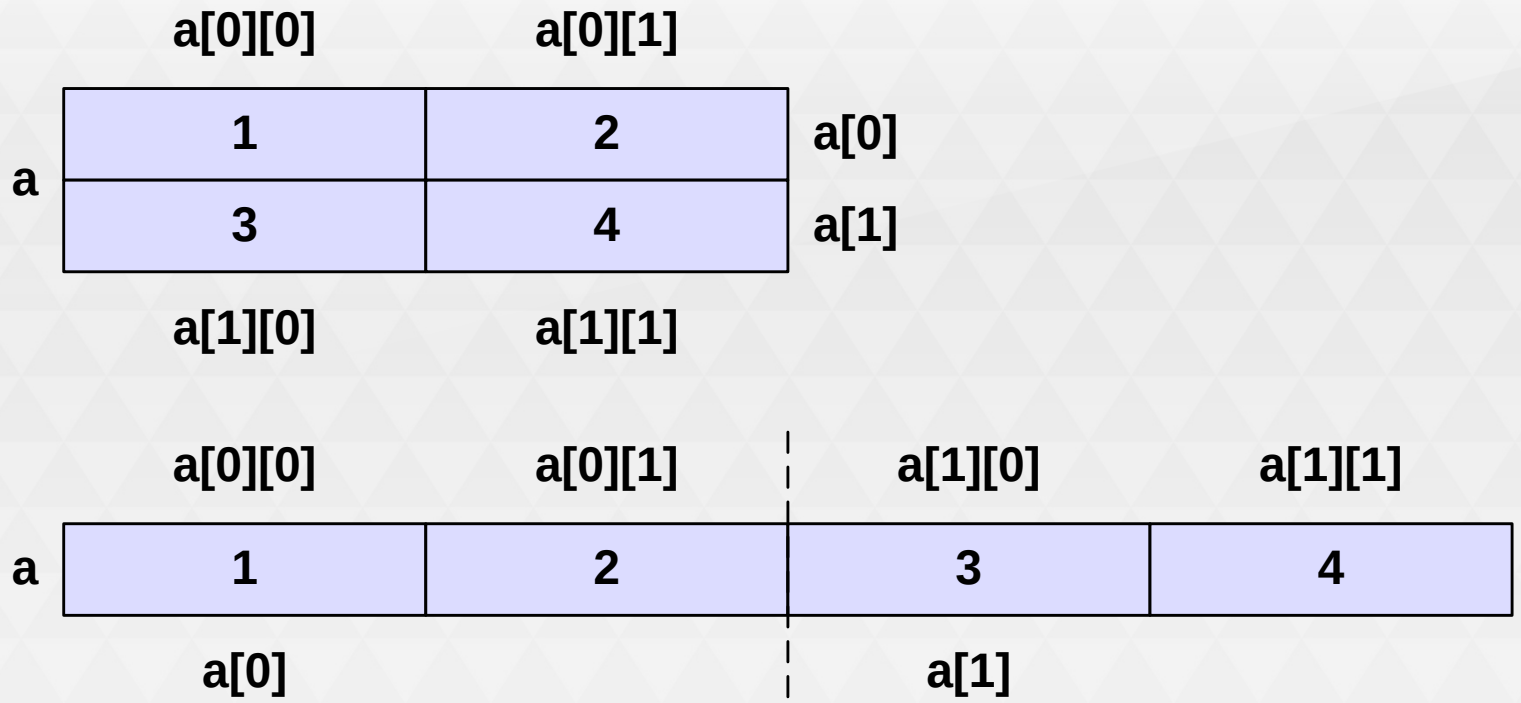
多维数组的初始化

与一维数组类似： int a[2][3] = {1, 2, 3, 4, 5, 6};

单独初始化每一维： int a[2][3] = { {1, 2, 3}, {4, 5, 6} };

多维数组的存储布局

同单维数组，先行后列顺序存放



多维数组的存储布局

测量湖泊水深

在湖面上等距离打上网格，分别测量每个网格的水深，就可以从整体上表示湖泊的情况。图中每个网格中的数字表示水深，值为 0 的表示湖岸，数字 1～5 表示水深（单位为米），每网格对应实际面积为5m×5m。编写程序，计算湖泊的面积和平均水深

测量湖泊水深程序代码

```
#include <iostream>

using namespace std;

#define  NUM_OF_X_GRIDS  9
#define  NUM_OF_Y_GRIDS  4

static const double
lake_region_depths[NUM_OF_Y_GRIDS][NUM_OF_X_GRIDS] =
{
  { 0.0, 0.0, 1.0, 2.0, 2.0, 3.0, 0.0, 0.0, 0.0 },
  { 0.0, 2.0, 3.0, 5.0, 5.0, 3.0, 2.0, 0.0, 0.0 },
  { 0.0, 1.0, 4.0, 3.0, 4.0, 2.0, 2.0, 1.0, 0.0 },
  { 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0 }
};

const double lake_grid_width = 5.0;
```

测量湖泊水深程序代码

```
int main()
{
  double num_of_lake_grids = 0.0, lake_area = 0.0;
  double total_lake_depth = 0.0, mean_lake_depth = 0.0;
  unsigned int i, j;
  for( i = 0; i < NUM_OF_Y_GRIDS; i++ ){
    for( j = 0; j < NUM_OF_X_GRIDS; j++ ){
      if( lake_region_depths[i][j] > 0 ){
        num_of_lake_grids += 1.0;
        total_lake_depth += lake_region_depths[i][j];
      }
    }
  }
  lake_area = lake_grid_width * lake_grid_width * num_of_lake_grids;
  mean_lake_depth = total_lake_depth / num_of_lake_grids;
  cout << "Area of the lake is " << lake_area << "(m2)" << endl;
  cout << "Mean depth of the lake is " << mean_lake_depth << "(m)" << endl;
  return 0;
}
```

结构体

结构体的意义与性质

结构体的存储表示

结构体数据对象的访问

结构体与函数

结构体的意义与性质

结构体的意义

与数组的最大差别：不同类型数据对象构成的集合

当然也可以为相同类型的但具体意义或解释不同的数据对象集合

结构体类型的定义：注意类型定义后面的分号

```
struct  结构体名称
{
  成员类型 1  成员名称 1;
  成员类型 2  成员名称 2;
  ……
  成员类型 n  成员名称 n;
};
```

结构体的意义与性质

结构体类型示例

示例一：日期结构体

```
struct  DATE
{
  int year;
  int month;
  int day;
};
```

示例二：复数结构体

```
 struct COMPLEX
{
  double real;
  double imag;
};
```

结构体类型的声明

仅仅引入结构体类型的名称，而没有给出具体定义，其具体定义在其他头文件中或本文件后续位置

```
 struct COMPLEX;
```

结构体的意义与性质

如何表示学生信息？

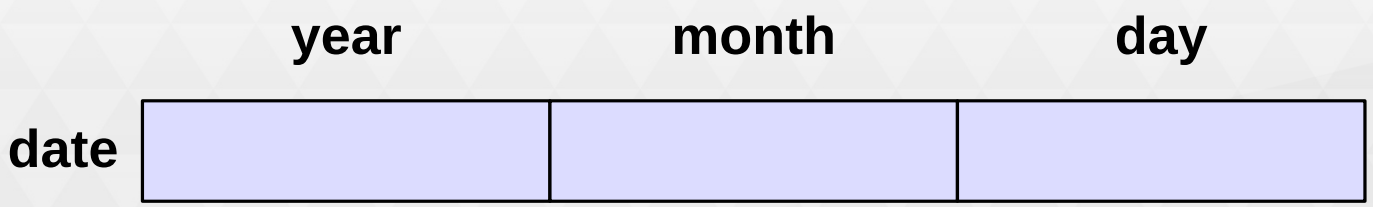
成员：整数类型的学号 id、字符串类型的姓名 name、性别（单独定义枚举类型）gender、年龄 age、字符串类型的地址 addr

```
enum GENDER{ FEMALE, MALE };
struct STUDENT
{
  int id;
  STRING name;     // 假设已有字符串类型的定义
  GENDER gender;
  int age;
  STRING addr;
};
```

结构体的存储表示

按照成员定义顺序存放

各成员的存储空间一般连续



特殊情况

因为不同硬件和编译器的原因，不同类型的成员可能会按照字（两个字节）或双字（四个字节）对齐后存放

使用 sizeof 获得结构体类型量占用空间大小（以字节为单位），下述两种使用方式均可

```
sizeof date;
sizeof( date );
```

结构体数据对象的访问

结构体类型的变量与常量：按普通量格式定义

示例一：DATE date;

示例二：STUDENT zhang_san;

示例三：STUDENT students[8];

结构体量的初始化

示例四：DATE date = { 2008, 8, 8 };

结构体量的赋值

与数组不同，结构体量可直接赋值，拷贝过程为逐成员一一复制

示例五：DATE new_date; new_date = date;

结构体数据对象的访问

结构体量成员的访问

使用点号操作符“.”解析结构体量的某个特定成员

示例一：DATE date; date.year = 2008; date.month = 8; date.day = 8;

嵌套结构体成员的访问

可以连续使用点号逐层解析

示例二：

```
struct FRIEND{ int id; STRING name; DATE birthday; };
FRIEND friend;
friend.birthday.year = 1988;
```

复杂结构体成员的访问

严格按照语法规范进行

示例三：

```
FRIEND friends[4];
friends[0].birthday.year = 1988;
```

结构体与函数

编写一函数，使用结构体类型存储日期，并返回该日在该年的第几天信息，具体天数从 1 开始计数，例如 2016 年 1 月 20 日返回 20，2 月 1 日返回 32

```
unsigned int GetDateCount( DATE date )
{
  static unsigned int days_of_months[13] =
    { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
  unsigned int i, date_id = 0;
  for( i = 1; i < date.month; i++ )
    date_id += days_of_months[i];
  date_id += date.day;
  if( date.month > 2 && IsLeap(date.year) )
    date_id++;
  return date_id;
}
```

结构体与函数

计算机屏幕上的点使用二维坐标描述。编写函数，随机生成一个屏幕上的点。设计算机屏幕分辨率为 1920×1200，屏幕坐标总是从 0 开始计数

```
struct POINT{ int x, y; };
const int orignal_point_x = 0;
const int orignal_point_y = 0;
const int num_of_pixels_x = 1920;
const int num_of_pixels_y = 1200;

POINT GeneratePoint()
{
  POINT t;
  t.x = GenerateRandomNumber( orignal_point_x, num_of_pixels_x - 1 );
  t.y = GenerateRandomNumber( orignal_point_y, num_of_pixels_y - 1 );
  return t;
}
```

编程实践

6.1 编写函数Sort，对包含n个元素的整数数组a，按从小到大顺序排序。此问题有多种算法，不同算法的效率可能不同。

6.2 编写函数Search，对于已排序的包含n个元素的整数数组a，在其中查找整数key。如果该整数在数组中，函数返回true，否则返回false。

6.3 继续编程实践题5.2。将去除大小王的52张扑克牌平均分配给四个玩家，每家13张牌。为描述问题方便，2～9的牌张使用对应字符‘2’～‘9’，字符‘T’表示10，‘J’、‘Q’、‘K’、‘A’表示四类大牌。记每张2～10为0点，J为1点，Q为2点，K为3点，A为4点，统计每家大牌点值。上述牌点计算方法主要用于桥牌游戏。