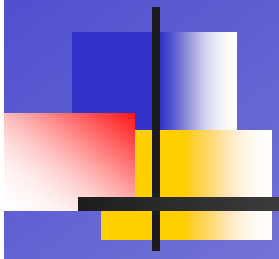
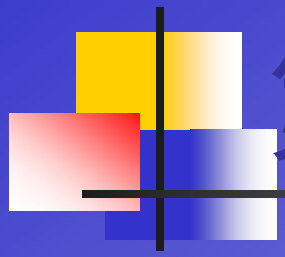


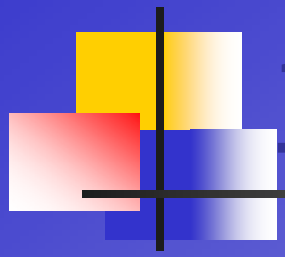
面向对象程序设计

李瑞轩
华中科技大学计算机学院



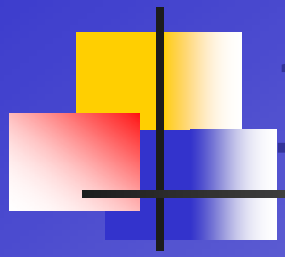
第1章　引　论

- **本章内容：**
  - 1.1　程序设计语言
  - 1.2　程序编译技术
  - 1.3　面向对象的语言及程序设计
  - 1.4　面向对象的基本概念
  - 1.5　C++语言的特点
  - 1.6　C++的程序结构



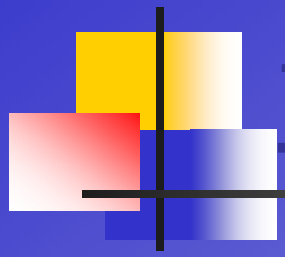
1.1 程序设计语言

- **程序设计语言**
  - 机器语言
  - 汇编语言Microsoft MASM, Borland TASM
  - 高级语言FORTRAN,ALGOL,PASCAL, C, ADA
  - 面向对象的程序设计语言C++, JAVA, C#
  - 数据库查询语言SQL
  - 人工智能逻辑推理语言PROLOG, LISP
  - 元语言BNF



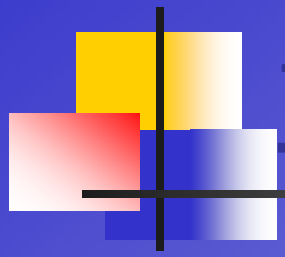
1.1 程序设计语言

- **机器语言**:机器语言程序是由机器指令和数据形成的二进制文档。
- **语言特点**：
  - 效率高
  - 难于理解、编程繁琐、不易维护
- **语言用途**：
  - 编写操作系统、中断服务程序、库函数等需要高效率的场合



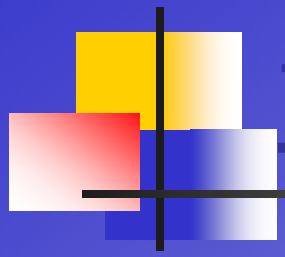
1.1 程序设计语言

- **汇编语言**：符号化的功能上接近于机器语言的语言。
- **语言特点**：
  - 效率高
  - 仍然难于理解、编程繁琐、不易维护
- **语言用途**：
  - 编写操作系统、中断服务程序、库函数等需要高效率的场合。
  - 编写编译程序等系统程序



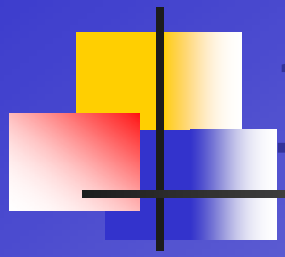
1.1 程序设计语言

- **高级语言**：形式化的类似于自然语言的程序设计语言。
- **语言特点**：
  - 效率略低
  - 易于理解、编程容易、易于维护
- **语言用途**：
  - 编写编译、数据库管理系统等系统程序
  - 编写科学计算，金融商业等应用程序



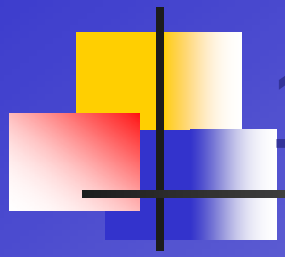
1.1 程序设计语言

- **面向对象的程序设计语言**：是一种面向客观对象描述其特征和行为并通过类封装能够重用和继承的程序设计语言。
- **语言特点**：
  - 面向对象定义新类型，重载、继承
  - 效率更低，但更易维护
- **语言用途**：
  - 系统软件，仿真软件，分布式及网络软件
  - 科学计算、金融商业等



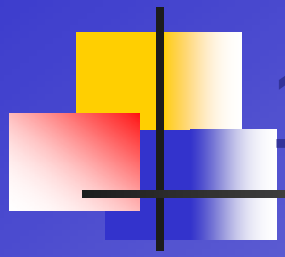
1.1 程序设计语言

- **元语言**：用于描述其它语言语法、语义等语言。
- **元数据**：用于描述其它数据的数据，例如关系型数据库的数据字典。
- **元函数**：用于描述其它函数的函数，例如函数模板。
- **元类型**：用于描述其它类型的类型，例如类模板。



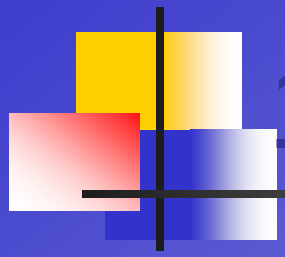
1.2 程序编译技术

- **程序翻译方式**：解释方式、编译方式、混合方式（编译解释）
- **解释方式**：不翻译成机器码，直接读入源程序，语句用内部指令执行
- **编译方式**：翻译成可被计算机直接执行的机器语言程序。
- **混合方式**：先编译成中间语言或虚拟机语言，然后在裸机或操作系统级解释执行。



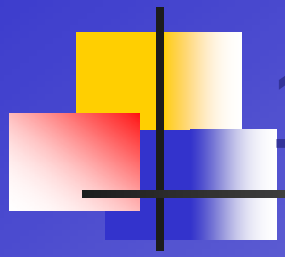
1.2 程序编译技术

- **编译过程**：预处理、词法分析、语法分析、代码生成、模块连接。
- **预处理**：编译宏命令的替换、解释与执行，生成“纯”的高级语言程序。
- **词法分析**：识别程序的保留字，运算符、常量、变量名、类型名、函数名等单词。
- **语法分析**：分析程序的语法结构，检查程序是否合乎语法。例IF…ELSE结构。



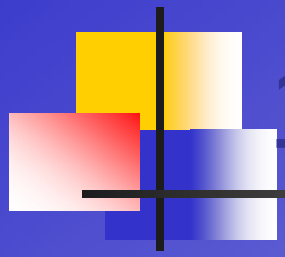
1.2 程序编译技术

- **代码生成**：根据已经识别的程序语义生成和源程序等价的可解释或执行的中间代码。
- **模块连接**：将中间代码同标准库或非标准库连接，生成可被计算机执行的目标语言程序。两种连接：静态连接与动态连接。
- **静态连接**：又称编译时连接，由编译程序完成
- **动态连接**：又称运行时连接，由操作系统完成



1.2 程序编译技术

- **函数绑定**：函数调用同函数的入口地址相联系的过程 。分静态绑定与动态绑定。
- **静态绑定**：又称早期绑定，在程序执行前完成，由编译程序或操作系统装入程序计算函数的入口地址 。
- **动态绑定**：又称晚期绑定，在程序执行过程中完成，由程序自身计算函数的入口地址。
- 静态绑定和动态绑定的函数都可以由静态或动态连接完成。

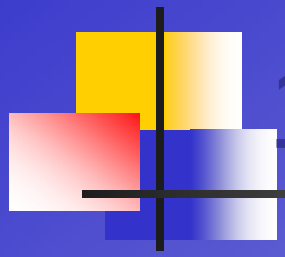


1.3 面向对象的语言及程序设计

- **发展历程**：
  - 封装机制：Ada, Modula-2
  - 对象抽象：Simula
  - 面向对象的语言：Smalltalk-80，C++, Objected Pascal(Delphi), Java, C#, VB .NET
- **C++**:以C为基础的多继承的OOPL
- **Java**:以C++为基础的单继承的OOPL
- **C#**： 以C++为基础的单继承的可重载运算符的OOPL

1.3 面向对象的语言及程序设计

- **四个阶段：**系统分析，系统设计，对象设计，对象实现。
- **系统分析：**理解应用系统的定义及建立应用系统的对象模型。分析和发现对象，定义对象的属性和内部状态变换，定义外部事件和对象之间的联系及约束，最终建立应用系统的对象模型、动态模型及功能模型。
- **系统设计：**确立对象模型的实现方法。分解系统，确定系统固有的并发性，分配处理器的任务，选择数据存储的管理手段，处理全局资源的访问，确立应用系统的边界。



1.3 面向对象的语言及程序设计

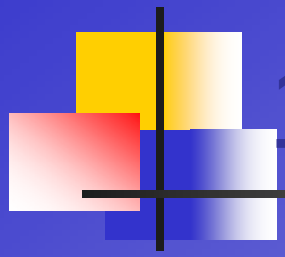
- **对象设计：**将系统分析阶段建立的三种模型转换成类。将对象模型转换为类的属性，将动态模型转换为类的方法，将功能模型转换为主控或子控模块。转换对象模型时要考虑对象关联；转换动态模型应选用事件驱动；转换功能模型时用顺序或事件驱动实现流程控制。
- **对象实现：**用面向对象的程序设计语言实现对象设计阶段定义的类。选择合适的程序语言及开发环境，定义类的属性及其方法，编写主控模块及子控模块，针对具体语言进行性能调整。

1.4 面向对象的基本概念

- **对象**：现实世界中的各种具体的或抽象的“事物”。

现实世界	对象模型	程序设计
特征	属性	数据成员
行为	操作	函数成员

- **简单对象**：简单类型的对象。如int类型的常量或变量。
- **复杂对象**：复杂类型的对象。如class、struct、union类型的常量或变量。

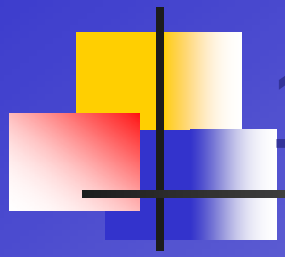


1.4 面向对象的基本概念

- 人类的定义：
- **struct HUMANKIND{**
- **char *name;**
- **double weight;**
- **public:**
- **void eat( );**
- **void wear( );**
- **void reside( );**
- **void travel( );**
- **};**
- 人类的实例：**HUMANKIND** person;

1.4 面向对象的基本概念

- **封装：**是将对象的“属性”和“方法”包装在一起、并对外部提供不同权限的访问接口的机制。
- **意义：**
  - 封装” 定义对象边界，对象的私有“属性”和“操作”被隔离起来，建立系统的对象模型时只需考虑公共“属性”和“操作”，降低了系统模型的复杂程度。
  - “封装” 定义对象接口，公共“属性”和“操作” 对外开放，供其他对象或函数对象访问。在接口不变的情况下，重新定义对象不会影响其他对象的访问。
  - “封装”屏蔽了对象“操作”的细节，从而能够保证核心算法不被泄露，有助于保护知识产权。

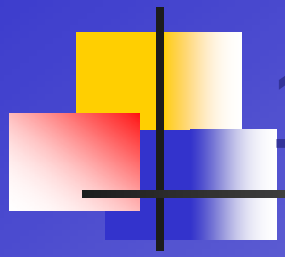


1.4 面向对象的基本概念

- **对象交互的两种形式：**
  - 直接交互：一对象直接调用另一对象的公共“操作”，这种方式在没有消息管理机制的单任务操作系统环境下用得最多。
  - 间接交互：一对象发送消息到消息队列中，由操作系统识别消息应该由哪个对象接受，然后调用相应对象的相关“操作”，在多任务操作系统环境下用得最多。
- **消息要素：**名称，类型，参数。可被响应传递
- **对象标识：**一对象区别于他对象的唯一标志。一般由计算机自动产生，用对象的内存首址做标识。

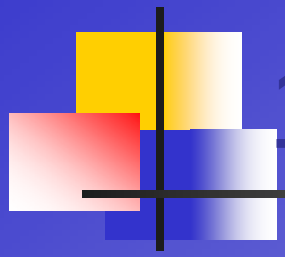
1.4 面向对象的基本概念

- **重载**：又称为编译时多态，指同名函数针对不同参数类型完成不同的功能。重载是多态的一种特例，多态则特指运行时多态。如+5和3+5
- **继承**：指一种类型接受并利用另一种类型的属性和操作。继承可以分为取代继承、包含继承、受限继承和异化继承等方式。
- **抽象**：指从事物实例得到事物的共同属性和操作从而形成事物类型描述的过程，或从若干低级的事物类型不断提炼形成高级或更具普遍意义事物类型描述的过程。



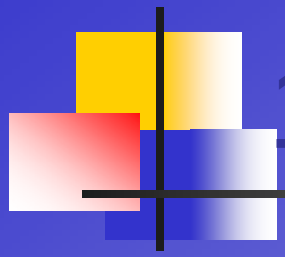
1.4 面向对象的基本概念

- **取代继承**：派生类对象完整地继承了所有基类的所有“属性”和“操作”，且没有修改、增加新的“属性”和“操作”。
- **包含继承**：派生类对象完整地继承了所有基类的所有“属性”和“操作”，并增加了新的“属性”和“操作”。
- **受限继承**：派生类对象部分地继承了基类的“属性”和“操作”，并且没有增加新的“属性”和“操作”。
- **异化继承**：派生类对象继承了基类的“属性”和“操作”，并对原有“属性”和“操作”进行了修改。



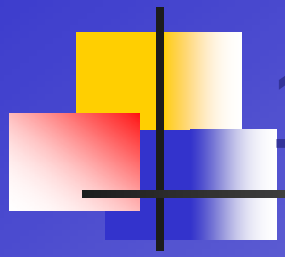
1.5 C++语言的特点

- C语言超集，代码质量高、速度快、可移植性好。
- 强类型语言，编译阶段就能发现程序潜在错误，不会将错误带到运行阶段 。
- 表达能力强，C++的多继承是JAVA等语言所没有的。
- 运算符重载，对象的运算更易表达且表达更加自然。
- 抽象能力强，函数模板和类模板提供更高级别的抽象。
- 内存管理高效，C++提供自动和人工回收两种方式。
- 异常处理加强，支持对象类型的异常。
- 支持名字空间，更加有利于大型软件工程项目。
- 非纯面向对象的语言，同时支持对象和模块描述结构。



1.6 C++的程序结构

- **完全兼容C：**C的语法、语义以及标准库函数都可在C++中使用，不适用的语义加以警告
- **强类型限制：**函数、变量必须先说明再使用
- **函数原型：**即函数名、参数及返回类型的说明。
  - 头文件包含函数、变量的说明。scanf等标准函数必须先include，起先说明的作用：
  - `int  scanf(const char *,  ...);`
  - `int  printf(const char *,  ...);`
- **文件结束：**UNIX等CTRL+D;MS DOS等CTRL+Z
- **注解：**/*...*/或//直到当前行结束

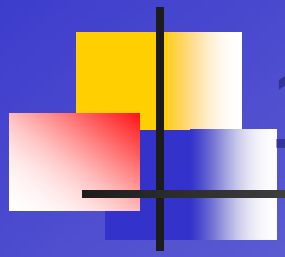


1.6 C++的程序结构

【例1.2】输入n，计算n!及其十进制有效数字位数，并打印计算结果。

说明：iostream.h文件声明标准输入流类变量cin，以及标准输出流类变量cout，重载的运算符>>和<<分别为cin和cout提供输入和输出功能。

```
#include  <stdio.h>
#include  <iostream.h>
void  main(void)
{  int  i, n;     long f;
    cout<<"Please input n: ";
    cin>>n;                    //注意，不能像scanf那样使用&n
    for(f=i=1; i<=n; i++)  f*=i;
    cout<<"\n"<<n;
    n=printf("%ld", f);   //注意，printf返回n!的有效位数
    cout<<"!=\nNumber of digits="<<n;
}
```



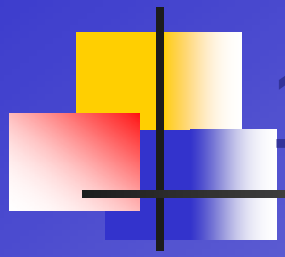
1.6 C++的程序结构

关于运算符重载和连续计算：

（1）cout<<"Please input n: "输出字符串"Please input n: "，运算结果为cout。

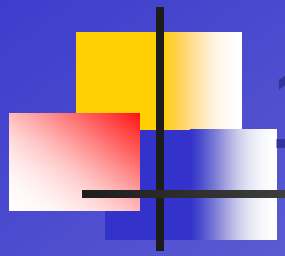
（2）正如C的加法可以连续相加，上述结果cout也可以再次进行<<运算。cout<<“\n”<<n就是连续进行<<运算的例子。

（3）cout<<“\n”输出换行并得到结果cout；运算结果cout再次和<<n结合得式cout<<n，该式输出整数n并再次得到结果cout。



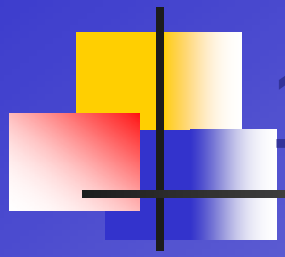
1.6 C++的程序结构

- 过时语义：结构或联合中标识符和结构名本身有相同作用域。
- **struct CALENDAR{**
  - **enum SEASON{spr, sum, aut, win};**
  - **SEASON season;**
  - **int year, month, day;**
- **};**
- **SEASON happy=CALENDAR::spr; //警告，C的枚举**
- **CALENDAR::SEASON blue=CALENDAR::win;//C++提倡**



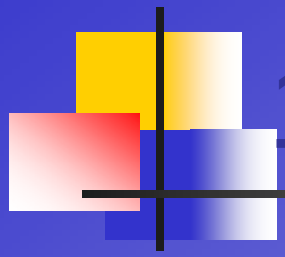
1.6 C++的程序结构

- 标识符可说明多次，但只能定义一次。
  - 变量定义：类型、名、初值。
  - 函数定义：返回类型、函数名、函数参数、函数体。
- 建立项目的方法：
  - 形成可分开编译的函数模块和类模块；
  - 对于各模块使用的变量、函数或对象，如果它们不为整个程序所共享，就将它们用static定义为局部于程序文件或者函数体内；
  - 根据所用函数建立包含文件，并形成一个总体包含文件提供给所有程序文件。包含文件只能是说明性的变量、函数原型，而非其定义。



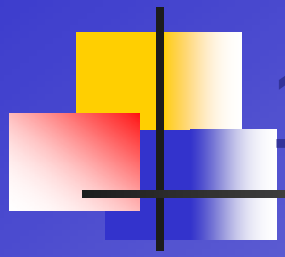
1.6 C++的程序结构

- 文件project.h：
- #include <iostream.h>
- #include "stack.h"
- extern int x, y;
- int max(int x, int y)；
- int min(int x, int y) ;



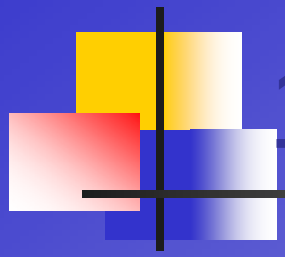
1.6 C++的程序结构

- 文件stack.h定义类的接口：
- class STACK{
- int *stack, sp, top;
- public:
- STACK(int);
- ~STACK( );
- int push(int), pop(int&);
- };



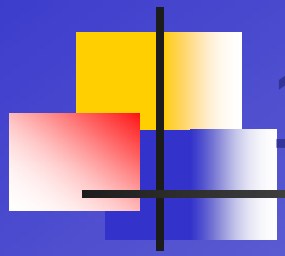
1.6 C++的程序结构

- 文件stack.cpp用于定义函数体:
- #include "project.h"
- STACK::STACK(int size)
- {stack=(int*)malloc(sizeof(int[top=size])); sp=0; }
- STACK::~STACK( ) { free(stack); }
- int STACK::push(int v)
- {if(sp<top){stack[sp++]=v; return 1;} else return 0;}
- int STACK::pop(int &v)
- {if(sp>0) {v=stack[--sp]; return 1;}else return 0;}



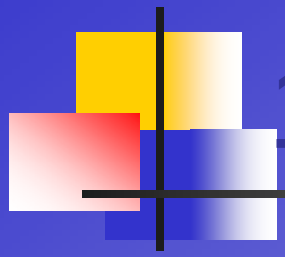
1.6 C++的程序结构

- 文件module.cpp：
- #include "project.h"
- static int z;
- int x, y;
- int max(int x, int y) { return x>y?x:y; }
- int min(int x, int y) { return x>y?y:x; }



1.6 C++的程序结构

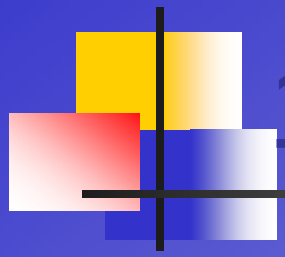
- 文件project.cpp：
- #include "project.h"
- void main(void)
- {
- STACK stk(20);
- cout<<"This is a example of project building";
- y=x=3;
- }



1.6 C++的程序结构

- **换名机制：**C++与C编译程序的换名机制不同。C++程序如要调用C函数，必须在C++程序中用extern “C”说明要调的C函数原型。
- C++文件main.cpp：

```
#include <stdio.h>
extern "C" long sum(int n);
void main(void)
{  int n;
   printf("Please input n:");
   scanf("%d",&n);
   printf("sum(%d)=%ld", sum(n));
}
```



1.6 C++的程序结构

- C程序文件sum.c：
- long sum(int n)
- { int i;
- long s;
- for(s=i=0; i<=n; i++) s+=i;
- return s;
- }