



中南大學  
CENTRAL SOUTH UNIVERSITY



# 面向对象程序设计

## Object Oriented Programing

### 第二章 程序设计基础：编程语言


张宝一

地理信息系

zhangbaoyi@csu.edu.cn



# 第二章：程序设计基础

- 编程语言的历史 
- C到C++
- C++到Java/C#面向对象编程语言
- 本章小结

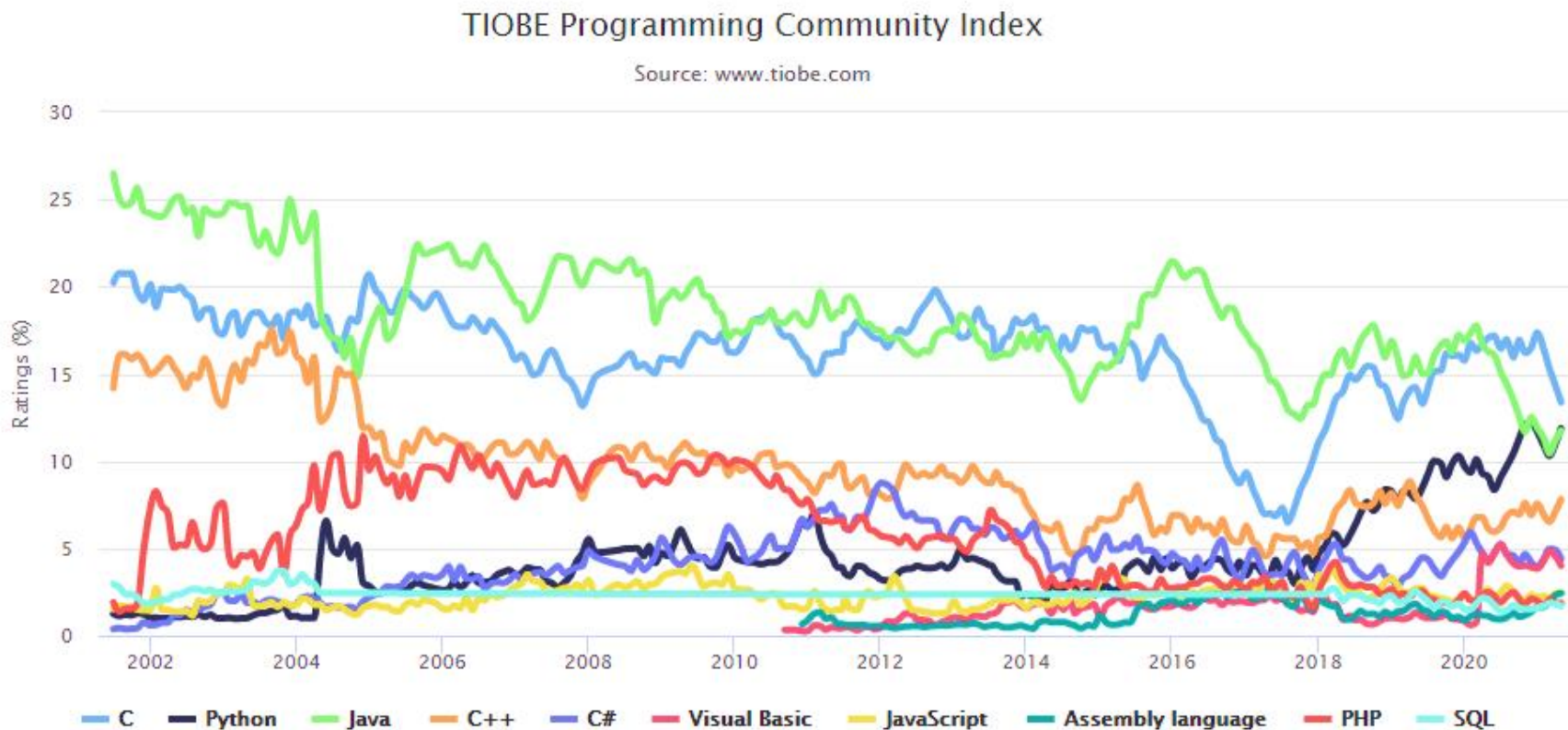
# 编程语言的历史

## ■ 热度Top20的编程语言 (<https://www.tiobe.com/tiobe-index>)

May 2021	May 2020	Programming Language	Ratings	Change
1	1	C	13.38%	-3.68%
2	3	Python	11.87%	+2.75%
3	2	Java	11.74%	-4.54%
4	4	C++	7.81%	+1.69%
5	5	C#	4.41%	+0.12%
6	6	Visual Basic	4.02%	-0.16%
7	7	JavaScript	2.45%	-0.23%
8	14	Assembly language	2.43%	+1.31%
9	8	PHP	1.86%	-0.63%
10	9	SQL	1.71%	-0.38%
11	15	Ruby	1.50%	+0.48%
12	17	Classic Visual Basic	1.41%	+0.53%
13	10	R	1.38%	-0.46%
14	38	Groovy	1.25%	+0.96%
15	13	MATLAB	1.23%	+0.06%
16	12	Go	1.22%	-0.05%
17	23	Delphi/Object Pascal	1.21%	+0.60%
18	11	Swift	1.14%	-0.65%
19	18	Perl	1.04%	+0.16%
20	34	Fortran	0.83%	+0.51%

# 编程语言的历史

- ## ■ 热度Top20的编程语言 (<https://www.tiobe.com/tiobe-index>)



# 编程语言的历史



# 编程语言的历史

## 1、最初使用**机器语言**编程程序

计算机只可以解释用二进制数编写的机器语言。采用二进制码表示指令集合，计算机对机器语言不进行任何检测，只是飞快地执行。

示例：使用机器语言编写的十六进制数表示的程序

A10010

8B160210

01D0

A10410

# 编程语言的历史

## 2、**编程语言**的第一步是汇编语言

为了改善机器语言低效的编程，汇编语言应用而生。汇编语言将无含义的机器语言（二进制码）用人类容易理解的符号表示出来。

示例：使用汇编语言编写的程序

```
MOV AX, X
```

```
MOV DX, Y
```

```
ADD AX, DX
```

```
MOV Z, AX
```

# 编程语言的历史

## 3、高级语言的发明使程序更加接近人类

高级语言采用更接近人类语言的编码、符号来表示特定的计算机执行指令。如：FORTRAN语言

示例：使用FORTRAN语言编写的程序

```
----- PROGRAM LOMOMENT -----
*****
*
*   JIANBO TANG, 2012.02.20
*
*   READ IN THE AT-SITE L-MOMENTS.
*   DATA FILE MAY CONTAIN ANY NUMBER OF REGIONAL
*   A 'REGIONAL DATA STRUCTURE' CONSISTS OF:
*   1. ONE RECORD CONTAINING:
*       (COLUMNS 1- 4) NUMBER OF SITES IN REGION
*       (COLUMNS 5-56) IDENTIFYING LABEL FOR THE
*   2. FOR EACH SITE, ONE RECORD CONTAINING:
*       (COLUMNS 1-12) AN IDENTIFYING LABEL FOR
*       (COLUMNS 13-16) THE RECORD LENGTH AT THE
*       (COLUMNS 17-56) SAMPLE L-MOMENTS L-1, L-C
*
*****
*   COMPUTE L-MOMENT PARAMETERS OF THE DATA[PRECT TXT]
*
-----

IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DOUBLE PRECISION X(N),XMOM(NMOM),SUM(20)
DATA ZERO/0D0/,ONE/1D0/
IF(NMOM.GT.20.OR.NMOM.GT.N) GOTO 1000
DO 10 J=1,NMOM
10 SUM(J)=ZERO
IF(A.EQ.ZERO.AND.B.EQ.ZERO) GOTO 50
IF(A.LE.-ONE.OR.A.GE.B) GOTO 1010

PLOTTING-POSITION ESTIMATES OF PWM'S

DO 30 I=1,N
PPOS=(I+A)/(N+B)
TERM=X(I)
SUM(1)=SUM(1)+TERM
DO 20 J=2,NMOM
```



# 编程语言的历史

## 4、提高子程序的独立性，强化可维护性

为了强化程序的可维护性，还有另外一种方法，就是提高子程序的独立性。

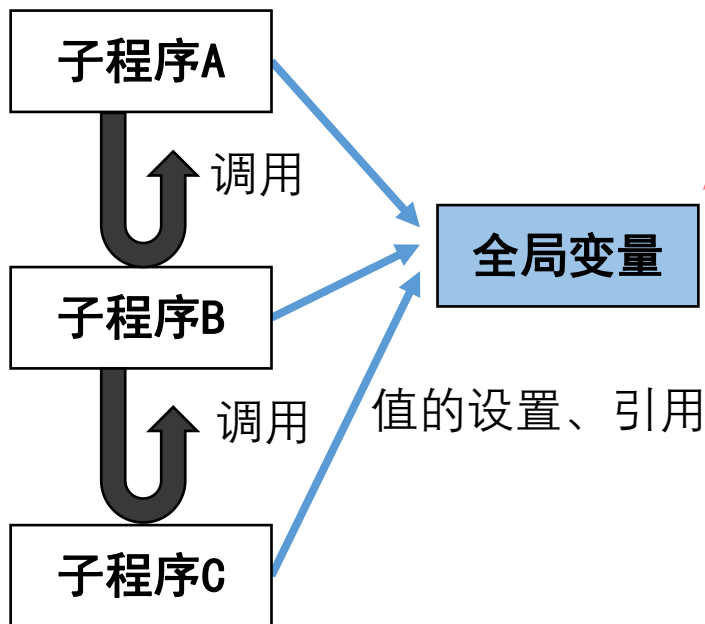
**□子程序**：该结构主要用于将在程序中的多个位置出现的相同命令汇总在一起，以减少程序的大小，实现代码部分复用，提高编程效率。

**□全局变量**：提高子程序独立性的方法：减少调用端（主程序）和子程序之间的共享信息。在多个子程序间共享的变量称为“**全局变量**”。由于从整个程序的所有位置都可以对全局变量进行访问，如果在调试时发现变量有错，就必须检查所有代码，为此，**减少全局变量对于提高程序整体的可维护性而言非常重要。**

# 编程语言的历史

## 4、提高子程序的独立性，强化可维护性

代码示例：



```
int count;
```

```
void subfunctionA(){  
    count = subfunctionB();  
}
```

```
int subfunctionB(){  
    if(count != 1){  
        count = subfunctionC();  
    }  
    return count;  
}
```

```
int subfunctionC(){  
    int value; std::cin >> value;  
    return value;  
}
```

# 编程语言的历史

## 4、提高子程序的独立性，强化可维护性

对于千百个子程序的应用程序而言，修改全局变量就是一个很严峻问题。为了避免出现这样的问题，人们设计了两种结构：

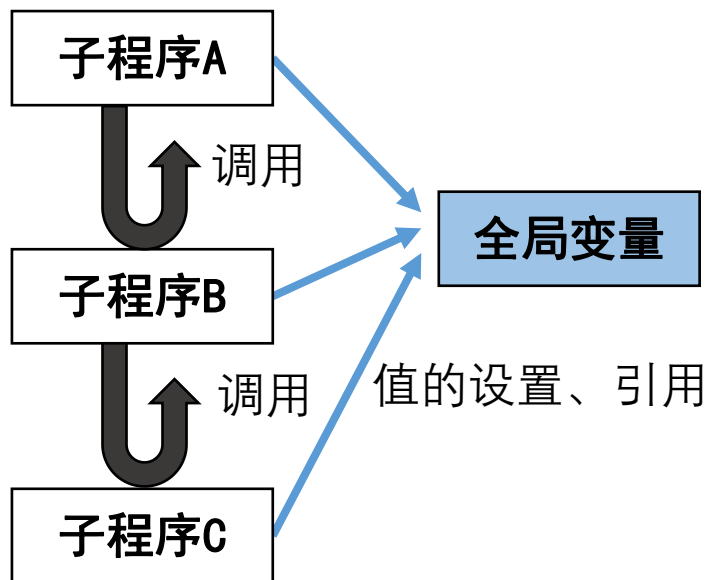
**□局部变量：**局部变量是只可以在子程序中使用的变量，在进入子程序时被创建，退出子程序时消失。

**□按值传递：**通过参数向子程序传递消息时，不直接使用调用端引用的变量，而是复制值以进行传递，这就是按值传递的结构。在这种结构情况下，即使修改了被调用的子程序接收的参数值，也不会影响调用端的变量。

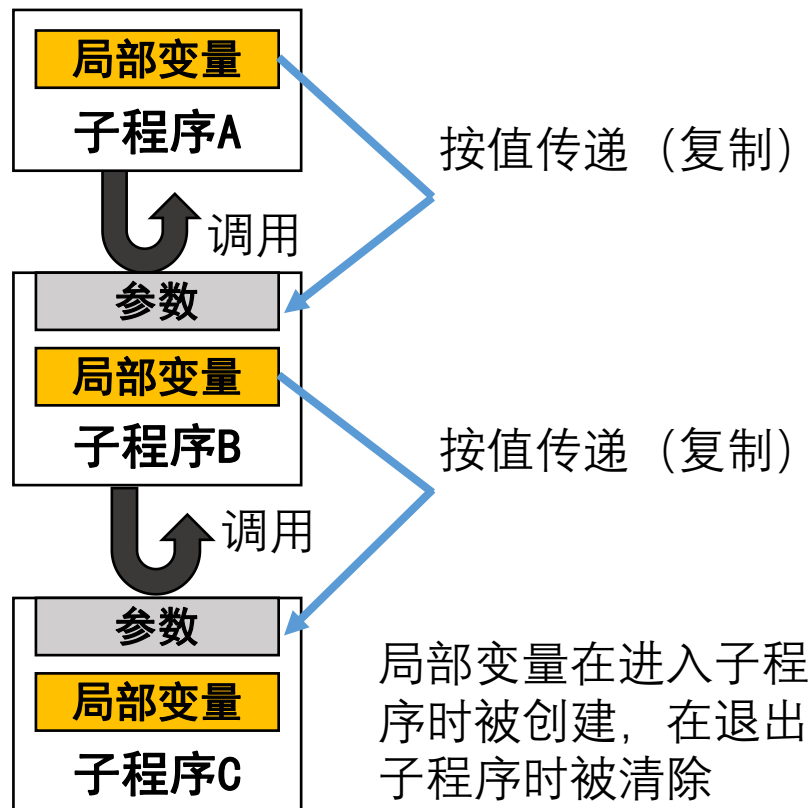
# 编程语言的历史

## 4、提高子程序的独立性，强化可维护性

### □ 全局变量



### □ 局部变量

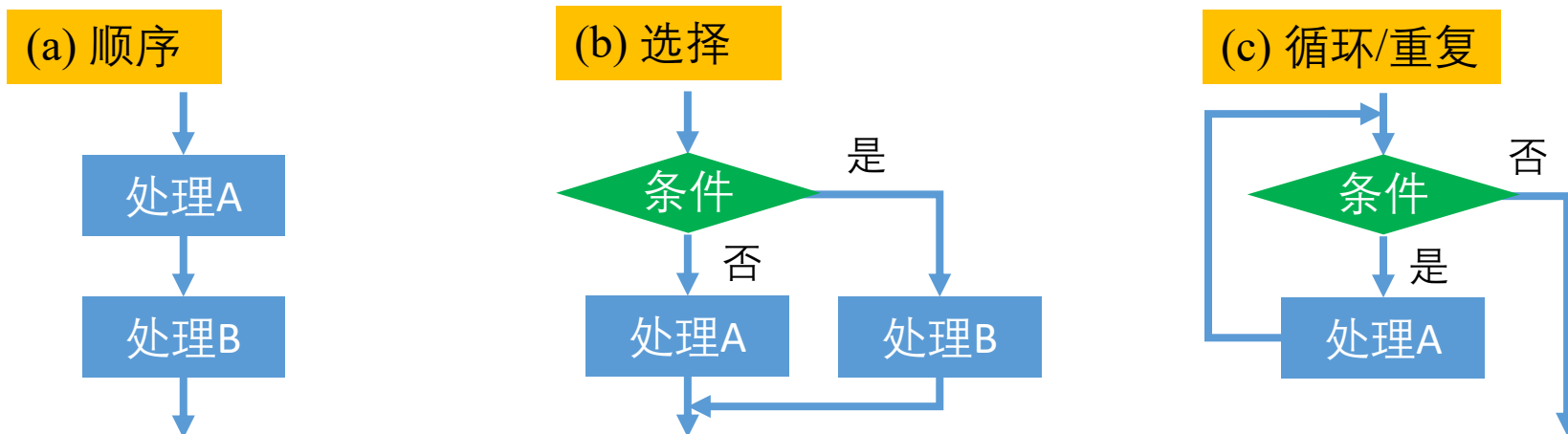


# 编程语言的历史

## 5、重视易懂性的**结构化编程**，实现无GOTO编程

随着高级语言的出现，编程的效率和质量都得到了很多提升。于是在20世纪60年代，国际会议提出了“**软件危机**”——20世纪末，即使全人类都成为程序员，也无法满足日益增大的软件需求。

为了应对软件危机，人们提出了各种新的思想和编程语言。其中最受欢迎的是**结构化编程**，如：最有名的**C语言**。



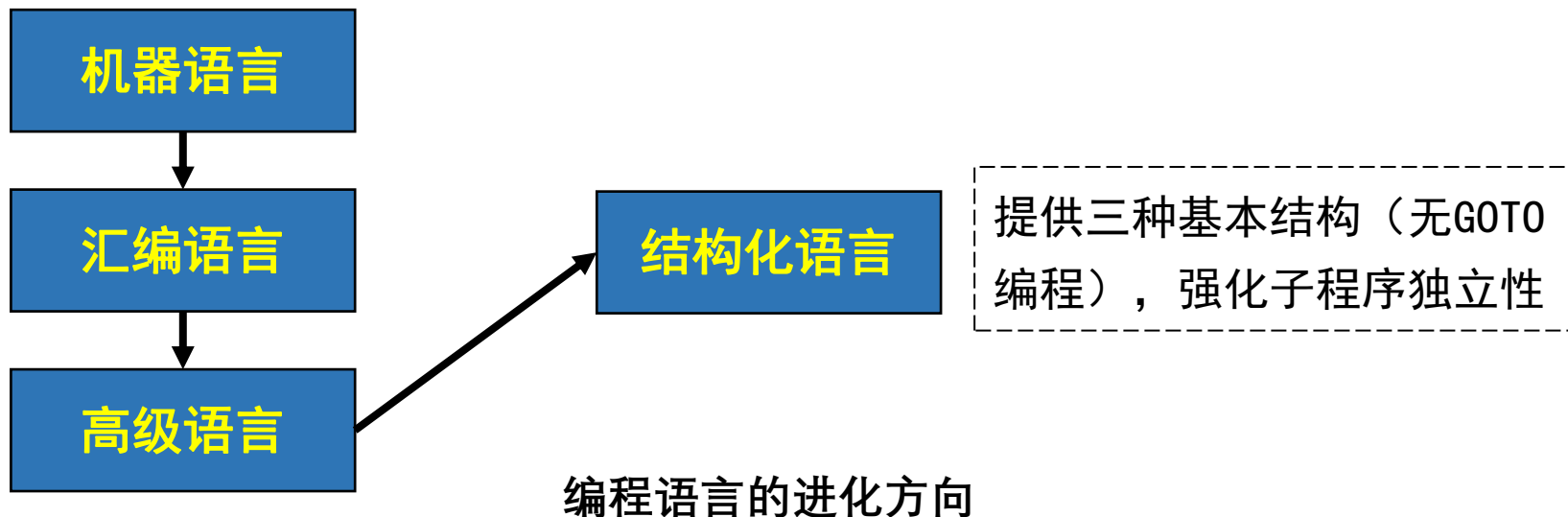
三种基本结构的流程

# 编程语言的历史

## ➤ 进化方向演变为重视**可维护性**和**可重用性**

从机器语言到汇编语言，再到高级语言的进化过程中，人们希望提供编程语言的表现能力，即用更贴近人类的方法简单地表示出希望计算机执行的作用。

□到目前，贴近人类的形式编写程序的目的已经基本实现，但仍无法拯救软件危机。



# 编程语言的历史

## ➤ 没有解决全局变量问题和可重用性差的问题

结构化编程成了程序设计的主流，直到最近才被面向对象夺了风头。

结构化编程有两个无法解决的问题：**全局变量问题**和**可重用性差**。

### 能够解决的问题

避免了滥用GOTO语句造

成的各种代码问题

通过公用子程序，实现了


代码的部分可重用

而能够打破该限制的正是OOP

全局变量

可重用性差

# 第二章：理解OOP-编程语言的历史

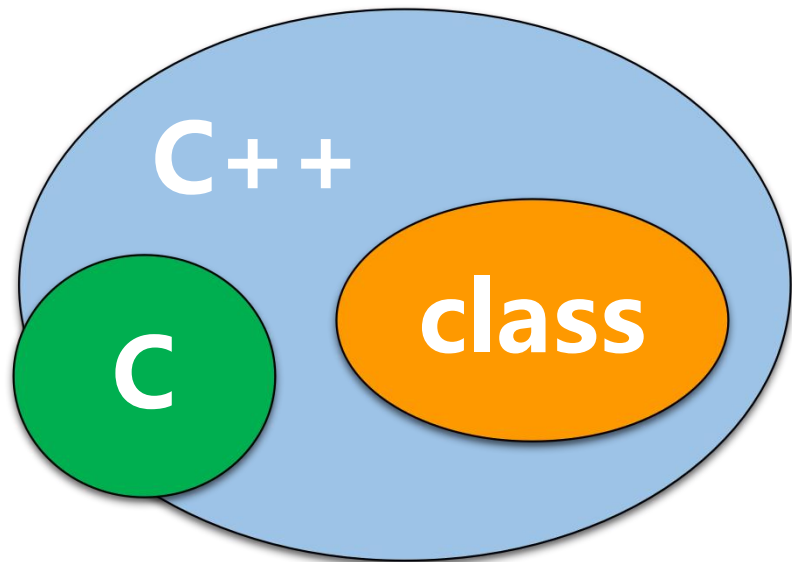
- 编程语言的历史
- C到C++ 
- C++到Java面向对象编程
- 本章小结



# 从C到C++编程基础

## □ C++与C的联系

- 1、C++是在C的基础上发展而来，是带类的C语言。
- 2、C语言支持结构化程序设计，C++语言支持面向对象程序设计。



# C++ 编程基础

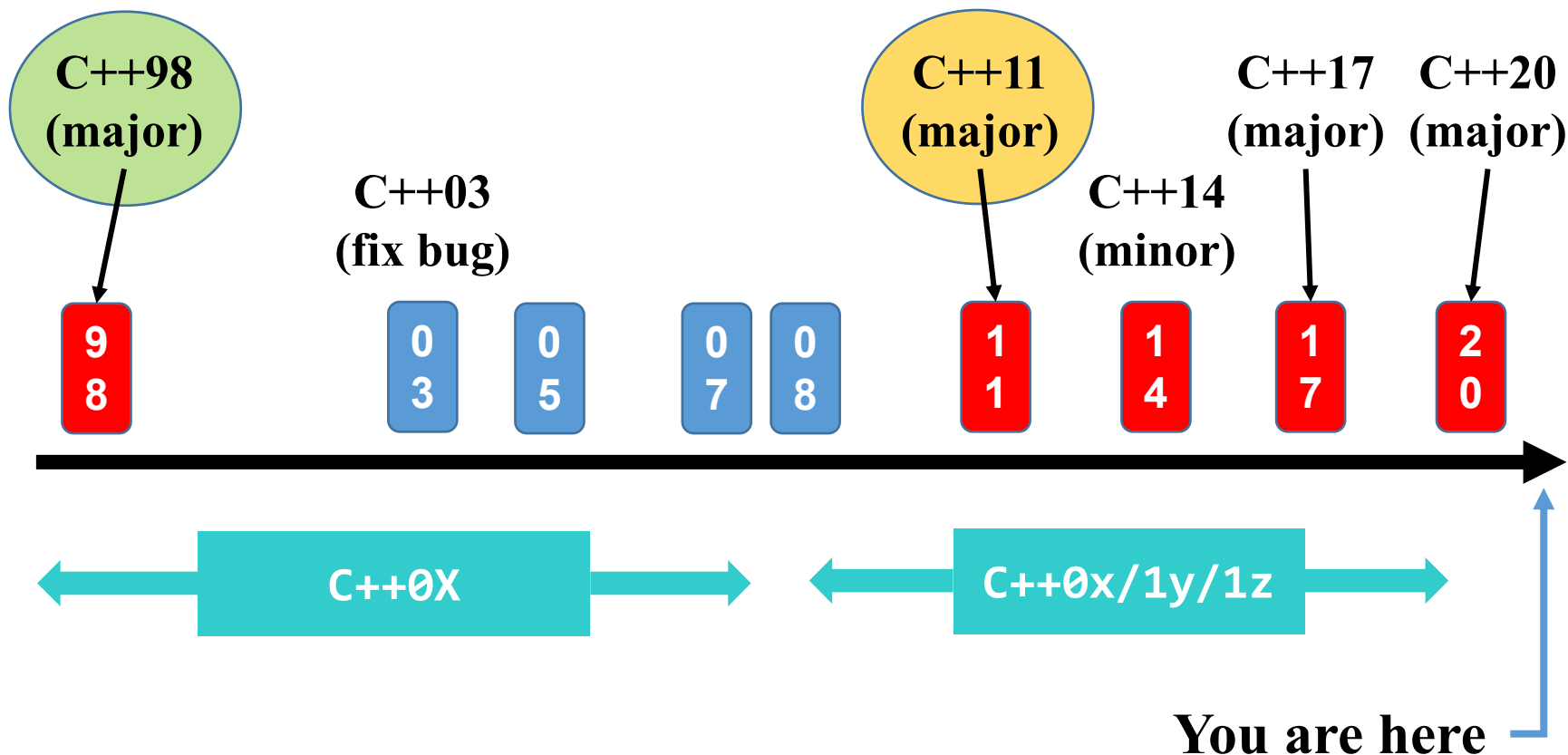
## 一、C++中几个重要概念

- **程序**：数据 (Data structure) + 操作 (Functions)
- **数据类型**：不同数据组织方式得到不同的数据类型  
如：整数 (int/short/long)、浮点数 (float/double)
- **表达式**：操作符 + 操作数，如：2+3, a\*b, m>n
- **语句控制**：数据操作的流程，如：int a=2; a=(2+3)\*b;
- **函数**：int main(); double sum(double x, double y);

# C++ 编程基础

## 一、C++中几个重要概念

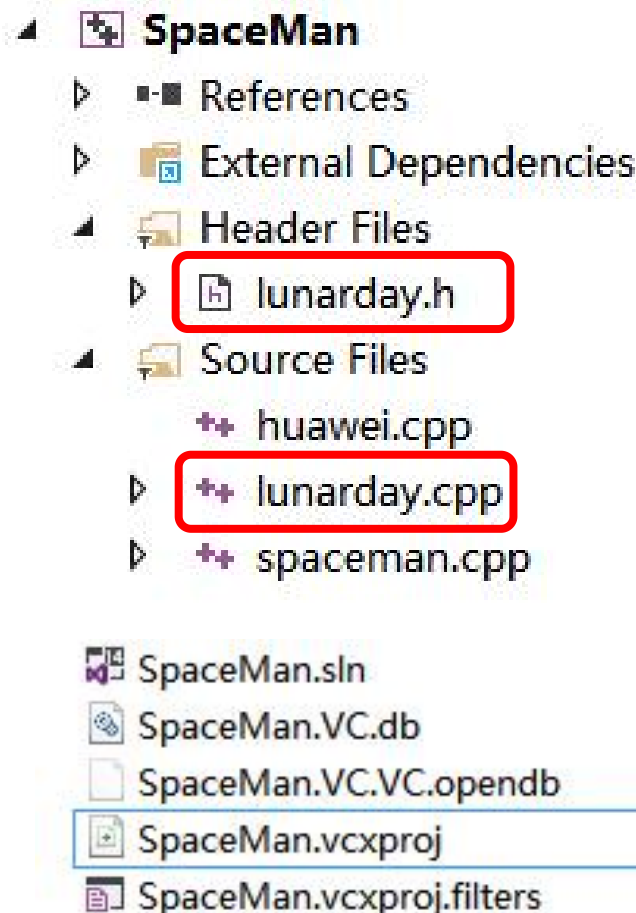
### □ C++语言的标准



# C++ 编程基础

## 二、C++程序的文件组织结构

- 头文件：\*.h、\*.hpp，是对函数、自定义数据类型等进行声明（或定义）的文件；
- 源文件：\*.cpp、\*.c，是对函数进行具体实现，编译生成目标文件（机器码）的文件；
- C++程序项目文件：如 Microsoft Visual Studio IDE 中有 \*.sln、\*.vcxproj、\*.sdf...



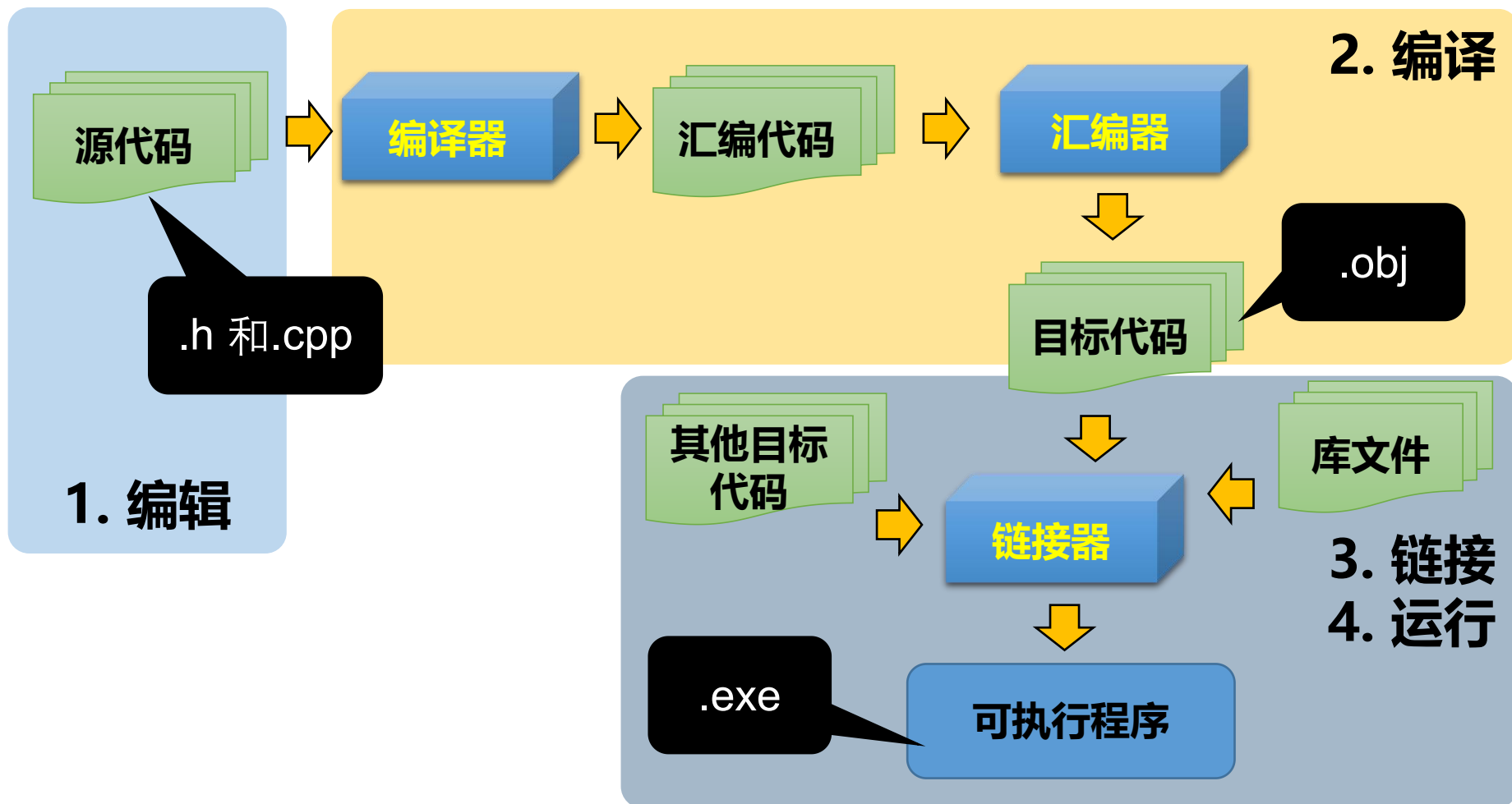
# C++ 编程基础

## 三、开发一个C++程序的四个步骤：

- 1、**编辑**（产生源文件，扩展名为.h/.cpp）
- 2、**编译**（产生目标文件，扩展名为.obj）
- 3、**连接**（产生执行文件，扩展名为.exe）
- 4、**运行**

# C++ 编程基础

## 三、开发一个C++程序的四个步骤：



# C++ 编程基础

## 四、C++集成开发环境



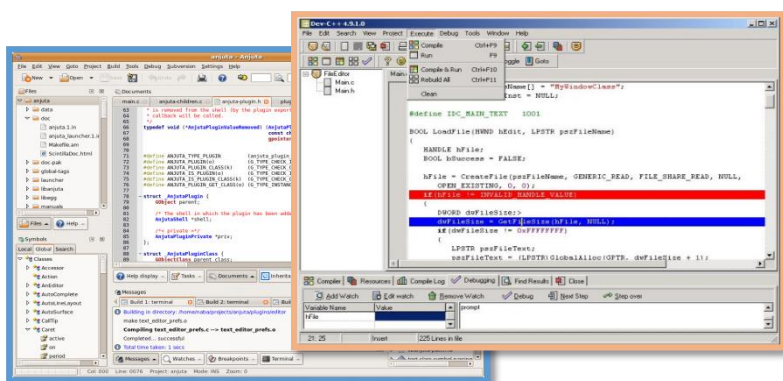
Microsoft Visual Studio



Eclipse CDT



QT



[Dev C++](#)

[kDevelop](#)



Anjuta Devstudio

[Ultimate++](#)

# C++ 编程基础

## 五、C++程序设计基础

### □ 编程风格

良好的编程风格，不仅有利于自己对程序的调试，而且会大大增加程序的可复用机会。

```
#include "myfunc.h"
```

```
int max(int x, int y)
{
    // 比较两个整数的大小
    if (x > y) {
        return x;
    }
    else {
        return y;
    }
}
```



```
#include "我是中文头文件.h"
```

```
int 比较daxiao(int x, int sss)
{
    if (x > sss) {
        return x;
    }
    else
    {
        return sss;}
}
```





# C++ 编程基础

## 五、C++程序设计基础

### □ 编程风格 1、注释

- 定义：为增加程序的可读性而在程序中附加的说明性文字。
- 形式：
  - ☆ 以符号//打头，只占一行。  
C++特有的注释形式。
  - ☆ 包含在符号/\*与\*/之间，可占多行。  
继承C的注释形式。

# C++ 编程基础

## 五、C++程序设计基础

### □ 编程风格

#### 2、命名（为常量、变量、函数取名）

- (1) 名字必须符合标识符的规范。
- (2) 标识符：由字母、数字、下划线组成，而且只能以字母、下划线打头。
- (3) 名字不能是保留字（系统有固定用途的标识符）。
- (4) 字母的大小写有区别。
- (5) 名字最好能表达一定的含义。

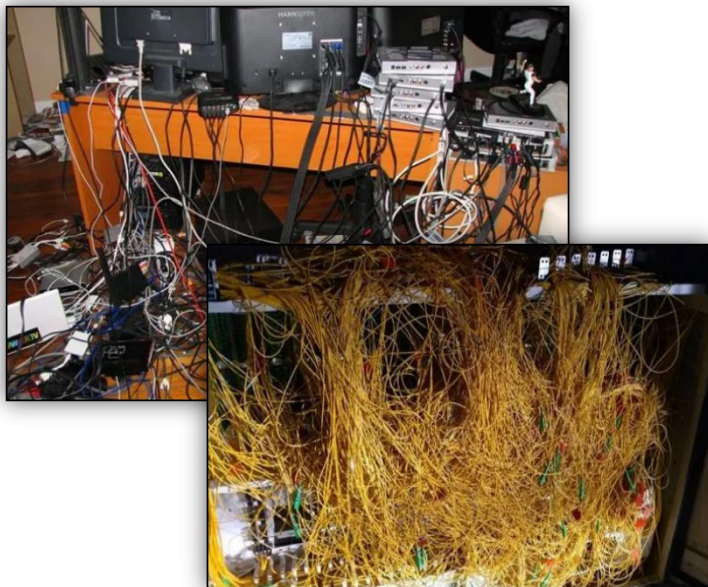
# C++ 编程基础

## 五、C++程序设计基础

### □ 编程风格

#### 3、编排

编排时使用缩进、空行、空格使程序更清晰。



# C++ 编程基础

## 五、C++程序设计基础

### □ 简单性原则

- 1、可以用一句话说清楚的，不要用一页纸去说明，可以用一个简单的语句完成的功能，不要用许多语句来完成。
- 2、不要写太长的函数，可以用函数调用来缩短函数的定义。
- 3、不要写太长的语句，可以用多条语句来代替一条语句。

# C++ 编程基础

## 五、C++程序设计基础

### □ 简单性原则

- 4、如果文件太长，将它分成几个小文件。
- 5、不要用太多的嵌套，可以考用switch 语句或者引入新的函数来解决问题。
- 6、定义类时，一个文件放一个类的定义。

## 五、C++程序设计基础

### □ 一致性原则

- 1、变量的命名应该有意义。
- 2、在程序中加入适当的注释。
- 3、利用缩进使程序清晰。
- 4、相关的内容组织在一起。
- 5、能简单，则简单 —— 简单即是美。

# C++ 编程基础

## 五、C++程序设计基础

### ➤ 学好一门编程语言我们需要做什么？

构成语句的“语法” — 编程风格与规则：  
如：if/switch/for/while的语句结构；  
类/函数的定义、结构与使用语法规则

语法

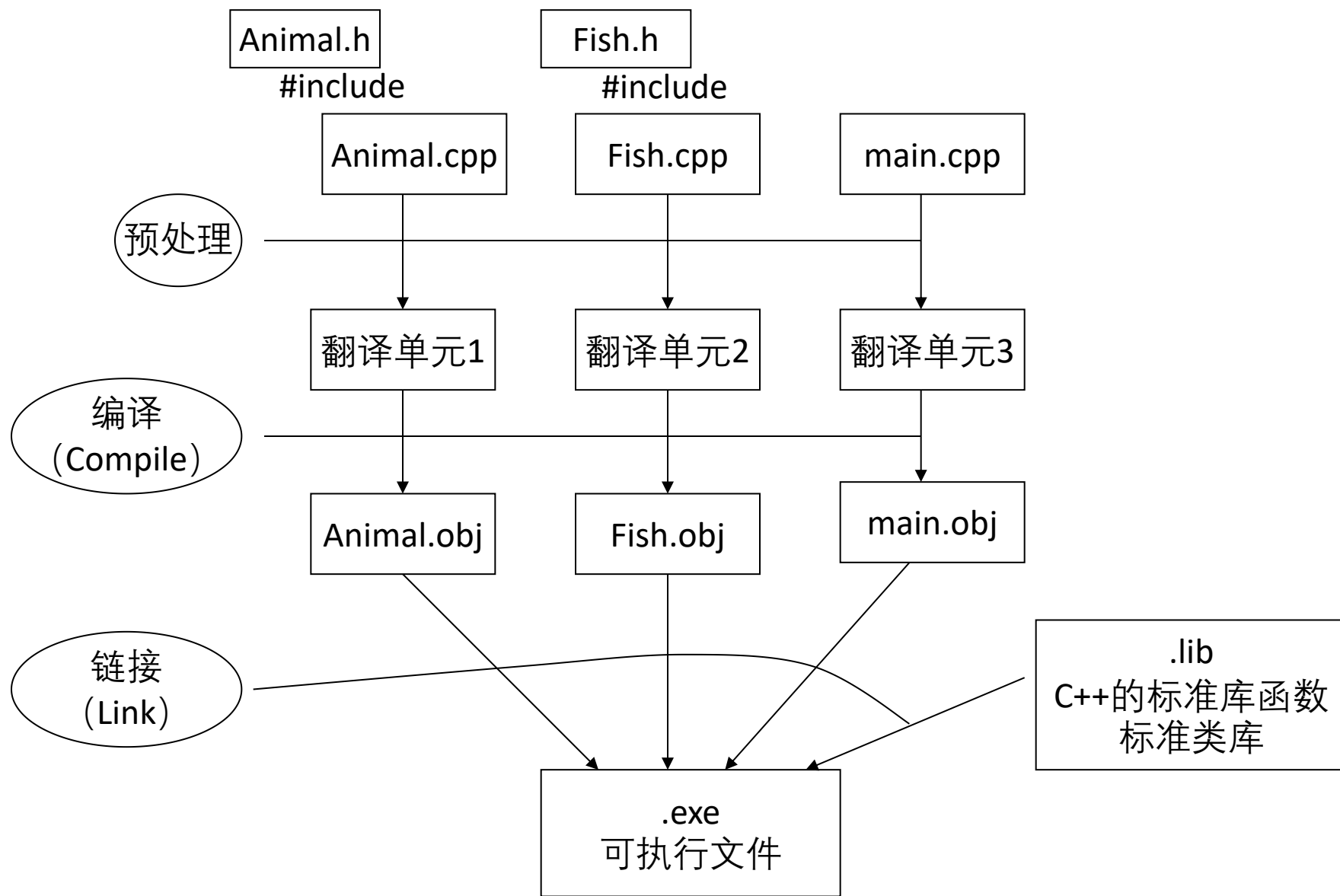
掌握语言的“单词” — 数据类型：  
如何定义，如：int, struct, class

单词

明确学习的动机：  
这种语言有哪些优点、有哪些缺点？

动机

# C++ 编程基础





# C++ 编程基础

## 五、C++程序设计基础

### □ 头文件 (\*.h)

- 问题：`#include <filename.h>` 和 `#include "filename.h"` 有什么区别？
- 答：对于 `#include <filename.h>`，编译器在标准库路径搜索 `filename.h`
- 对于 `#include "filename.h"`，编译器从用户的工作路径开始搜索 `filename.h`

# C++ 编程基础

## 五、C++程序设计基础

### □ 头文件 (\*.h)

#### • 文件包含(inclusion of files)

- 如果程序中宏定义语句很多，可以将它们包含于一个文件中，例如“macros.h”中。在程序中只需一条语句即可。即：
- `#include <macros.h>` 或 `#include "macros.h"`
- 其中头文件名称macros.h使用尖括弧（angle brackets）`<>` 或双引号`"`包括起来。
- 此macros.h文件中可为：
  - `// macros.h`
  - `#define MAX 32`
  - `#define sq(n) (n)*(n)`
  - ... ..
- macros.h称为头文件（header files）（或称包含文件，include files）。
- C++语言系统中有很多头文件，它们除包含宏替换定义语句外，更多的是函数原型说明和类定义，及其所用各种数据类型和常量的定义。

# C++ 编程基础

## 五、C++程序设计基础

### □ 编译预处理命令

- 预处理功能由一些预处理命令组成。常用的预处理命令有以下三项：宏定义命令、文件包含命令和条件编译命令。预处理命令具有以下特点：
  - 1. 在正常编译操作之前执行。
  - 2. 在左边加一 # 号，作为标志。
  - 3. 预处理命令不是编程语句，因此不加分号。
  - 4. 一般独占一行。

# C++ 编程基础

## 五、C++程序设计基础

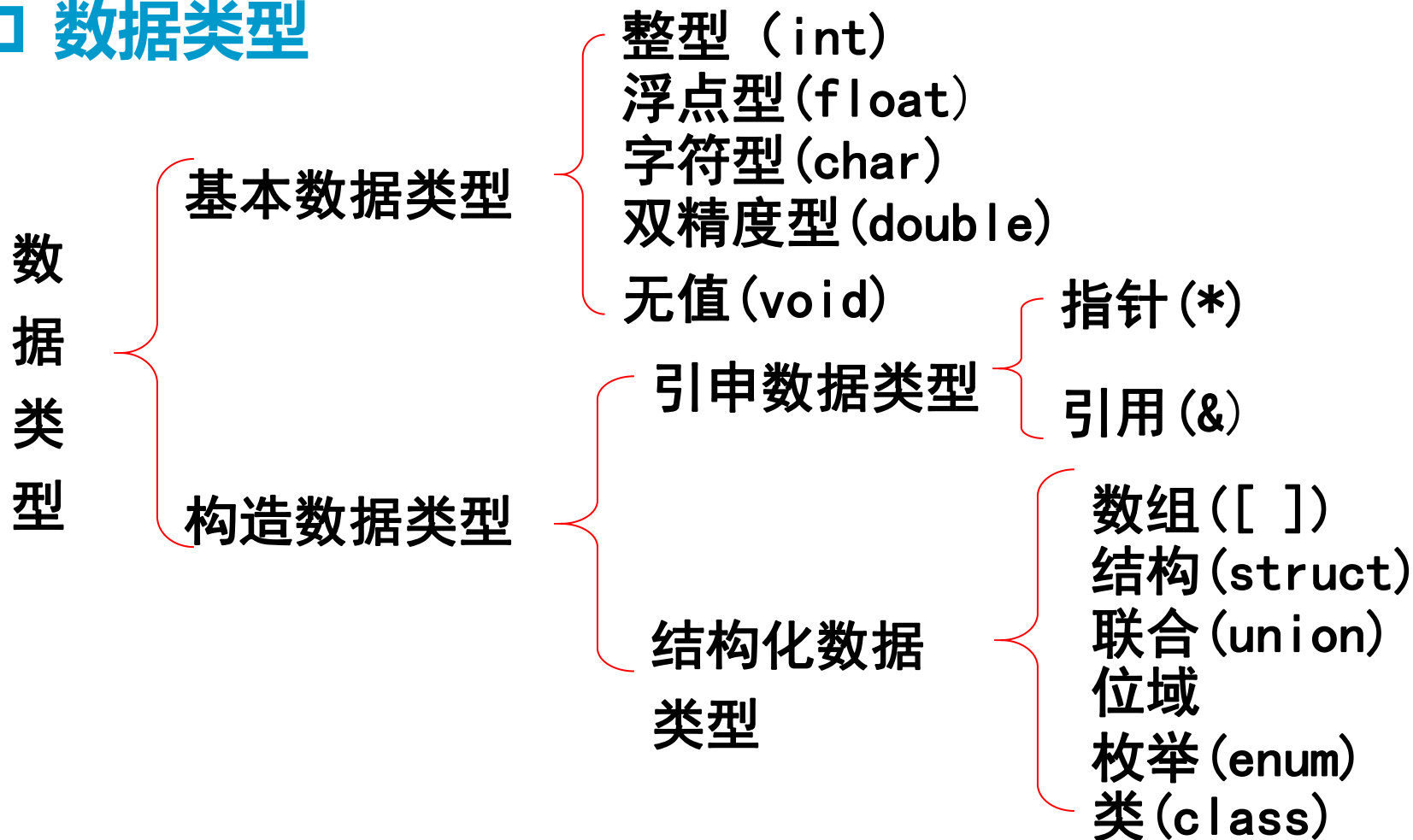
### □ 编译预处理命令

- `#include` 包含指令
  - 将一个源文件嵌入到当前源文件中该点处。
  - `#include<文件名>`
    - 按标准方式搜索，文件位于C++\vc++系统目录的include子目录下
  - `#include"文件名"`
    - 首先在当前目录中搜索，若没有，再按标准方式搜索。
- `#define` 宏定义指令（参见有关C语言教材）
  - 定义符号常量，已被`const`定义语句取代。
  - 定义带参数宏，已被内联函数取代。
- `#undef`
  - 删除由`#define`定义的宏，使之不再起作用。

# C++ 编程基础

## 五、C++程序设计基础

### □ 数据类型



## 五、C++程序设计基础

### □ 分号的使用

- 问题：C++语句中使用分号“;” 的场合和不使用分号的场合有哪些？

## 五、C++程序设计基础

### □ 分号的使用

使用 “; ” 的场合：

- A. 结构、联合和类的定义：
  - `struct str {...};`
  - `union uni {...};`
  - `class base {...};`
  - `class derive:public base {...};`
- B. 每条程序语句：
  - `int i;`
  - `cout<<i;`
  - `fun(i);`
- C. 函数原型说明：
  - `void fun(int, char *);`
  - `double& fun(double&, char *);`

## 五、C++程序设计基础

## □ 分号的使用

## 不使用“;” 的情况:

- **A. 预处理语句:**
  - `#include <iostream.h>`
  - `#define MAX 100`
- **B. 函数体（函数定义）：**
  - `void fun(int i, char *ptr)                   {...}`
  - `double& fun(double& d, char *ptr)`
  - `{           ...}`
  - `return d;       }`
- **C. 程序控制语句:**
  - `if (p) {           cout<<p;           }`
  - `for ( int i=0; i<10; i++)               { cout<<i<<endl; }`



# C++ 编程基础

## 五、C++程序设计基础

### □ 指针

- 指针是一种用于存放某个变量的地址值的变量。该指针被认为是指向该变量。一个指针的类型决定于它所指向的变量的类型。指针既可指向基本数据类型（即预定义类型），又可指向数组、函数、指针和文件等。
- 指针可以初始化为0、NULL（即0，这是标准库头文件中定义的符号化常量）或一个地址。内容为0或NULL的指针不指向任何变量。

• 例如：

- `int a;`
- `int *ptr;`
- `ptr = &a;`



- 其中指针ptr被初始化为指向变量a。它们也可表示为：

- `a = *ptr;`



# C++ 编程基础

## 五、C++程序设计基础

### □ 指针

请问运行Test函数会有什么样的结果？

- 问题：
- `void GetMemory(char *p)`
- `{`
- `p = (char *)malloc(100);`
- `}`
- `void Test(void)`
- `{`
- `char *str = NULL;`
- `GetMemory(str);`
- `strcpy(str, "hello world");`
- `printf(str);`
- `}`



- 答：程序崩溃。
- 因为GetMemory并不能传递动态内存，
- Test函数中的 str一直都是 NULL。
- `strcpy(str, "hello world");`将使程序崩溃。

# C++ 编程基础

## 五、C++程序设计基础

### □ 指针与数组

- 数组（array）是由一组相同类型的数据组成的数据结构。它具有一组连续的内存地址。
- 数组名是一个常量指针（其值不能改变），也是该数组中首元素的地址值。
- 一维数组的指针如下：

- `int a[5] = { 1, 3, 5, 7, 9 };`
- `int *ptr;`
- `ptr = a;`



# C++ 编程基础

## 五、C++程序设计基础

### □ 指针与数组

- 数组指针与数组名两者的异同：
  - 1. 整型变量指针ptr与整型数组指针ptrA的声明格式相同，其间空格可放可不放。
  - 2. arr既是数组名，又是数组地址，还是数组指针（称为常量指针），三位一体。
- 因此arr可在一定范围内与ptrA等效地使用。[]可在一定范围内与\*等效地使用。例如：
  - arr[0]即\*arr即\*ptrA即ptrA[0] = 1
  - arr[3]即\*(arr+3)即\*(ptrA+3)即ptrA[3] = 7
- 但arr的使用不如ptrA灵活，如：
- 不允许\*arr++，而允许\*ptrA++。

# C++ 编程基础

## 五、C++程序设计基础

### □ 指针数组

- [例1]使用一维一级字符指针数组来存放一星期中的各天。

- `// arr_ptr1.cpp`
- `// Show the function of a character array`
- `#include <iostream.h>`
- `char *name[ ] = { " ", "Monday", "Tuesday",`
- `"Wednesday", "Thursday", "Friday",`
- `"Saturday", "Sunday"};`
- `void main()`
- `{`
- `int week;`
- `while (1)`
- `{`
- `cout<<"Input sequential number:";`
- `cin>>week;`
- `if ( week<1 || week>7 ) break;`
- `cout<<"This is "<<name[week]<<endl;`
- `}`
- `}`

# C++ 编程基础

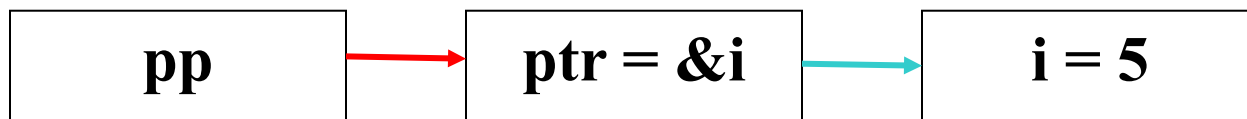
## 五、C++程序设计基础

### □ 指向指针的指针

- 例如：

- `int i=5;`
- `int *ptr = &i;`
- `int **pp = &ptr;`

- 即：指向指针ptr的指针    整型变量i的指针    整型变量



- `i = *ptr;`
- 或
- `i = **pp;`

# C++ 编程基础

## 五、C++程序设计基础

### □ 指向指针的指针

- 问题:
- `void GetMemory(char *p)`
- `{`
- `p = (char *)malloc(100);`
- `}`
- `void Test(void)`
- `{`
- `char *str = NULL;`
- `GetMemory(str);`
- `strcpy(str, "hello world");`
- `printf(str);`
- `}`



请问这个代码要如何改？

## 五、C++程序设计基础

### □ 动态内存申请

**new与delete**

**内存空间申请**

(1) **new操作符**：表示从堆内存中申请一块空间。

(2) **返回值**：

**申请成功**：返回所申请到的空间的首地址。

**申请失败**：返回空指针（NULL/nullptr）。



## 五、C++程序设计基础

### □ 动态内存申请

(3) new的三种格式：

☆ new 数据类型

☆ new 数据类型 (初始化值)

☆ new 数据类型[常量表达式]

例： `int * ip1=new int;`

`int * ip2=new int(3);`

`int * str=new int[10];`

## 五、C++程序设计基础

### □ 动态内存申请

#### 内存空间释放

(1) delete操作符：表示将从堆内存中申请的一块空间返还给堆。

(2) delete的两种格式：

☆ delete 指针名

☆ delete [ ]指针名

# C++ 编程基础

## 五、C++程序设计基础

### □ 动态内存申请

(3) 几点说明:

☆ new与delete需要配套使用:

new 数据类型

new 数据类型 (初始化值)

new 数据类型[常量表达式]

} delete 指针名

↔ delete [ ]指针名

# C++ 编程基础

## 五、C++程序设计基础

### □ 动态内存申请

☆ 在用delete来释放一个指针所指的空间时，必须保证这个指针所指的空间是用new申请的，并且只能释放这个空间一次。

例： `int i;     int * ip=&i;     delete ip;(×)`

`float * fp=new float(3.4);`

`delete fp;(√)`

`delete fp; (×)`

## 五、C++程序设计基础

### □ 动态内存申请

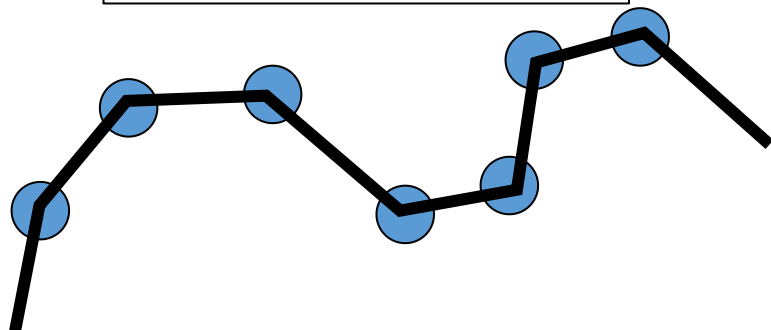
- ☆ 如果在程序中使用new申请了空间，就应该在结束程序前释放所有申请的空间。
- ☆ 当一个指针没有指向合法的空间，或者指针所指的内存已经释放以后，最好将指针的值设为nullptr。

# C++ 编程基础

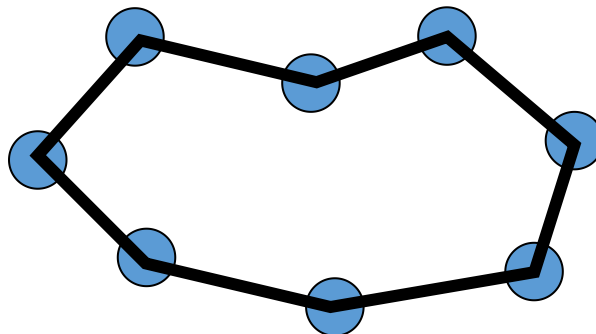
## 五、C++程序设计基础

### □ 链表

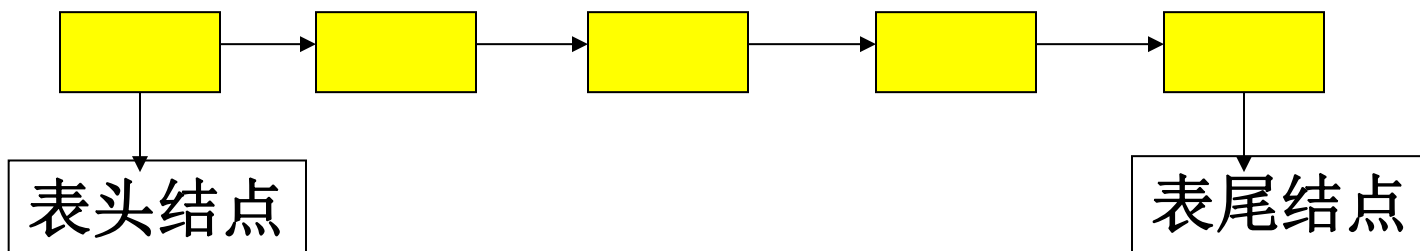
多段线Polyline



多边形Polygon



(1) 链表是由若干个结点链接而成的



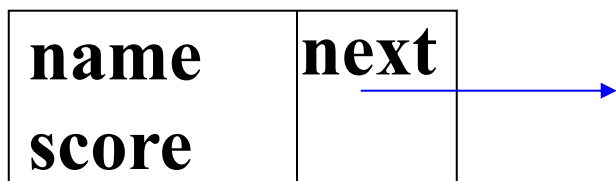
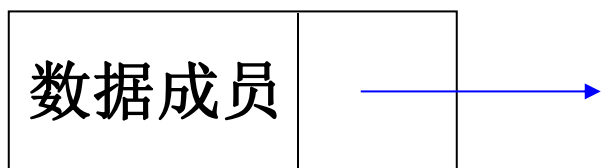
# C++ 编程基础

## 五、C++程序设计基础

### □ 链表

#### (2) 结点用结构体来描述

结点由数据成员和指针成员组成。数据成员描述每一个结点的信息；指针成员用来指向链表中的下一个结点。



例：struct StudentNode  
    { string name;  
      float score;  
      StudentNode \* next;  
    };

# C++ 编程基础

## 五、C++程序设计基础

### □ 链表

#### (3) 链表的创建

创建StudentNode类型的动态结点，定义链表头指针（head）和尾指针（tail）。

```
StudentNode * pNew; //创建新的结点
```

```
pNew=new StudentNode;
```

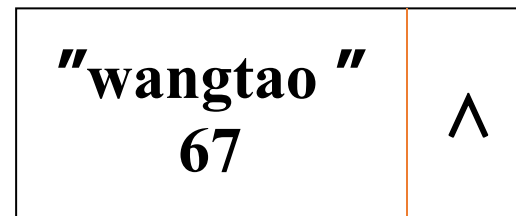
```
pNew→name = "wangtao";
```

```
pNew→score = 67;
```

```
pNew→next = nullptr;
```

```
StudentNode * head = pNew;
```

```
StudentNode * tail = head;
```



head (tail)



# C++ 编程基础

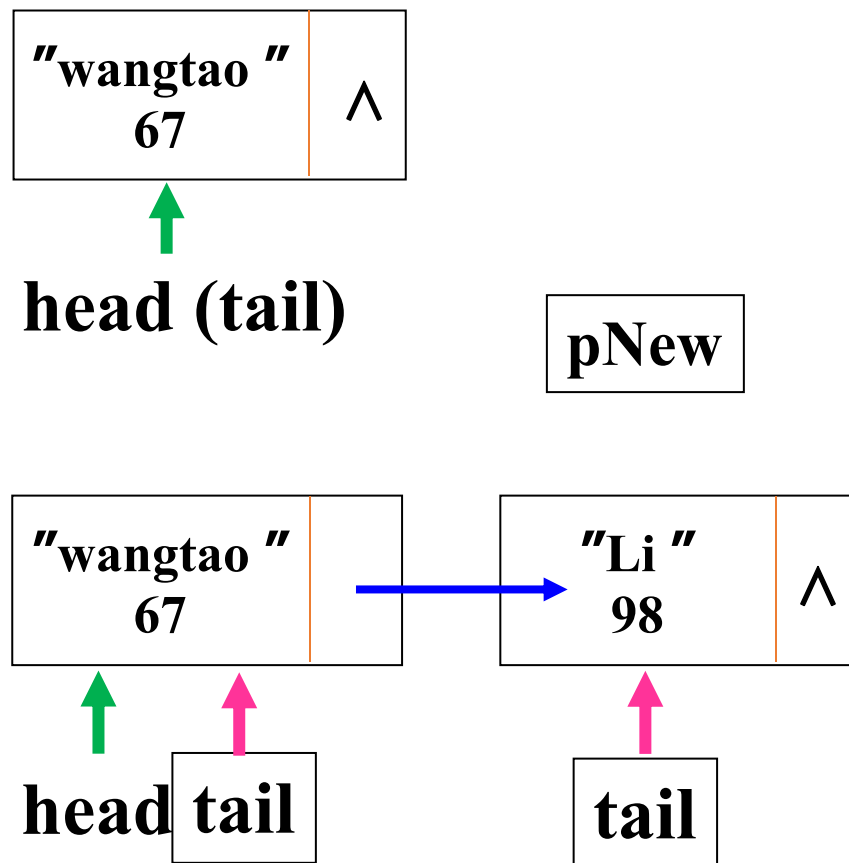
## 五、C++程序设计基础

### □ 链表

在链表末尾插入一个新结点：

```
StudentNode * pNew;  
pNew=new StudentNode;  
pNew→name = "Li";  
pNew→score = 98;  
pNew→next = nullptr;
```

```
tail → next = pNew;  
tail = tail → next;
```



# C++ 编程基础

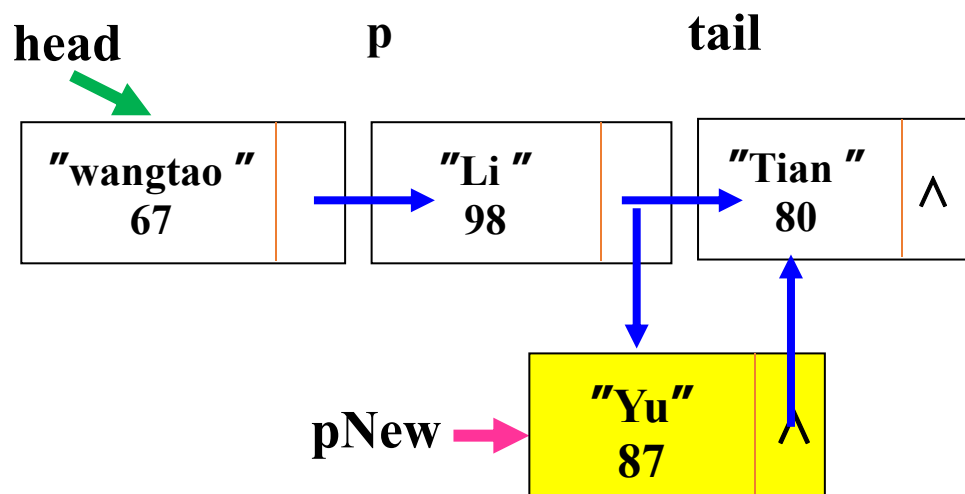
## 五、C++程序设计基础

### □ 链表

在链表中间插入一个新结点：

```
StudentNode * pNew;  
pNew=new StudentNode;  
pNew→name = "Yu";  
pNew→score = 87;  
pNew→next = nullptr;
```

```
StudentNode * p=head;  
while(p!=NULL&& p->next!=NULL){  
    if(p->name=="Li" && p->next->name=="Tian")  
        break;  
    p = p->next;}  
pNew->next=p->next;  
p→ next = pNew;
```



➤ 结点的删除： `StudentNode *pre = head;`  
`pre->next = p->next;`  
`del p;`

# C++ 编程基础

## 五、C++程序设计基础

### □ 引用

引用是给一个对象或变量建立别名

是对象本身，而不是对象的副本，故一个对象改变时，另一个对象也会相应改变

• 定义形式：type & , 例如：

- `int i;`
- `int &j = i;`
- `i = 5;`     //执行后i, j的值均为5
- `j = i + 1;` //执行后i, j的值均为6

• 若： `int i;`

• `int &j; //error`

• `j = i;`

若： `int i, k;`

`int &j = i;`

`j = k; //error`

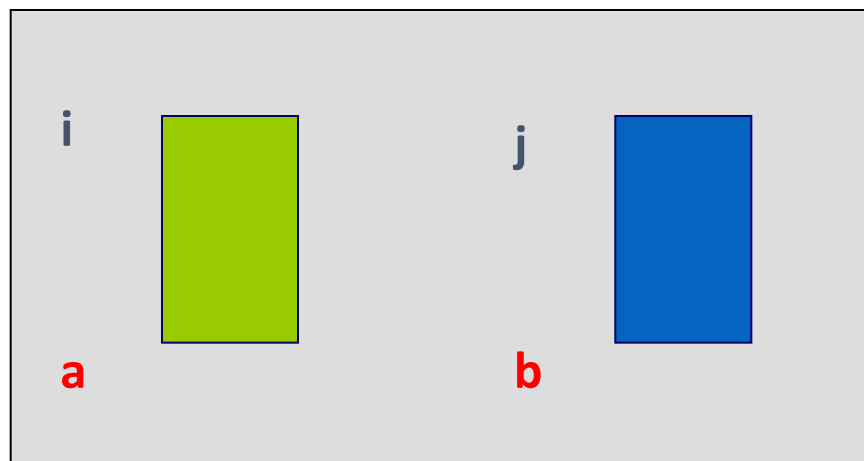


# C++ 编程基础

## 五、C++程序设计基础

### □ 引用

- `#include <iostream>`
- `using namespace std;`
- `void swap(int&, int&);`
- `int main() {`
- `int i = 7, j = -3;`
- `swap( i, j);`
- `cout<<"i="<<i<<"\n";`
- `<<"j="<<j<<"\n";`
- `return 0;`
- `}`



- `void swap(int& a, int& b) {`
- `int t;`
- `t = a;`
- `a = b;`
- `b = t;`
- `}`

# C++ 编程基础

## 五、C++程序设计基础

### □ 引用

//当返回值不是本函数体内定义的局部变量可以返回引用

```
int fn1() {  
    //....  
    return i;  
}
```

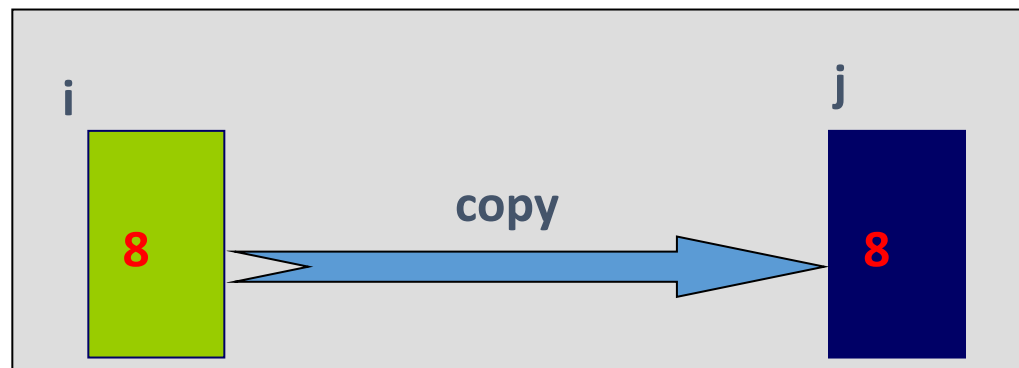
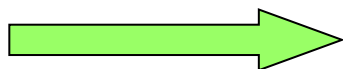
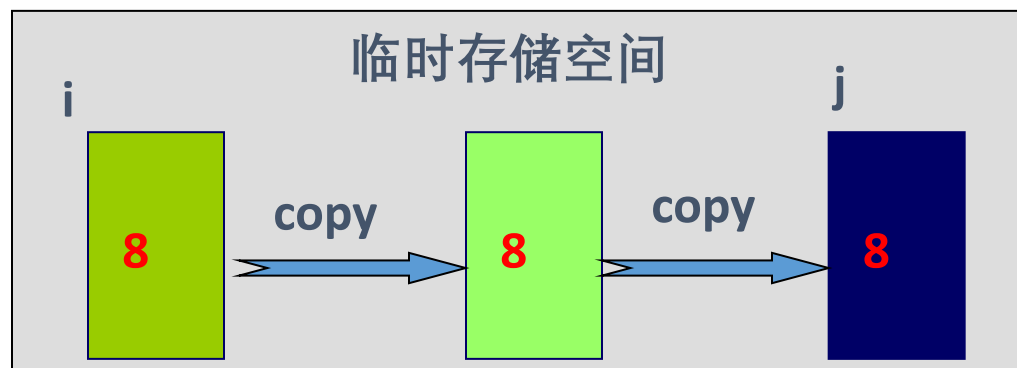
```
j = fn1();
```

```
int& fn2() {
```

```
    //....
```

```
    return i;  
}
```

```
j = fn2();
```



# C++ 编程基础

## 五、C++程序设计基础

### □ 指针和引用

引用的内存模型

```
int a=5;
int &b=a;
```

0012:FF7C      a=5

b

指针变量的内存模型

```
int a=5;
int *pA=&a;
```

0012:FF7C      a=5

↑

0012:FF78      pA=0012:FF7C

# C++ 编程基础

## 五、C++程序设计基础

### □ const的应用

常类型的对象必须进行初始化，而且不能被更新。

- 常引用：被引用的对象不能被更新。

const 类型说明符 &引用名

- 常对象：必须进行初始化，不能被更新。

类名 const 对象名

- 常数组：数组元素不能被更新。

类型说明符 const 数组名[大小]...

- 常指针：指向常量的指针。

## 五、C++程序设计基础

### □ const的应用

- `int main(){`
- `int val=10;`
- `int val2=20;`
- `const int *u=&val;`
- `// *u=20;非法`
- `u=&val2;`
- `}`



## 五、C++程序设计基础

### □ const的应用

```
#include<iostream.h>
void display(const double& r);
int main ( )
{  double d(9.5);
   display(d);
   return 0;
}
void display(const double& r)
//常引用做形参，在函数中不能更新 r所引用的对象。
{  cout<<r<<endl;  }
```

## 五、C++程序设计基础

### □ 程序控制结构

- 顺序结构
- 选择结构if,switch
- 循环结构while,do...while,for
- 控制转向语句break;continue;goto

# C++ 编程基础

## 五、C++程序设计基础

### □ Visual C++.NET集成开发环境

- Visual C++.NET环境为帮助程序开发而设计提供的诸多服务包括：
  - 对开发进程的不同方面，从类和源文件的列表到编译器消息，都提供了观察窗口
  - 访问在线帮助的扩展系统的菜单
  - 创建和维护源文件的文本编辑器，设计对话框用的优秀对话框编辑器，并创建其他界面组件，如位图、图标、鼠标及工具栏的图形编辑器
  - 为程序创建启动器文件的向导在建立新项目的常规任务上提供一个良好的开端。


# C++ 编程基础

## 五、C++程序设计基础

### □ Visual C++.NET集成开发环境

- Visual C++.NET提供的服务：
  - 帮助MFC应用程序创建和维护类的帮手
  - Gallery维护的内置可执行组件给程序增加方便的特征
  - 优秀的调试器
  - 通过对菜单和工具栏对命令进行合理和方便的访问。你可以定制Visual C++中已有的菜单或创建新的菜单
  - 通过宏和附加的动态链接库来添加自己的环境工具的能力。你可以自己开发这些附加项，或从各种各样的供应商那里购买它们

# 第二章：理解OOP-编程语言的历史

- 编程语言的历史
- C到C++
- C++到Java面向对象编程 
- 本章小结

# Java面向对象程序设计语言

## Java语言：纯面向对象编程语言

JAVA语言在设计上很好地借鉴了C++语言，是一种完全“面向对象”的编程语言。JAVA语言的最大优点就是“Write Once, Run Everywhere”

JAVA语言相对C++来说，增加了一些新的特性：

- ❑ 提供了GC，对内存进行自动管理，无需再程序中进行分配、释放内存
- ❑ 不再使用指针，而是采用其他的方法来弥补
- ❑ 与C++相比，JAVA取消了多重继承这个类特性，使得类的继承变得简洁
- ❑ 避免了赋值语句和逻辑语句的混淆，同时取消了其他值与布尔值之间的自动转换，这一点有效地降低了某些运行时错误。

# Java面向对象程序设计语言

## Java语言：纯面向对象编程语言

相比C/C++语言，JAVA语言的几个关键特性：

**(1) 简洁有效。**JAVA语言没有C++语言中难以理解的、容易混淆的特性，例如指针、结构、运算符重载、虚拟基础类等。

**(2) 面向对象。**JAVA语言一门完全面向对象的语言，不支持C语言那样的面向过程程序设计技术。JAVA语言将数据和对数据的操作都封装在一个类中，并提供类，接口和继承机制。

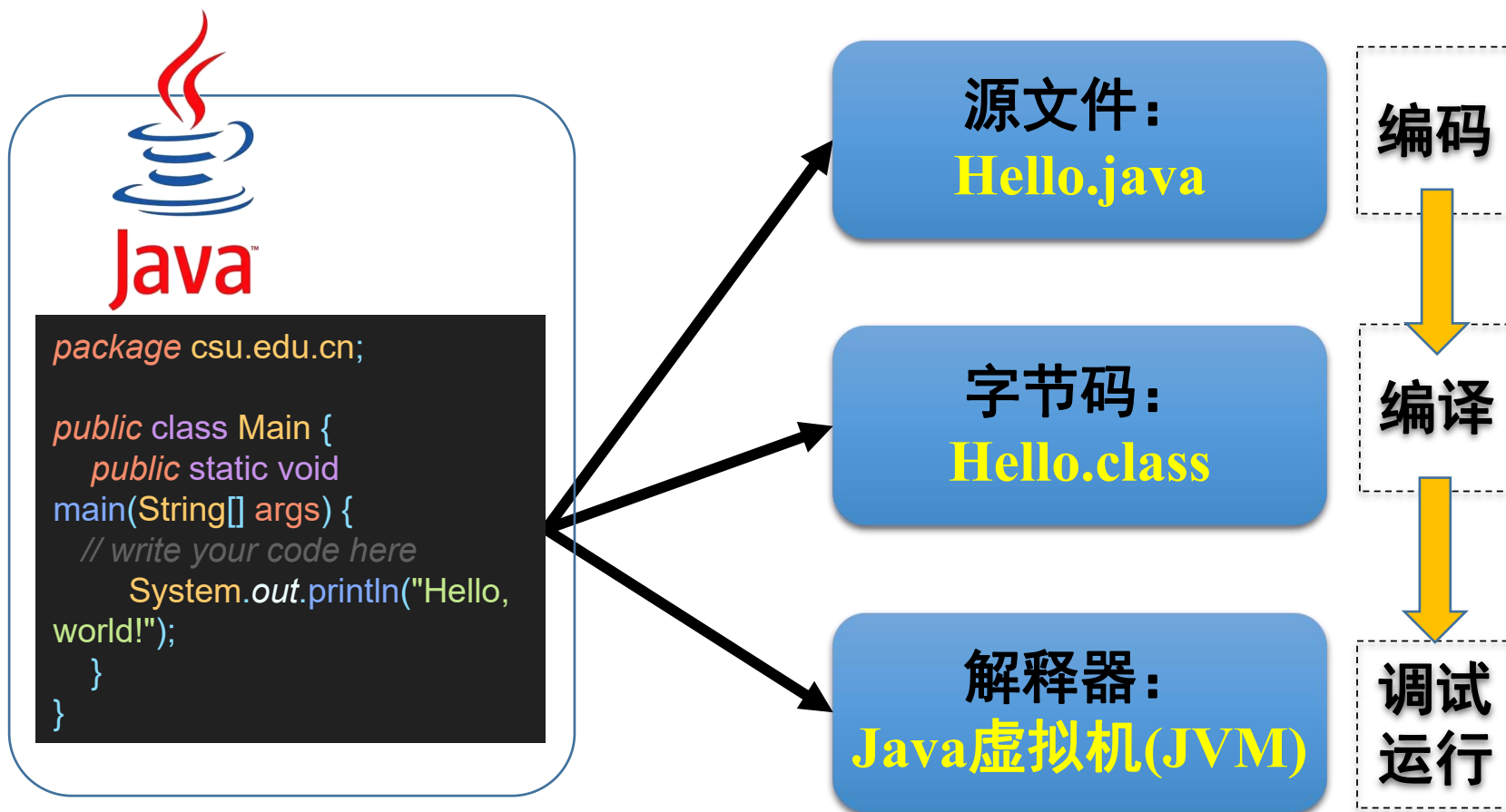
**(3) 可移植性。**JAVA应用程序可以在配备了JAVA解析器和运行环境的任何计算机系统中运行，便于移植。

**(4) 解释型。**JAVA语言是一门解释型语言，相对C/C++来说，JAVA程序执行效率低，速度慢，通过JAVA解释器，对JAVA代码进行解释，实现跨平台目标。

**(5) 适合分布式计算。**Java是适合于网络应用程序开发的语言。

# Java面向对象程序设计语言

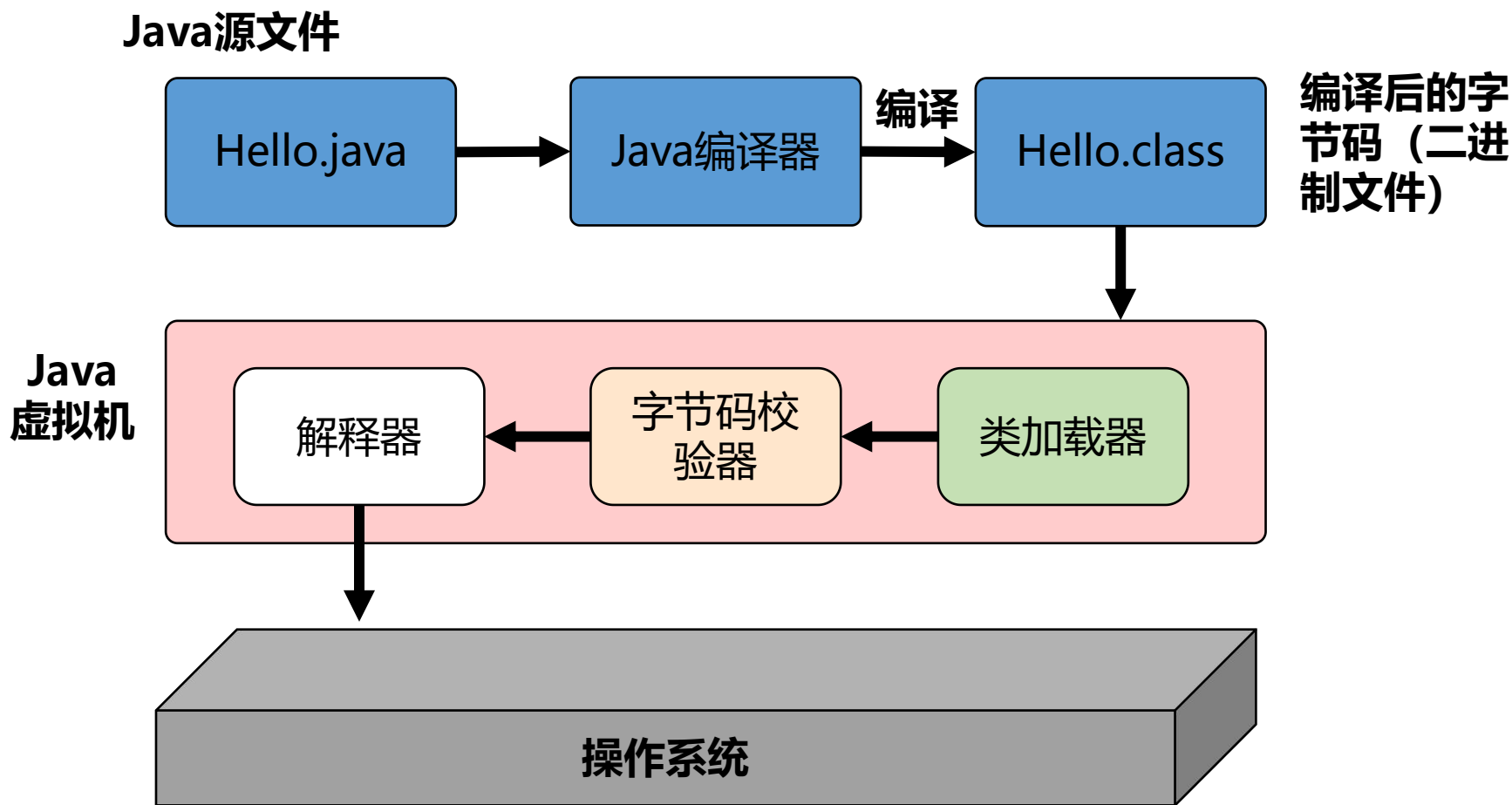
## Java程序执行过程





# Java面向对象程序设计语言

## Java程序执行过程



# Java面向对象程序设计语言

## Java程序是如何运行的：

### □ Java虚拟机

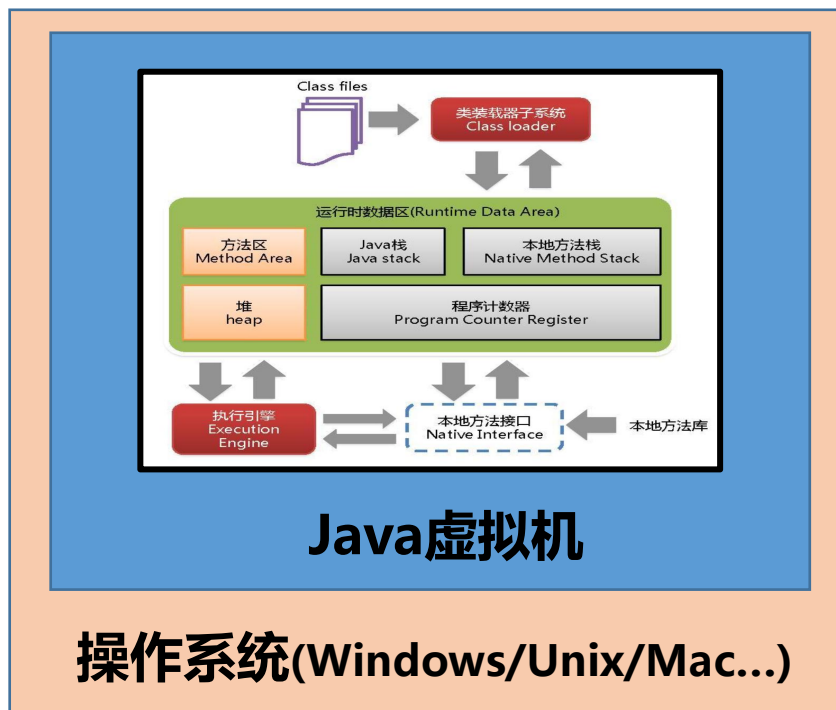
Java虚拟机 (Java Virtual Machine简称 JVM) 是运行所有Java程序的抽象计算机，是Java语言的运行环境，它是Java 最具吸引力的特性之一。



C#



VirtualBox



# Java面向对象程序设计语言

一个简单的Java程序:

// HelloWorld.java文件

```
package csu.edu.cn;  
  
public class Main {  
    public static void main(String[] args) {  
        // write your code here  
        System.out.println("Hello, world!");  
    }  
}
```

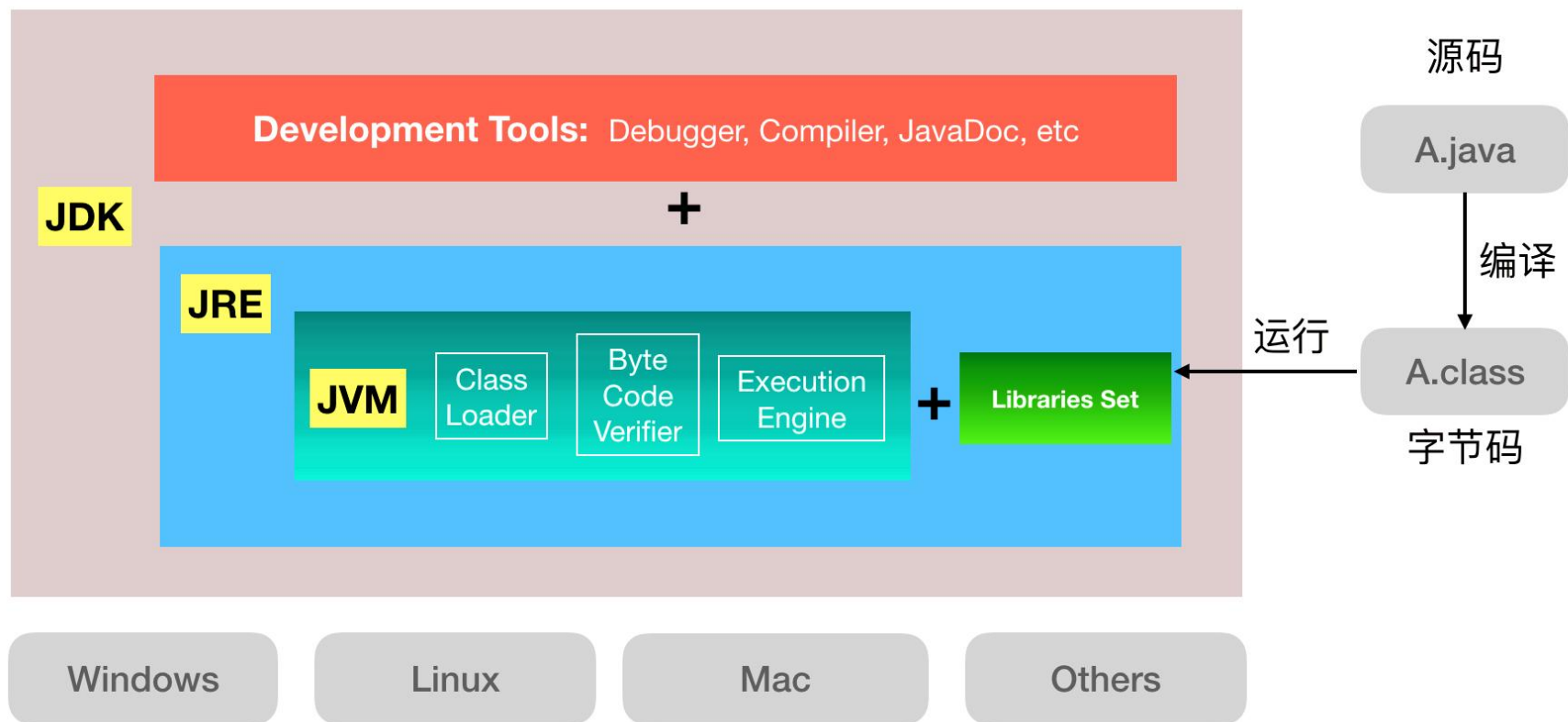
# Java面向对象程序设计语言

## Java程序设计集成开发环境：




# Java面向对象程序设计语言

## Java程序设计集成开发环境：



# 第二章：理解OOP-编程语言的历史

- 编程语言的历史
- C到C++
- C++到Java面向对象编程
- 本章小结 

# 本章小节

- 编程语言的发展历史
- C++编程基础：
  - 数据类型、数组、指针与引用、结构体
  - 链表
  - 控制语句
  - 函数、函数重载、函数模板
- Java面向对象程序设计语言
  - Java的特性
  - Java程序是如何运行的
  - Java的集成开发环境