



C++语言程序设计

第二章 C++简单程序设计



本章主要内容

- C++语言概述
- 基本数据类型和表达式
- 数据的输入与输出
- 算法的基本控制结构
- 自定义数据类型



C++语言的产生

C++语言概述

- C++是从C语言发展演变而来的，首先是一个更好的C
- 引入了类的机制，最初的C++被称为“带类的C”
- 1983年正式取名为C++
- 从1989年开始C++语言标准化工作
- 于1994年制定了ANSI C++标准草案
- 于1998年11月被国际标准化组织（ISO）批准为国际标准，成为目前的C++



C++的特点

- 全面兼容C
 - 它保持了C的简洁、高效和接近汇编语言等特点
 - 对C的类型系统进行了改革和扩充
 - C++也支持面向过程的程序设计，不是一个纯正的面向对象的语言
- 支持面向对象的方法



C++程序实例—例2-1

```
//2_1.cpp
#include <iostream>
using namespace std;
void main()
{
    cout<<"Hello!\n";
    cout<<"Welcome to c++!\n";
}
```

运行结果：
Hello!
Welcome to c++!



C++字符集

- 大小写的英文字母：A~Z, a~z
- 数字字符：0~9
- 特殊字符：

空格	!	#	%	^	&	*
_(下划线)		+	=	-	~	<
>	/	\	,	"	;	.
,	()	[]	{}			



词法记号

- **关键字** C++预定义的单词
- **标识符** 程序员声明的单词，它命名程序正文中的一些实体
- **文字** 在程序中直接使用符号表示的数据
- **操作符** 用于实现各种运算的符号
- **分隔符** () { } , : ;
用于分隔各个词法记号或程序正文
- **空白符** 空格、制表符（TAB键产生的字符）、换行符（Enter键所产生的字符）和注释的总称



标识符的构成规则

- 以大写字母、小写字母或下划线()开始。
- 可以由以大写字母、小写字母、下划线()或数字0~9组成。
- 大写字母和小写字母代表不同的标识符。



数据类型

——常量与变量

基本数据类型和表达式

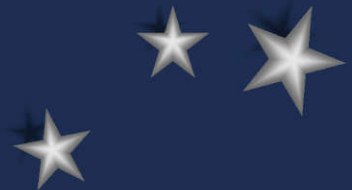
变量先声明后使用

```
#include <iostream>
using namespace std;
void main()
{
    const int PRICE=30;
    int num, total;
    float v , r, h;
    num=10;
    total=num*PRICE;
    cout<<total <<endl;
    r=2.5;
    h=3.2;
    v=3.14159*r*r*h;
    cout<<v<<endl;
}
```

符号常量

常量

变量



数据类型

——整型数据

基本数据类型和表达式

整型变量声明

整型变量

整型常量

```
#include <iostream>
using namespace std;
int main()
{
    const int PRICE=30;
    int num, total;
    float v , r, h;
    num=10;
    total=num*PRICE;
    cout<<total <<endl;
    r=2.5;
    h=3.2;
    v=3.14159*r*r*h;
    cout<<v <<endl;
}
```



数据类型

——整型数据

基本数据类型和表达式

说明：

1. 以数字0开头的为8进制整型常量。
2. 以0x开头的为16进制整型常量。
3. 整型常量可以用后缀字母L(或l)表示长整型，用U(或u)表示无符号型。



数据类型

— 整型数据及取值范围

基本数据类型和表达式

类型	说明符	位数	数值范围
短整	short	16	-32768~32767
基本	int	32	$-2^{31} \sim (2^{31}-1)$
长整	long	32	$-2^{31} \sim (2^{31}-1)$
无符号			
unsigned short	short	16	0~65535
unsigned [int]	[int]	32	$0 \sim (2^{32}-1)$
unsigned long	long	32	$0 \sim (2^{32}-1)$

数据类型

——实型数据

基本数据类型和表达式

- 实型数据有一般形式和指数形式。
- 实型常量默认为double类型。
- 常数后加字母F(或f)表示float类型。

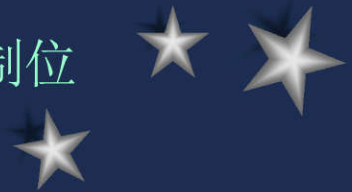


数据类型

——字符型数据 (一)

基本数据类型和表达式

- 字符常量
 - 单引号括起来的一个字符，
如：'a', 'D', '?', '\$'
- 字符变量
 - 用来存放字符常量
例：`char c1, c2;`
`c1='a';`
`c2='A';`
- 字符数据在内存中的存储形式
 - 以ASCII码存储，占1字节，用7个二进制位



数据类型

——字符型数据 (二)

基本数据类型和表达式

- 字符数据的使用方法


- 字符数据和整型数据之间可以运算。
- 字符数据与整型数据可以互相赋值。

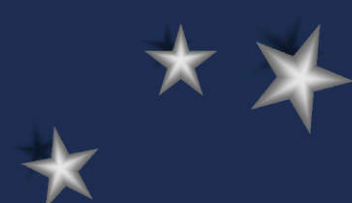
- 字符串常量

例：“CHINA” C H I N A \0

“a” a \0

'a' a

所以：char c;
c="a"; 

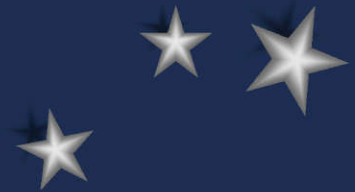


数据类型

——布尔型数据

基本数据类型和表达式

- 布尔型变量的说明：
例： `bool flag;`
- 布尔型数据的取值：
只有 `false` 和 `true` 两个值



数据类型

——变量初始化

基本数据类型和表达式

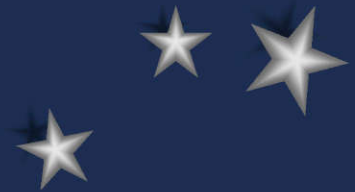
例：

```
int a=3;
```

```
double f=3.56;
```

```
char c='a';
```

```
int c(5);
```



数据类型

—混合运算时的类型转换

基本数据类型和表达式

- 不同类型数据进行混合运算时，C++编译器会自动进行类型转换。
- 为了避免不同的数据类型在运算中出现类型问题，应尽量使用同种类型数据。
- 可以采用强制类型转换：
例如：
`float c;`
`int a, b;`
`c=float(a)/float(b);`
或 `c=(float)a/(float)b;`

变量的存储类型

- auto
 - 属于一时性存储，其存储空间可以被若干变量多次覆盖使用。
- register
 - 存放在通用寄存器中。
- extern
 - 在所有函数和程序段中都可引用。
- static
 - 在内存中是以固定地址存放的，在整个程序运行期间都有效。



算术运算符与算术表达式

- 基本算术运算符

+ - * / (若整数相除, 结果取整)

% (取余, 操作数为整数)

- 优先级与结合性

先乘除, 后加减, 同级自左至右

- ++, -- (自增、自减)

例: `i++;` `--j;`



赋值运算符和赋值表达式

简单的赋值运算符“=”

基本数据类型和表达式

- 举例
 $n=n+5$
- 表达式的类型
等号左边对象的类型
- 表达式的值
等号左边对象被赋值后的值



赋值运算符和赋值表达式

复合的赋值运算符

基本数据类型和表达式

- 有10种复合运算符:

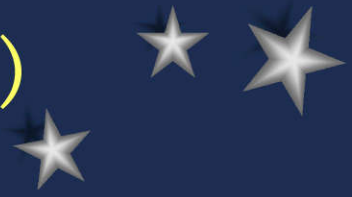
$+=$, $-=$, $*=$, $/=$, $\%=$,

$\ll=$, $\gg=$, $\&=$, $\^=$, $|=$

- 例

$a+=3$ 等价于 $a=a+3$

$x*=y+8$ 等价于 $x=x*(y+8)$



赋值运算符和赋值表达式

——赋值表达式举例

基本数据类型和表达式

$a=5$ 表达式值为5

$a=b=c=5$ 表达式值为5, a, b, c均为5

$a=5+(c=6)$ 表达式值为11, a为11, c为6

$a=(b=4)+(c=6)$

表达式值为10, a为10, b为4, c为6

$a=(b=10)/(c=2)$

表达式值为5, a为5, b为10, c为2

$a+=a-=a*a$ 相当于 $a=a+(a=a-a*a)$



逗号运算和逗号表达式

- 格式
表达式1, 表达式2
- 求解顺序及结果
先求解1, 再求解2, 最终结果为表达式2的值
- 例
a=3*5 , a*4 最终结果为60



关系运算与关系表达式

- 关系运算是一种比较简单的一种逻辑运算，优先次序为：



- 关系表达式是一种最简单的逻辑表达式
其结果类型为 `bool`，值只能为 `true` 或 `false`。
例如： `a > b`， `c <= a + b`， `x + y == 3`



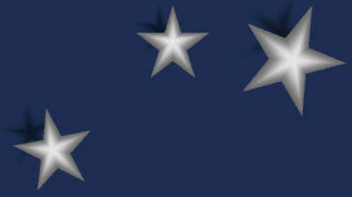
逻辑运算与逻辑表达式

- 逻辑运算符

 !(非) &&(与) ||(或)
优先次序: 高 → 低

- 逻辑表达式

例如: $(a > b) \&\& (x > y)$
其结果类型为 `bool`, 值只能为 `true` 或 `false`



条件运算符与条件表达式

- 一般形式
表达式1? 表达式2: 表达式3
表达式1可以是任意类型，非零表示true。
- 执行顺序
 - 先求解表达式1，
 - 若表达式1的值为true，则求解表达式2，表达式2的值为最终结果
 - 若表达式1的值为false，则求解表达式3，表达式3的值为最终结果
- 例： `x=a>b? a:b;`



条件运算符与条件表达式

- 注意：
 - 条件运算符优先级高于赋值运算符，低于逻辑运算符
 - 表达式2、3的类型可以不同，条件表达式的最终类型为 2 和 3 中较高的类型。

● 例： $x = a > b ? a : b ;$

①
②



sizeof 运算符

- 语法形式
sizeof (类型名)
或 sizeof (表达式)
- 结果值：
“类型名”所指定的类型或“表达式”的结果类型所占的字节数。
- 例：
sizeof(short)
sizeof(x)



运算符优先级

基本数据类型和表达式

括号

++, --, sizeof

*, /, %

+, -

==, !=

位运算

&&

||

?:

赋值运算

逗号运算

高

低



混合运算时数据类型的转换

——隐含转换

基本数据类型和表达式

- 一些二元运算符（算术运算符、关系运算符、逻辑运算符、位运算符和赋值运算符）要求两个操作数的类型一致。
- 在算术运算和关系运算中如果参与运算的操作数类型不一致，编译系统会自动对数据进行转换（即隐含转换），基本原则是将低类型数据转换为高类型数据。

char, short, int, unsigned, long, unsigned long, float, double

低  高



混合运算时数据类型的转换

——隐含转换

基本数据类型和表达式

- 当参与运算的操作数必须是bool型时，如果操作数是其它类型，编译系统会自动将非0数据转换为true，0转换为false。
- 位运算的操作数必须是整数，当二元位运算的操作数是不同类型的整数时，也会自动进行类型转换，
- 赋值运算要求左值与右值的类型相同，若类型不同，编译系统会自动将右值转换为左值的类型。



混合运算时数据类型的转换

——强制类型转换

基本数据类型和表达式

- 语法形式：
类型说明符(表达式)
或
(类型说明符)表达式
- 强制类型转换的作用是将表达式的结果类型转换为类型说明符所指定的类型。



语句

- 声明语句
- 表达式语句
- 选择语句
- 循环语句
- 跳转语句
- 复合语句
- 标号语句



表达式语句

语句

- 格式：
表达式;
- 表达式语句与表达式的区别：
表达式可以包含在其他表达式中，而语句不可。
例如：if ((a=b)>0) t=a;
不可写为：if ((a=b;)>0) t=a;

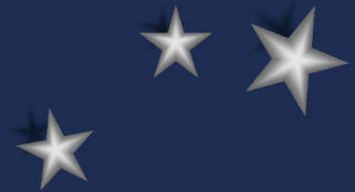


复合语句

语 句

- 将多个语句用一对大括号包围，便构成一个复合语句
- 例如

```
{  
    sum=sum+i;  
    i++;  
}
```



简单的输入、输出

- 向标准输出设备（显示器）输出

例：`int x;`

`cout<<"x="<<x;`

- 从标准输入设备（键盘）输入

例：`int x;`

`cin>>x;`



算法的基本控制结构

- 顺序结构
- 分支结构
- 循环结构



如何解决分支问题？

例2-2

输入一个年份，判断是否闰年。



```
#include <iostream>
using namespace std;
int main()
{ int year;
  bool IsLeapYear;
  cout<<"Enter the year: ";
  cin>>year;
  IsLeapYear = ((year % 4 == 0 &&
    year % 100 != 0) || (year % 400 == 0));
  if (IsLeapYear)
    cout<<year<<" is a leap year"<<endl;
  else
    cout<<year<<" is not a leap year"<<endl;
}
```


运行结果：

```
Enter the year: 2000  
2000 is a leap year
```

if 语句

——三种形式

算 法 的 基 本 控 制 结 构

if (表达式) 语句

例: if (x>y) cout<<x;

if (表达式) 语句1 else 语句2

例: if (x>y) cout<<x;

else cout<<y;

if (表达式1) 语句1

else if (表达式2) 语句2

else if (表达式3) 语句3

...

else 语句 n



如何解决多分支问题？

例2-3

输入两个整数，比较两个数的大小。



```
#include<iostream>
using namespace std;
int main()
{
    int x,y;
    cout<<"Enter x and y:";
    cin>>x>>y;
    if (x!=y)
        if (x>y)
            cout<<"x>y"<<endl;
        else
            cout<<"x<y"<<endl;
    else
        cout<<"x=y"<<endl;
}
```

运行结果1:

Enter x and y:5 8

$x < y$

运行结果2:

Enter x and y:8 8

$x = y$

运行结果3:

Enter x and y:12 8

$x > y$

if 语句

——嵌套

算法的基本控制结构

- 一般形式

```
if( )  
    if( ) 语句 1  
    else 语句 2  
else  
    if( ) 语句 3  
    else 语句 4
```

- 注意

语句 1、2、3、4 可以是复合语句，每层的 if 与 else 配对，或用 { } 来确定层次关系。

特殊的多分支结构

例2-4

输入一个0~6的整数，转换成星期输出。



```
#include <iostream>
using namespace std;
int main()
{ int day;
  cin >> day;
  switch (day)
  { case 0: cout<<"Sunday"<<endl; break;
    case 1: cout<<"Monday"<<endl; break;
    case 2: cout<<"Tuesday"<<endl; break;
    case 3: cout<<"Wednesday"<<endl; break;
    case 4: cout<<"Thursday"<<endl; break;
    case 5: cout<<"Friday"<<endl; break;
    case 6: cout<<"Saturday"<<endl; break;
    default:
      cout<<"Day out of range Sunday .. Saturday"
        <<endl;
      break;
  }
}
```


switch 语句

算法的基本控制结构

● 一般形式

```
switch (表达式) ← 可以是整型、字符型、枚举型
{
  case 常量表达式 1: 语句1
  case 常量表达式 2: 语句2
  ⋮
  case 常量表达式 n: 语句n
  default : 语句n+1
}
```

每个常量表达式的值不能相同，次序不影响执行结果。

可以是多个语句，但不必用{ }。

● 执行顺序

以case中的常量表达式值为入口标号，由此开始顺序执行。因此，每个case分支最后应该加break语句。

使用switch语句应注意的问题

- case分支可包含多个语句，且不用{ }。
- 表达式、判断值都是int型或char型。
- 若干分支执行内容相同可共用一组语句。



如何有效地完成重复工作

例2-5

求自然数1~10之和

分析：本题需要用累加算法，累加过程是一个循环过程，可以用while语句实现。



```
#include<iostream>
using namespace std;
int main()
{
    int i(1), sum(0);
    while(i<=10)
    {
        sum+=i; //相当于sum=sum+i;
        i++;
    }
    cout<<"sum="<<sum
    <<endl;
}
```

运行结果：
sum=55

while 语句

算法的基本控制结构

- 形式

while (表达式) 语句

↑ 可以是复合语句，其中必须含有改变条件表达式值的语句。

- 执行顺序

先判断表达式的值，若为 true 时，执行语句。



例2-6

输入一个整数，将各位数字反转后输出。



```
#include <iostream>
using namespace std;
int main()
{
    int n, right_digit, newnum = 0;
    cout << "Enter the number: ";
    cin >> n;

    cout << "The number in reverse order is ";
    do
    {
        right_digit = n % 10;
        cout << right_digit;
        n /= 10; //相当于n=n/10
    }
    while (n != 0);
    cout<<endl;
}
```

运行结果：

```
Enter the number: 365
```

```
The number in reverse order is 563
```


do-while 语句

- 一般形式

do 语句 ← 可以是复合语句，其中必须含有改变条件表达式值的语句。
while (表达式)

- 执行顺序

先执行循环体语句，后判断条件。
表达式为 true 时，继续执行循环体

- 与while 语句的比较：

-While 语句执行顺序

先判断表达式的值，为true 时，再执行语句

对比下列程序：

算法的基本控制结构

程序1:

```
#include<iostream>
using namespace std;
int main()
{ int i, sum(0);
  cin>>i;
  while(i<=10)
  { sum+=i;
    i++;
  }
  cout<<"sum="<<sum
    <<endl;
}
```

程序2:

```
#include<iostream>
using namespace std;
int main()
{ int i, sum(0);
  cin>>i;
  do{
    sum+=i;
    i++;
  }while(i<=10);
  cout<<"sum="<<sum
    <<endl;
}
```

for 语句

语法形式

for (表达式1; 表达式2; 表达式3) 语句

循环前先求解 | 每次执行完循环体后求解
为true时执行循环体



例2-8

输入一个整数，求出它的所有因子。



```
#include <iostream>
using namespace std;
int main()
{
    int n, k;

    cout << "Enter a positive integer: ";
    cin >> n;
    cout << "Number " << n << " Factors ";

    for (k=1; k <= n; k++)
        if (n % k == 0)
            cout << k << " ";
    cout << endl;
}
```

运行结果1:

```
Enter a positive integer: 36
```

```
Number 36  Factors 1 2 3 4 6 9 12 18 36
```

运行结果2:

```
Enter a positive integer: 7
```

```
Number 7  Factors 1 7
```

例2-9 编写程序输出以下图案

```
          *
         ***
        *****
       *********
      **********
     **********
    *****
   ***
  *
```



```
#include<iostream>
using namespace std;
int main()
{   int i, j, n=4;
    for (i=1; i<=n; i++)    //输出前4行图案
    {   for (j=1; j<=30; j++)
        cout<<'  ' ;    //在图案左侧空30列
        for (j=1; j<=8-2*i ; j++)
            cout<<'  ' ;
        for (j=1; j<=2*i-1 ; j++)
            cout<<' *' ;
        cout<<endl;
    }
}
```



```
for (i=1;i<=n-1;i++) //输出后3行图案
{ for (j=1;j<=30;j++)
    cout<<' '; //在图案左侧空30列
  for (j=1; j<=7-2*i ;j++)
    cout<<' *' ;
  cout<<endl;
}
}
```

循环结构与选择结构相互嵌套

算法的基本控制结构

```
#include<iostream>
using namespace std;
int main()
{
    int n;
    for(n=100; n<=200; n++)
    {
        if (n%3!=0)
            cout<<n;
    }
}
```



例2-10

- 读入一系列整数，统计出正整数个数 i 和负整数个数 j ，读入0则结束。
- 分析：
 - 需要读入一系列整数，但是整数个数不定，要在每次读入之后进行判断，因此使用while循环最为合适。循环控制条件应该是 $n \neq 0$ 。由于要判断数的正负并分别进行统计，所以需要在循环内部嵌入选择结构。



```

#include<iostream>
using namespace std;
int main()
{ int i=0, j=0, n;
  cout<<"请输入若干整数(输入0则结束): ";
  cin>>n;
  while( n!=0 )
  { if(n>0) i++;
    if(n<0) j++;
    cin>>n    ;
  }
  cout<<"正整数个数: "<<i
    <<"    负整数个数: "<<j<<endl;
}

```

break 和 continue 语句

算法的基本控制结构

- break语句

使程序从循环体和switch语句内跳出，继续执行逻辑上的下一条语句。不宜用在别处。

- continue 语句

结束本次循环，接着判断是否执行下一次循环。



typedef语句

自定义数据类型

- 为一个已有的数据类型另外命名
- 语法形式
`typedef 已有类型名 新类型名表;`
- 例如
`typedef double area, volume;
typedef int natural;
natural i1, i2;
area a;
volume v;`



枚举类型—enum

自定义数据类型

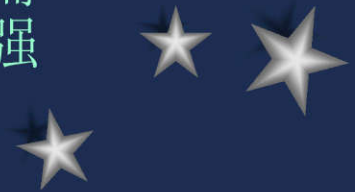
- 只要将需要的变量值一一列举出来，便构成了一个枚举类型。
- 枚举类型的声明形式如下：
`enum 枚举类型名 {变量值列表};`
- 例如：
`enum weekday
{sun, mon, tue, wed, thu, fri, sat};`



枚举类型—enum

自定义数据类型

- 枚举类型应用说明：
 - 对枚举元素按常量处理，不能对它们赋值。例如，不能写：`sun=0`；
 - 枚举元素具有默认值，它们依次为：`0, 1, 2, ……`。
 - 也可以在声明时另行指定枚举元素的值，如：
`enum weekday {sun=7, mon=1, tue, wed, thu, fri, sat} ;`
 - 枚举值可以进行关系运算。
 - 整数值不能直接赋给枚举变量，如需要将整数赋值给枚举变量，应进行强制类型转换。



例2-11

- 设某次体育比赛的结果有四种可能：胜（win）、负（lose）、平局（tie）、比赛取消（cancel），编写程序顺序输出这四种情况。
- 分析：由于比赛结果只有四种可能，所以可以声明一个枚举类型，声明一个枚举类型的变量来存放比赛结果。



```

#include <iostream>
using namespace std;
enum game_result {WIN, LOSE, TIE, CANCEL};
int main()
{ game_result result;
  enum game_result omit = CANCEL;
  int count;
  for (count = WIN ; count <= CANCEL ; count++)
  { result = (game_result)count;
    if (result == omit)
    { cout << "The game was cancelled\n"; }
    else
    { cout << "The game was played ";
      if (result == WIN)   cout << "and we won!";
      if (result == LOSE) cout << "and we lost.";
      cout << "\n";
    }
  }
  return 0;
}

```

运行结果

The game was played and we won!

The game was played and we lost.

The game was played

The game was cancelled

结构体——结构的声明

自定义数据类型

- 结构的概念

结构是由不同数据类型的数据组成的集合体。

- 声明结构类型

```
struct 结构名
{
    数据类型 成员名 1;
    数据类型 成员名 2;
    :
    数据类型 成员名 n;
};
```



结构体——结构的声明

自定义数据类型

- 举例：

```
struct student //学生信息结构体
{
    int num; //学号
    char name[20]; //姓名
    char gender; //性别
    int age; //年龄
    float score; //成绩
    char addr[30]; //住址
}
```



结构体——结构变量说明

自定义数据类型

- 变量说明形式

结构名 结构变量名;

- 注意:

- 结构变量的存储类型概念、它的寿命、可见性及使用范围与普通变量完全一致。
- 结构变量说明在结构类型声明之后，二者也可同时进行。
- 结构变量占内存大小可用 `sizeof` 运算求出：
`sizeof(运算量)`



结构体

——结构变量的初始化和使用

自定义数据类型

- 初始化

说明结构变量的同时可以直接设置初值。

- 使用

结构体成员的引用形式：

结构变量名. 成员名



例2-12

自定义数据类型

结构体变量的初始化和使用

```
#include <iostream>
#include <iomanip>
using namespace std;
struct student //学生信息结构体
{   int num;//学号
    char name[20];//姓名
    char gender;//性别
    int age;//年龄
}stu={97001,"Lin Lin",'F',19};
int main()
{   cout<<setw(7)<<stu.num<<setw(20)<<stu.name
    <<setw(3)<<stu.sex<<setw(3)<<stu.age;
}
```

运行结果:

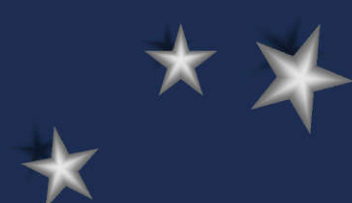
97001 Lin Lin F 19

联合体

自定义数据类型

- 声明形式：

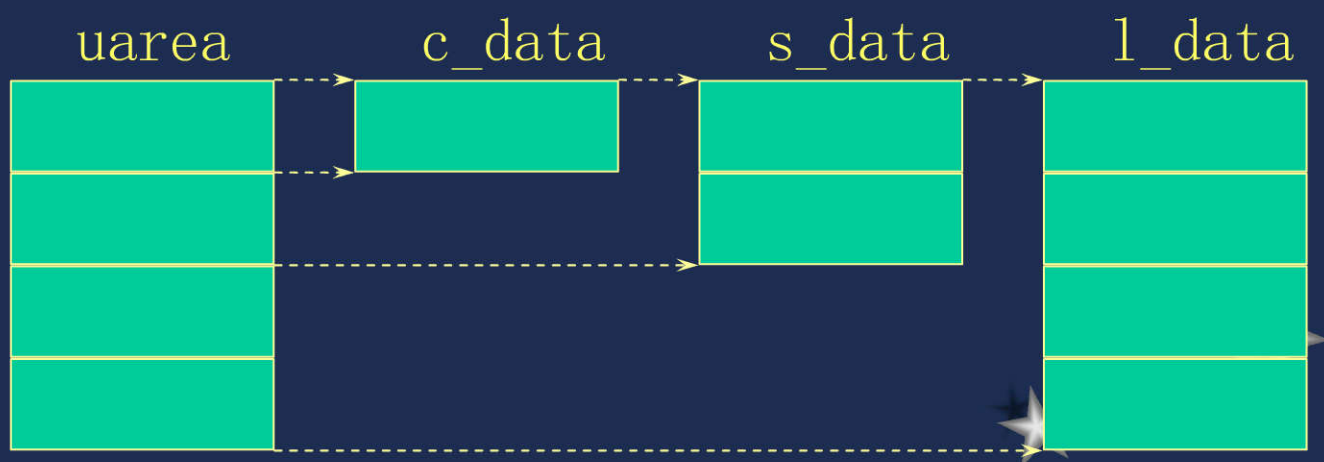
```
union 联合名
{
    数据类型 成员名 1;
    数据类型 成员名 2;
    :
    数据类型 成员名 n;
};
```
- 联合体类型变量说明的语法形式
联合名 联合变量名;
- 引用形式：
联合名. 成员名



联合体

自定义数据类型

```
例: union uarea  
{ char    c_data;  
  short  s_data;  
  long   l_data;  
}
```



无名联合

自定义数据类型

- 无名联合没有标记名，只是声明一个成员项的集合，这些成员项具有相同的内存地址，可以由成员项的名字直接访问。
- 例：

```
union
{
    int    i;
    float  f;
}
```

在程序中可以这样使用：

```
i=10;
f=2.2;
```



小结与复习建议

- 主要内容
 - C++语言概述、基本数据类型和表达式、数据的输入与输出、算法的基本控制结构、自定义数据类型
- 达到的目标
 - 掌握C++语言的基本概念和基本语句，能够编写简单的程序段。
- 实验任务
 - 实验二
- 作业
 - 2-13 2-14 2-17 2-28 2-36 2-37

