CPP!
C语言

@文歆潼

不对相的地方请多包涵!!
请雅正——

Lec1

20% Project
20% Group Pro
Week9: mid-term

有用的网站: http://cpp.sh/
http://www.onlinegdb.com/

计算机语言: 汇编语言, 机器语言　　高级语言: 人能更懂

工具　GCC: GNU Compiler Collection 编译器

编程阶段: edit 编辑
Preprocess 预处理
Compile 编译
Link 连接
Load 加载
execute 运行

解释一下: in Cygwin 子系统　vs中./a.out 相当于 Win中的b.exe, 也可以跑!

g++ ……cpp → (./a 可以跑) g++指令可以提供 link 和 compile.

g++ -c ….cpp 编译原文件, 生成 object 文件 …o

g++ -o name file.o 连接 object 文件, 产生可运行文件 …exe

CPP程序样例

内容
Comments
Preprocessor 预处理 引入头文件
Function header 函数的开头
Function body 函数本体

Argument list Namespace　Statements

```
// myfirst.cpp -- displays a message
#include <iostream>                    // a PREPROCESSOR directive
int main()                             // function header
{                                      // start of function body
    using namespace std;               // make definitions visible
    cout << "Come up and C++ me some time.";  // message
    cout << endl;                      // start a new line
    cout << "You won't regret it!" << endl;   // more output
    return 0;                          // terminate main()
}                                      // end of function body
```

名 命 分号 Semicolon

terminate function
Return statement

注释: // 你好吗。
/* 这是注释 */

Identifiers 标识符
- 字母, 数字和 下划线.
  不要以数字开头
- 认大小写! (数据库的不认)
- 不为 keyword

Library 库
可以用现成函数.

main()函数
① 函数头, 总结接口
② 函数体, 具体完整的指令需要加分号;
return是终结函数!

预处理 Preprocessor
① 编译前先转换源代码
将输出传递给编译器.
② 可以用一行指令控制
③ 不用分号, 用#
#include #define #pragma
#include <iostream>
#include <boost/tokenizer.hpp> 文件路径
#include "　　"
④ 也可以定义宏. #define getmax(a,b) a>b?a:b
⑤ 条件编译　#if ..

◦ Keywords

alignas	default	noexcept	this
alignof	delete	not	thread_local
and	do	not_eq	throw
and_eq	double	nullptr	true
asm	dynamic_cast	operator	try
auto	else	or	typedef
bitand	enum	or_eq	typeid
bitor	explicit	private	typename
bool	export	protected	union
break	extern	public	unsigned
case	false	register	using
catch	float	reinterpret_cast	virtual
char	for	return	void
char16_t	friend	short	volatile
char32_t	goto	signed	wchar_t
class	if	sizeof	while
compl	inline	static	xor
const	int	static_assert	xor_eq
constexpr	long	static_cast	override*
const_cast	mutable	struct	final*
continue	namespace	switch	
decltype	new	template	

头文件名 Header Filenames

C++: iostream

C: cmath ← prefix

命名空间 namespace

① 简写大程序

② 组织几家公司的预存码

microflop::wanda("go!");

③ std::cout<<"..."; std::cout<<std::endl

using namespace std 整个操作都可用

using std::cout; //让cout可用

cout: C++的输出 ...

① 一个在stream中写好的object

String用双引号，用 << 插入，用 endl 和 \n 换行。

cout<<"....";

cin 读取输入!

cin>>a;

指将读入的内容赋予a;

它们是iostream中预定义对象，将输出输出视为字符流，<< 允许数据插入输出流中，>> 允许从输入流提取信息，

<cmath>里 pow( ), sqrt( )也好用，

declaration 声明 创建一个变量

储存类型 位置标志

int number;

assignment 赋值 向变量提供值.

number=10; ← 算术版

a=b=c=1; ← 连续版

编译器分配它的存空间;)

一些Linux基本命令

ls	显示此目录文件列表
ls-a	显示包括隐藏文件的文件
ls-l	显示文件属性(大小,日期,名号连接,是否读写与执行)
cd dir	切至当前目录下的dir目录
cd /	切至根目录
cd ..	切至上一级目录
cd ../..	切至上两级目录
cd ~	切至用户目录
rm file	删除某一个文件
rm -fr dir	删列dir的整个目录
cp sourse target	将s复制为t
cp /root/source	将/root下的s文件复制到当前目录.

◦ 函数 Function

Header 头

body with 体

multiple statement

? argument

会返回一个值 return

此值类型于header给出.

需要函数原型 prototype

在main前给上面

定义函数不必声明，否则要在main()前面调用.

一个程序的组成

① 自己创建的文件
- .h文件 - .cpp文件

② 标准文件 included

③ 函数库文件 included <.lib文件 对象文件中的代码>

④ 动态链接库文件：.dll

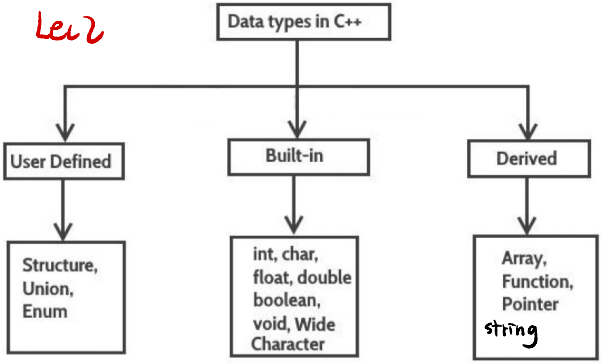
函数的组成

① 函数原型

② 函数头 { argument, name, return type }

③ 函数体 { 语句, 返回语句, pair of braces }

Lec2



OOP: object Oriented Programming 面向对象的程序设计

OOP的本质是设计和扩展自己的数据类型

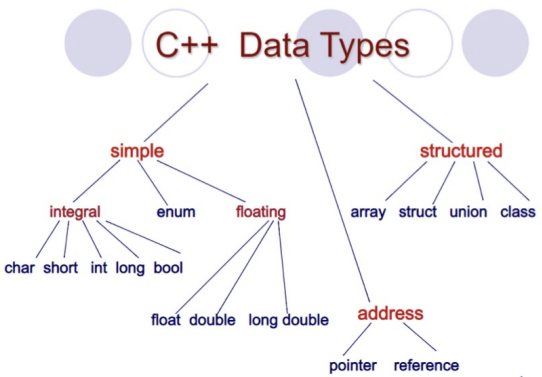
Built-in类型 会是你的构建块.

基本类型 整数, 浮点数
integers floating-point

复合类型 数组, 字符串, 指针, 结构
arrays strings pointers structures

注意：地址是一种变量 in C/C++

Void是一种函数类型



变量 variable

- 信息应存储在哪？存储什么类型的信息？什么值被存？

我们声明一个变量，以类型描述这个信息，变量名指示值。程序会定位一块足够大的内存来容纳一个整数，记下位置，并将值复制至此。

给变量起名也不要数字开头，用keyword做，尽量起meaningful的！

整数 integer

- char (1)
- short (2), int (4), long (4), long long (8)

通常选int，而short可节省内存空间。

差异在内存量上，宽度来描述差异，有符号/无符号 ②

Bits 比特

电脑内存的单位为比特，一比特像电子开关 { off: 0, on: 1 }，8比特可以被设为 2^8 = 256个不同值

Bytes 字位

通常为8比特，是char的基本大小。

千 1 kilobyte = $1024 = 2^{10}$ bytes；1 megabyte = 1024 kilobytes 北

4北 1 gigabyte = 1024 megabytes；1 terabyte = 1024 gigabytes 北北

宽度取决于CPU、OS与Complier.

sizeof 操作可以得到长度，climits头文件中有很多特号常量，比如 INT_MAX，LLONG_MAX. 最值的

字符 char　用单引号

存储字母、加字母、标点和数字.

- 对字母用数字代码
- char是另一种整数hhh
- Ascii表很重要!
- 国际unicode字符集!

		AsciiCode
换行	\n	10
tab	\t	9
backslash\	\\	92
问号	?	63

布尔bool

预值有 { true, false }，也可以转换为 int：

```
int b = false; b=0
int a=true; a=1
```

int也是可转换为bool的，非零为true，零为false.

const常值

使用 const 后，它是个不可改变的常值！ read only!

浮点数 Floating-Point Numbers

C++基于二进制数.

会分成两部分为储存，一个代表值，另一部为向上或下缩放该值，移动小数点.

算术符

+ 加　*乘　%取余
- 减　/除　用括号表示优先.

Cast 转换

(long) thorn　long (thorn)　结果都为 a type long conversion of thorn

auto

允许编译器从初始值推断类型！

```
auto n=100; // n is int
auto x=1.5; // x is double
```

操作符 Operators

- Arithmetic 算术　+, -, *, /, %
- Assignment 赋值　=, +=, -=, *=, /=, %=
- Relational 关系　==, !=, >, <, >=, <=
- Bitwise 位运算　&, |, ^, ~, <<, >>
- logical 逻辑　&&, ||, !
- Auto-increment decre 自增 自减　++ --
- Ternary 三元的　?:
- 其它: comma sizeof

printf 打印

C文件

- d 有符号十进制整数
- o 有符号八进制整数
- f 浮点数
- c 字符

读取 scanf

C文件

```
int a;
scanf("%d", &a)
float c;
scanf("%f", &c)
char str[20];
scanf("%s", str);
```

空格, tabs 和空行都会断开读入.

读什么读都行.

Example:

```
int a=1234;
float f=123.456;
char ch='a';
printf("%8d,%2d\n",a,a);
printf("%f,%8f,%8.1f,%.2f,%.2e\n",f,f,f,f,f);
printf("%3c\n",ch);
```

对齐8格

Sample output:

```
    1234,1234
123.456000,123.456000,   123.5,123.46,1.23e+02
  a
```

Cout

C++ 输入

a=3; c=4.5;　　换行结束

cout << "a=" << a << ",c=" << c << endl;

a=3,c=4.5

```cpp
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    cout.setf(ios base::fixed, ios base::floatfield);
    cout << 56.8 << setw(12) << 456.77 << endl;

    cout << setprecision(2) << 123.356 << endl;
    cout << setprecision(5) << 3897.6784385 << endl;

    return 0;
}
```

iomanip manipulator

Setw(12) 对后者起效

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab02$ g++
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab02$ ./a.
56.800000  456.770000
123.36
3897.67844
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab02$
```

12位

上面是 #include <iostream>
using namespace std;

- width(n) 设置宽度为n
- fill('x') 填充补齐
- percision(n) 保留n位小数

```cpp
    cout << 56.8;
    cout.width(12);
    cout.fill('+');
    cout << 456.77 << endl;

    cout.precision(2);
    cout << 123.356 << endl;
    cout.precision(5);
    cout << 3897.678485 << endl;

    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2
56.800000++456.770000
123.36
3897.67848
```

C++ cin 输入

空格、换行、制表符都可以做分隔!

cin >> a; 忽略空白

注 cin.getline(str,20)　cin.get(str,20) 不被空格挡下

!getline() 丢弃换行符，而get()不会.

C语言 fgets(str)、put(str)

char str[20];

gets()读一行，只读string 会被建议换 fgets()

```cpp
    char str[20];

    cout << "Enter a string:";
    cin.get(str, 20);
    cout << "You entered: " << str << endl;

    cin.get();
    cout << "Enter an other string:";
    cin.getline(str, 20);
    cout << "You entered: " << str << endl;

    return 0;
}
```

要加这个才有新读入，否则程序会↓

Enter another string: You entered:

C++ string datatype

string str;

getline(cin, str);

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab
maydlee@LAPTOP-U1MO0N2F:/mnt/d/csourcecode/2021Spring/lab
Enter a string:C and C++
You entered: C and C++
Enter an other string:Programming is fun.
You entered: Programming is fun.
```

若 int a;
a=cin.get(); b=cin.get();
输入"空格""回车". ⇒ 32　10
会转整型，其ascii返回值

Lec3

Array 数组

要素：① 数组内元素类型 ② 数组名字 ③ 元素数目为整数常数 ④ 用方括号 square brackets [ ]

若 sizeof(数组名) 会得到整个数组的字节数要得到大小必得靠 .size

声明 → 赋值 → 初始化

可以在定义数组时初始化

不初始化会为0，也不可以把数组赋给别的数组。

```
➢ Cannot
  ✓ Use initialization later
  ✓ Assign one array wholesale to another

int cards[4] = {3, 6, 8, 10};   // okay
int hand[4];                    // okay
hand[4] = {5, 6, 7, 9};         // not allowed
hand = cards;                   // not allowed
```

```
Yes, you can!

float hotelTips[5] = {5.0, 2.5};
long totals[500] = {0};
short things[] = {1, 5, 3, 8};
```

初始化好后可 cards[1]=5; 修改它的内容

多维数组

矩形数组

```
int arr[3][2] = { {1,5}, {4,0}, {9,1} }; cout << "test[0][1]=" << arr[0][1] << endl;
```

-5

字符数组 C-string

```
char str[] = "C++";
char str[4] = "C++";   ← 字符串自带\0
char str[] = {'C','+','+','\0'};
char str[4] = {'C','+','+','\0'}
```

三种定义方式

[C]

strcpy(dst, ".....");

Strings 字符串

存于连续的字节内存 ie. char[] 以 '\0' 结尾 (C style)

使用双引号就相当于带了\0在结尾

头文件 #include <string>

C++允许两个字符串（空格分开）相连

☆ 小心 overflow

cin.getline() 可以读入一整行 会丢弃换行符

使用cin.get()读多个单词会把\n也读

会把换行符在输入队列中保留.

structure 结构

用户可以定义的类型!

```
struct Ultraman{
    char name[25];
    int  age;
};
```

在main里或外定义都ok ie. 可以有成员函数

可以建结构的数组，以 { "...", ... } 去赋值，逗号分开成员变量.

```
#include <iostream>
using namespace std;

struct Person  //structure declaration
{
    char name[20];
    int age;
    float salary;
};

int main()
{
    Person p1;
    Person p2 = {
        "Glorious Gloria",   //name value
        23,    //age value
        1034.9   //salary value
    };

    cout << "Enter full name:";
    cin.get(p1.name, 20);
    cout << "Enter age:";
    cin >> p1.age;
    cout << "Enter salary:";
    cin >> p1.salary;

    cout << "\nDisplaying Information:" << endl;
    cout << "Name: " << p1.name << endl;
    cout << "Age: " << p1.age << endl;
    cout << "Salary: " << p1.salary << endl;

    return 0;
}
```

Declare a structure — 声明

Define a structure variable — 定义

Define and initialize a structure variable — 赋值

Access a structure members use . operator — 点"." 可以赋成员值

union 联合

可以拥有好几种数据类型，但一次一种.

赋一次值丢一次上一个值.

```
union one4all
{
    int int_val;
    long long_val;
    double double_val;
};

one4all pail;
pail.int_val = 15;        // store an int
cout << pail.int_val;
pail.double_val = 1.38;   // store a double, int value is lost
cout << pail.double_val;
```

Enumeration 枚举

创建一些符号常量，赋值范围限制此枚举.

重点、 Lec4

Pointers 指针

基本属性
- 信息存哪儿
- 存了什么值
- 存了什么类型

地址符 & 可以访问地址，以16进制表示

星号 * 逆向，间接值

*地址 = 值

&值 = 地址

C/C++的重要内容是内存管理 ∴ 指针很重要

```
int main(){

int a;
a=18;
int * b=&a;
cout<<"a:"<<a<<endl;
cout<<"&a:"<<&a<<endl;
cout<<"b:"<<b<<endl;
cout<<"*b:"<<*b<<endl;
cout<<"&b:"<<&b<<endl;

}
```

a 和 *b 为值

&a 和 b 为 a 的地址

&b 为 b 的地址

```
a:18
&a:0x7fffeff4ae8c
b:0x7fffeff4ae8c
*b:18
&b:0x7fffeff4ae90
```

注意：指针是分配已内存装地址，没有给数据留内存。 `int * ptr; // NULL`

C++中 segmentation fault `*ptr=22; // 更不妙`

若不幸是无效内存引用，注意指针。↑编译*ptr和ptr都不可

New

`int *ptr=new int;` `*ptr=2;` ✓

告诉new你想为什么类型腾空间！

它会返回给指针:~

C语言:free( )

delete

new的

删指针来让内存释放

ps. 不可以删普通变量！

删数组要加 delete[]，new[]也一样

- 指针与整数的异同

指针实是可做加减的整数，但非整数类。

也不能单纯将int赋给指针，但可以…

`int * ptr = (int *) 0xB8000000.` (不太好)

注：括号数组表示法相当于取消引用指针☺

Dynamic Array 动态数组

指针指向首元素

`double *p3= new double[3];`

p3=p3+1; p3=p3-1; ∴地址可增减

↑下一个 ↑返回去

向struct的指针

```
struct Man{ int age }
Man m, *ptr;
ptr = &m;
```

创指针 ptr指向man变量

`(*ptr).age` `ptr -> age` `= m.age` 都可访问struct成员

用同

- `short A[10];`
  A为short指针，&A为指向10个short的数组的指针
- `short (*pas)[10]=&A;`
  (*pas)=A为short指针，pas=&A为指向10个short的数组的指针
- `short* pas[10];`
  pas为含有10个short指针的数组。

注：Lec4最后4面没看

```
double  *pd;          // pointer to double
char **ppc;           // pointer to pointer to char
int *ap[15];          // array of 15 pointers to ints
int (*fp)(char *);    // pointer to function taking a char* argument, returns an int
int *f(char *);       // function taking a char* argument, returns a pointer to int
```

char 指针

无括号

```
Displaying address using array:
&arr[0] = 006FFCD0
&arr[1] = 006FFCD4
&arr[2] = 006FFCD8
&arr[3] = 006FFCDC
&arr[4] = 006FFCE0
Displaying address using pointer:
ptr + 0 = 006FFCD0
ptr + 1 = 006FFCD4
ptr + 2 = 006FFCD8
ptr + 3 = 006FFCDC
ptr + 4 = 006FFCE0
Displaying values of elements using pointer:
*(ptr + 0) = 0
*(ptr + 1) = 2
*(ptr + 4) = 4
*(ptr + 3) = 6
*(ptr + 4) = 8
```

int arr[5];
int * ptr;

←ptr= arr;

每次+1 移3个字节

括号括起来！

指针与数组

不加括号则：

*ptr +1=1

*ptr +2=2

因为*ptr =0

`char* pvalue = NULL;` 指针

`pvalue = new char[20];` 求空间存变量

`delete [] pvalue;`

`double (*pvalue)[4] = null;` 指针

`pvalue= new double[3][4];` 分配3x4空间

或：

```
int **p;
p = new int*[3];
for(int i=0;i<3;i++) p[i]=new int[4];
```

newarray.cpp > ...

```cpp
#include <iostream>
using namespace std;

int main()
{
    int * pArray = NULL ,*t ;
    pArray = new int [10] ;
    if ( pArray == NULL )
    { cout << "allocation failure.\n" ;
      exit(0) ;
    }
    for ( int i = 0 ; i < 10 ; i ++ )
        pArray[i] = 100 + i ;

    cout << "Displaying the Array Content" << endl;
    for ( t = pArray ; t < pArray + 10 ; t ++ )
        cout << *t << "  " ;


    delete [] pArray ;

    return 0;
}
```

指针实例！

t也是

pArray为指针，

Allocate the memory to store 10 integers, and assign its address to the pointer **pArray**.

对P new：分配空

Assign 10 values to the memory by the pointer **pArray**.

通过指针赋值存入memory

memory leak 是内存泄漏

If you access the value by * operator, be sure do not move the pointer which assign the address by new.

Segmentation fault (core dumped)

☆ pArray 是用new来赋值 → 不要对它 ++ 否则最后pArray会超出范围

Release the memory.

Sample out:

```
Displaying the Array Content
100  101  102  103  104  105  106  107  108  109
```

short tell[10]

tell为指向short的指针；

数组名平时创建不是数组是指针，可以 tell[i]与*(tell+i)取值！

&tell 是指向有10个short的数组的指针；

short (*pas)[10] = &tell

(*pas) = tell 为指向short的指针；

pas = &tell 是指向有10个short的数组的指针，

short *pas[10]

pas是一个有10个指向short的指针的数组

3. Dynamic Memory Allocation for Structures

newstructure.cpp > ...

```cpp
#include <iostream>
struct inflatable   // structure declaration
{
    char name[20];
    float volume;
    double price;
};

int main()
{
    using namespace std;
    inflatable *ps = new inflatable;   // allocate memory for structure

    cout << "Enter name of inflatable item: ";
    cin.get(ps->name, 20);      // use -> to access the member
    cout << "Enter volume of cubic feet: ";
    cin >> (*ps).volume;        // use (*). to access the member
    cout << "Enter price: $";
    cin >> ps->price;

    cout << "Name: " << (*ps).name << endl;
    cout << "Volume: " << ps->volume << "cubic feet\n";
    cout << "Price: $" << ps->price << endl;

    delete ps;          // free memory used by structure

    return 0;
}
```

Create an unnamed structure of the inflatable type and assign its address to **ps** pointer using **new** operator

Access the structure members using -> or (*).

Release the memory.

Sample output:

```
Enter name of inflatable item: Black Base
Enter volume of cubic feet: 35.4
Enter price: $91.25
Name: Black Base
Volume: 35.4cubic feet
Price: $91.25
```

structure 的指针

man M;

man *p = new man;

(*p).name → M.name

p→name 都可以！

结构数组

man *pA;

pA = new (nothrow) man[3];

失败不异常

pA[0].Name = ...;

automatic storage 自动存储

LIFO ↓

函数中定义的普通变量使用它，存于栈中，称为自动变量。

Lifetime：函数结束时

manage data in memory

static storage 静态存储

整个程序执行过程中的存储。于函数外定义，所有文件可见；关键词static定义对应范围。

Lifetime：程序结束

Dynamic storage 动态存储

可以用new \ delete 来提供更灵活的方法

lifetime：不是任意地与程序或函数绑定的，与new和delete也有关

vector数组就是动态分配的，array则是静态，元素数不能是变量。

以上为lec4

- Lec 5

Loop 循环 → 执行重复性任务！

- 设置初值
- 测试 loop 是否继续
- 运行 循环体
- 更新用值

- for (; ;)
- while ( )

用continue开始新周期

↑ Jump ↓

break很必要！

branch 跳转

- if( ) else ~~~

- switch (integer)
  { case ~: —
    case ~: —
    default: …
  }

逻辑运算符

- logical and &&
- logical or ||
- not !

优先级低于关系版，若||左边为真，不会运行||右边的表达式～

Lecb Function 函数

引入header

声明两部分：head与body；函数原型，调用函数。可以用库的 or 自己创。

有返回：typename funcname (参)
{ 语句 ; return ... ; }

无返：void funcname (...)
{ ... return ; (可有可无) }

function prototype 函数原型，在使用前声明。

是编译器的函数接口
处理返回值，检查参数量
检查参数类型并转换为正确的

int max(int, int); [有分号]
int *swap (int*, int);

函数名↑ 加分号～不用为var起名

```
#include <iostream>
using namespace std;

//Declaring a function
int sum(int x, int y);
int main()
{
    int a = 10;
    int b = 20;
    int c;

    //Calling a function
    c = sum(a,b);

    cout << a << " + " << b << " = " << c << endl;

    return 0;
}

// Defining a function
int sum(int x, int y)
{
    return (x + y);
}
```

声明在前

Declaring a function (function prototype)

Calling a function

Actual parameter ← 实参

定义在后

Defining a function Outside from all functions

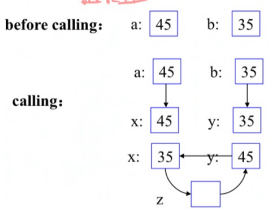
Formal parameter 形参

Output:

① 按值传递

1. Passing arguments to a function by value 值传递

```
passvalue.cpp > ...
#include <iostream>
using namespace std;

void swap(int x, int y)
{
    int z;
    z = x;
    x = y;
    y = z;
}

int main()
{
    int a = 45, b = 35;
    cout << "Before Swap\n";
    cout << "a = " << a << ",b = " << b << endl;

    swap(a,b);

    cout << "After Swap\n";
    cout << "a = " << a << ",b = " << b << endl;

    return 0;
}
```



Output:

```
Before Swap
a = 45,b = 35
After Swap
a = 45,b = 35
```

可以看到并不影响外面的实参

② 按指针传递

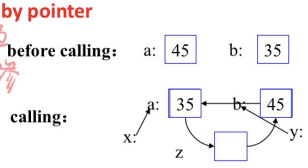
2. Passing arguments to a function by pointer

嗯...迫不得已的调用指针

```
passpointer.cpp > ...
#include <iostream>
using namespace std;

void swap(int *x, int *y)
{
    int z;
    z = *x;
    *x = *y;
    *y = z;
}

int main()
{
    int a = 45, b = 35;
    cout << "Before Swap\n";
    cout << "a = " << a << ",b = " << b << endl;

    swap(&a, &b);

    cout << "After Swap\n";
    cout << "a = " << a << ",b = " << b << endl;

    return 0;
}
```

指针传递 会改变实参

swap(&a, &b); ← a、b的值



Output:

```
Before Swap
a = 45,b = 35
After Swap
a = 35,b = 45
```

③ 传递一个数组

(array name as parameters and arguments)

```cpp
#include <iostream>
using namespace std;

void sum(int arr1[], int arr2[],int n);

int main()
{
    int a[5] = {10,20,30,40,50};
    int b[5] = {1,2,3,4,5};

    cout << "Before calling the function, the contents of a are:\n";
    for(int i = 0; i < 5; i++)
        cout << a[i] << "  ";

    sum(a,b,5);

    cout << "\nAfter calling the function, the contents of a are:\n";
    for(int i = 0; i < 5; i++)
        cout << a[i] << "  ";
    cout << endl;

    return 0;
}
void sum(int arr1[],int arr2[],int n)
{
    int temp;
    for(int i = 0; i < n; i++)
    {
        temp = arr1[i] + arr2[i];
        arr1[i] = temp;
    }
}
```

Using array as a parameter

arr1 = &a[0] or arr1 = a

将数组名视为指针

作用同以②，会改变传入的

```
Before calling the function, the contents of a a
10  20  30  40  50
After calling the function, the contents of a ar
11  22  33  44  55
```

The values of elements in array **a** are changed

4. Passing multidimensional array to a function

```cpp
#include <iostream>
using namespace std;

void square(const int arr[][3],int n);

int main()
{
    int a[2][3] = {
        {1,2,3},{4,5,6}
    };
    square(a,2);

    return 0;
}
void square(int arr[][3],int n)
{
    int temp;
    for(int i = 0; i < n; i++)
        for(int j = 0; j < 3; j++)
        {
            temp = arr[i][j];
            cout << temp * temp << " ";
        }
    cout << endl;
}
```

```cpp
void square(int (*p)[3],int n)
{
    int temp;
    for(int i = 0; i < n; i++)
        for(int j = 0; j < 3; j++)
        {
            temp = *(*(p + i) + j);
            cout << temp * temp << " ";
        }
    cout << endl;
}
```

the same as p[i][j]

```cpp
void square(const int (*p)[3],int n)
{
    int temp;
    for(int i = 0; i < n; i++)
        for(int j = 0; j < 3; j++)
        {
            temp = p[i][j];
            cout << temp * temp << " ";
        }
    cout << endl;
}
```

多维的

3. Passing arrays to a function

(protect the value of the argument from modifying, please use const pointer

```cpp
#include <iostream>
using namespace std;

void sum(const int *p1, const int *p2,int n);

int main()
{
    int a[5] = {10,20,30,40,50};
    int b[5] = {1,2,3,4,5};

    cout << "Before calling the function, the contents of a are:\n";
    for(int i = 0; i < 5; i++)
        cout << a[i] << "  ";

    sum(a,b,5);

    cout << "\nAfter calling the function, the contents of a are:\n";
    for(int i = 0; i < 5; i++)
        cout << a[i] << "  ";
    cout << endl;

    return 0;
}
void sum(const int *p1,const int *p2,int n)
{
    int temp;
    for(int i = 0; i < n; i++)
    {
        temp = *p1 + *p2;
        *p1 = temp;
        p1++;
        p2++;
    }
}
```

也可以传入指针类型

recommended to use the **pointer-to-const** form to protect data!!

这样就不能改啦

Using const pointer as a parameter

保护数值！

The value of array can not be modified.

p1 = a or p1 = &a[0]

改了会报错！

```
passarray3.cpp: In function 'void sum(const int*, const int*, int)':
passarray3.cpp:30:13: error: assignment of read-only location '* p1'
   30 |         *p1 = temp;
      |         ~~~~^~~~~~
```

5. Passing C-style string to a functio

passcstring.cpp > ...

```cpp
#include <iostream>
#include <cstring>
using namespace std;

void mcopy(char *s,int m);

int main()
{   void mcopy(char *s,int m);
    char str[81];
    int m;
    cout<<"Enter a string:\n";
    cin.getline(str,80);

    cout<<"Enter m:\n";
    cin>>m;

    mcopy(str,m);

    cout << str << endl;

    return 0;
}

void mcopy(char *s,int m)
{
    strcpy(s,s+m-1);
}
```

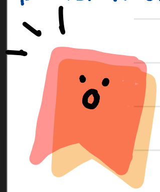
m=3 Hello m-1

前四个词去掉 就要m什么

④字符传递

You ca
as a p

char array 或 pointer-to-char



⑤ 传递结构

A. ——(struct student rec)
{ ... rec.name ;
}

main(){
——(rec);
}

B. ——(struct student *rec)
{ rec->name; }

main(){
——(&rec); }

Recursive function 递归

函数里还有它本身 ...

通常更慢，更占内存.

⑥指引用传递

plecode > swap.cpp > ...

```cpp
#include <iostream>
using namespace std;

void Swap(int &x, int &y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}

int main()
{
    int a = 45, b = 35;
    cout << "Before Swap" << endl;
    cout << "a = " << a << ",b = " << b << endl;

    Swap(a, b);

    cout << "After Swap(passing by reference)" << endl;
    cout << "a = " << a << ",b = " << b << endl;

    return 0;
}
```

Only by checking the function prototype or function definition can you tell whether the function passing by value or by reference. In the called function's body, the reference parameter actually refers to ***the original variable*** in the calling function, and the original variable can be ***modified*** directly by the called function.

The style of the arguments are like common variables

会改变

output:

```
Before Swap
a = 45,b = 35
After Swap(passing by reference)
a = 35,b = 45
```

函数指针
Pointers to Functions
函数也有地址.
可以用它调用函数.

```cpp
int findmax(int,int);
int(*funptr)(int,int);
funptr = findmax;
int max=funptr(3,5);
```

- lec7 函数调用也是像计组那样，将指令加载到内存再运行。

Inline function 内联函数

比常规快，附带存储 penalty，用它要有选择性.

关键字 inline，减少函数调用开销，增加运行时间，编译时替换定义.

Function 重载 overload

针对不同的传入参数而写不同的方法。

```cpp
#include <iostream>
using namespace std;

//overloaded functions...
void add(int i, int j);
void add(int i, double j);
void add(double i, int j);
void add(int i, int j, int k);

int main()
{
    int a = 1, b = 2, c = 3;
    double d = 1.1;

    // overloaded functions with difference type
    // and number of parameters
    add(a,b);    // 1 + 2 => add prints 3
    add(a,d);   // 1 + 1.1 => add prints 2.1
    add(d,a);
    add(a, b, c);  // 1+ 2 +3 => add prints 6

    return 0;
}
```

The same function name but different parameter list

```cpp
void add(int i, int j)
{
    cout << "Result: " << i + j << endl;
}
void add(int i, double j)
{
    cout << "Result: " << i + j << endl;
}
void add(double i, int j)
{
    cout << "Result: " << i + j << endl;
}
void add(int i, int j, int k)
{
    cout << "Result: " << i + j + k << endl;
}
```

Output:

```
Result: 3
Result: 2.1
Result: 2.1
Result: 6
```

函数模板 Function Templates

用于多种可以整体泛型一样的调用

compile internally generates and adds right code respectively.

```cpp
template <typename T>
T Max(T x, T y)
{
    return (x > y? x : y);
}

int main()
{
    cout << "Max int = " << Max<int>(3,7) << endl;
    cout << "Max char = " << Max<char>('g','e') << endl;
    cout << "Max double = " << Max<double>(3.1,7.9) << endl;

    return 0;
}
```

```cpp
int Max(int x, int y)
{
    return (x > y? x : y);
}
```

```cpp
char Max(char x, char y)
{
    return (x > y? x : y);
}
```

```cpp
double Max(double x, double y)
{
    return (x > y? x : y);
}
```

output:

```
Max int = 7
Max char = g
Max double = 7.9
```

命名空间 namespace 避免名称冲突

```cpp
namespace Jill {
    double bucket(double n)
    double fetch;
    struct Hill { ... };
}
char fetch;
int main()
{
    using Jill::fetch;   // put fetch into local namespace
    double fetch;        // Error! Already have a local fetch
    cin >> fetch;        // read a value into Jill::fetch
    cin >> ::fetch;      // read a value into global fetch
    ...
}
```

variable declared in namespace Jill

global variable

全向变量

Using declaration

运算符的重载

Lec12

1. 概念填空题

1.1 运算符重载是对已有的运算符赋予_多重_含义，使同一个运算符在作用于_不同类型_对象时导致不同的行为。运算符重载的实质是_函数重载_，是类的_多态性_特征。

1.2 可以定义一种特殊的类型转换函数，将类的对象转换成基本数据类型的数据。但是这种类型转换函数只能定义为一个类的_成员_函数而不能定义为类的友元函数。类类型转换函数既没有_参数_，也不显式给出_返回类型_。类类型函数中必须有_return 表达式_的语句返回函数值。一个类可以定义_多个_类类型转换函数。

1.3 运算符重载时其函数名由_operator 运算符_构成。成员函数重载双目运算符时，左操作数是_对象_，右操作数是_函数参数_。

3.1在下列运算符中,不能重载的是(B)
A.！ B. sizeof C. new D. delete

3.2 不能用友员函数重载的是(A)。
A.= B.== C.<= D.++

3.3下列函数中,不能重载运算符的函数是(B)。
A.成员函数 B.构造函数 C.普通函数 D.友员函数

3.4如果表达式++i*k时中的”++”和”*”都是重载的友元运算符，则采用运算符函数调用格式，该表达式还可表示为（B)
A. operator*(i.operator++(),k)
B. operator*(operator++(i),k)
C. i.operator++().operator*(k)
D. k.operator*(operator++(i))

3.5已知在一个类体中包含如下函数原型：VOLUME operator-(VOLUME)const；下列关于这个函数的叙述中，错误的是（B ）
A.这是运算符-的重载运算符函数
B.这个函数所重载的运算符是一个一元运算符
C. 这是一个成员函数
D. 这个函数不改变数据成员的值

3.6在表达式x+y*z中，+是作为成员函数重载的运算符，*是作为非成员函数重载的运算符。下列叙述中正确的是（C ）。
A. operator+有两个参数，operator*有两个参数
B. operator+有两个参数，operator*有一个参数
C. operator+有一个参数，operator*有两个参数
D. operator+有一个参数，operator*有一个参数

Lec13

Deep Copy

显式的构造器

```
String::String(const String& str)
{
    m_data = new char[strlen(str.m_data) + 1];
    strcpy(m_data, str.m_data);
}
```

定义复制构造函数之所以必要，是因为某些类成员是指向数据的新初始化指针，而不是数据本身。

Shallow Copy

每个值都拷贝一下

```
Complex::Complex(const Complex& c) {
    real = c.real;
    imag = c.imag;
    std::cout << "Copy Constructor called" << std::endl;
}
```

只是整个值

Lec14

Inheritance

继承

class A → Base

↑

class B → Derived

! Single Inheritance

也可以多层，

也能多个爹。

懒得写了，就这样吧。

Quiz总结

Week1

Q1. The source code of program is as follows:

```
int main(int argc, char *argv[])
{ ...}
```

The source code has been compiled to program hello, when you run it as follows, what argc will be?

```
./hello I LOVE CPP
```

Answer: 4 (四个单词)

Q2. When a function prototype is declared in the header file you create, you do NOT need to define it in a CPP file.

A. True. B. False

需要定义此函数 in.cpp

Q3. Preprocessor is the FIRST step of compilation.

A. True. B. False

第2步 → edit 编辑 / Preprocess 预处理 / Compile 编译 / Link 连接 / Load 加载 / execute 运行

不能！就是预处理

Q4. Macro is a kind of special function, and it can be compiled similar with a function.

A. True. B. False

Q5. What statement would you use to print the phrase "Hello, world" and then start anew line?

Answer: cout<< "Hello world "<< endl;

Week2

long 为 $2^{31}-1$, long long $2^{63}-1$

Q1. What's the output of the source code?

```
short sam = 32766 ; sam += + 2 ;
cout << sam;
```

Answer: -32768

short 长度为 $2^{15}-1$ → 32767 → 32767 溢出了

Q 2. What's the output of the source code?

```
float a = 2.34E+15f ; float b = a + 100.0f ;
cout << b - a << endl;
```

Answer: 0

Q3. What's the output of the source code?

```
signed char c1= 127 ;
signed char c2= 1 ;
int csum = c1 + c2;
cout << csum ;
```

Answer: 128

Q4. What's the output of the source code?

```
int a = 124; float b = 3.0; cout << a + b ;
```

Answer: 127

Q5. What's the output of the source code?

```
int a = 2; char b = '3'; cout << a + (b – '0') ;
```

Answer: 5

Week3

Q1. What's the output of the source code?

```
int a=3610; h=a/3600, m=(a % 3600)/60;
pintf("%d:%02d\n", h, m)
```

h=1 m=10/60=0

Answer: 1:00 (两位)

Q 2. What's the output of the source code?

```
float a = 1.1, b=2.2, c=3.3; int e;
e = (int)(a+b+c+0.5); cout << e << endl;
```

6.6+0.5=7.1 → 7

Answer: 7

Q3. What's the output of the source code?

```
int a, b, c, d, e, f; a = 2; b = 3;
c = a+b; d = a - b; e = a * b;
f = c + d + e;
cout << f << endl ;
```

5, -1, 6, f=10

Answer: 10

Q4. What's the output of the source code?

```
int a = 10000, b=3; float c=0.035;
sum = a *(1 + c ); printf("%.2f", sum);
```

10350

Answer: 10350.00

9/3

Q5. What's the output of the source code?

```
int a=2, b=7; float c=(a+b)/3 ; cout << c << endl;
```

Answer: 3

Week 4

Q1. The incorrect string constant is
A. 'abc' B. "12'12"
C. "0" D. " "

Answer:

Q 2. What's the output of the source code?
```
char a[]="lavender"; char b[100]="lavender";
printf("%d ",sizeof(a)); printf("%d ",sizeof(b));
```
A. 8 100 B. 9 100

Answer:

Q3. What's the output of the source code?
```
struct s
{
 int a; char b; float c;
}
printf("%d", sizeof(struct s))
```
Answer: 12

Q4. The string class directly supports size comparison of different strings?
A. True B. False

Answer:

Q5. Which one has a syntax error?
A. char a[]="x+y=55."; B. char a[10]='5';

Answer:

Week 5

Q1. int a[][4]={1,2,3,4,5,6,7,8,9,10}; int *p=*(a+1); Which expression can get the value 9?
A. p+=3,*p++; B. p+=4,*(p++);
C. p+=4,*++p; D. p+=4,++*p;

Answer:

Q 2. There are some declarations, which one can complete the assignment function of i=j?
```
int i, j=2, *p=&i;
```
A. *p=*&j; B. i=&j;

Answer:

Q3. What's the output of the source code?
```
char *p = "abcdefgh", *r;
long *q = (long *)p;
q++;
r = (char *)q;
cout << r << endl;
```
Answer: efgh

Q4. Define two variables: int a[5], *p; which description is wrong ?
A. p=p+1 B. a=a+1

Answer:

Q5. char *name="newspaper"; The output of the cout<<name+2; is ( )

Answer: wspaper

Week 6

Q1. What's the output of the source code?
```
main() {int a=1, b=3, c=5;
int *p1=&a, *p2=&b, *p=&c;
*p = *p1*(*p2); printf("%d\n", c);}
```
Answer: 3

Q 2. What's the output of the source code?
```
#include<stdio.h>
main() {int **k, *a, b=100;
    a=&b; k=&a; printf("%d\n", **k);}
```
Answer: 100

Q3. What's the output of the source code?
```
char a[]="programming", b[]="language";
char *p1, *p2; int i; p1=a; p2=b;
for(i=0; i<7; i++)
  if (*(p1+i)==*(p2+i))
    printf("%c", *(p1+i));
```
Answer: ga

Q4. What's the output of the source code?
```
int i, j, k; for(i=0,j=10; i<=j; i++,j--) k=i+j;
cout << k << endl;
```
Answer: 10

Q5. int *p, m=5, n; which is the correct code ?
A. p=&n; *p=m; B. p=&n; scanf("%d", *p);

Answer:

Week7

Q1. What's the output of the source code?

```
void fun(int a, int b, int c) {a=456; b=567; c=678;}
main() {int x=10, y=20, z=30;
    fun(x,y,z); printf("%d, %d, %d\n", z, y, x);}
```

Answer: 30, 20, 10 按值不影响~

Q3. What's the output of the source code?

```
# include "stdio.h"
main() {int a=24, b=16, c; c=abc(a, b); printf("%d\n", c);}
int abc() {int u, int v} { int w;
  while(v) {w=u%v; u=v; v=w;} return u;}
```

A. 6 B. 8

Answer:

w=24%16=8
u=16 v=8
w=16%8=0 → false
u=8, v=0 return 8

Q 2. In the function prototype description, which item is not necessary?

A. The parameter name of the function
B. Type of function

Answer: 必要: header + type

Q4. Reference as a function parameter can operate on directly referenced variables in the function.

A. True B. False

Answer: 可以改! (按值才不改

第一个!

Q5. What's the output of the source code?

```
void main(){char str[10]={'a','b','c'}; cout<<*str;}
```

Answer: a

Week8

Q1. What's the output of the source code?

```
void sum (int num=10){int i, s=0;
  for(i=1; i<=num; i++) s=s+i; cout << s << endl;}
int main() { sum(100); sum();}
```

Answer: 5050

Q3. What's the output of the source code?

```
void Max(int a) { cout << "Max 1" << endl; }
void Max(int a, int b) { cout << "Max 2" << endl; }
void Max(int a, int b, int c) { cout << "Max 3" << endl; }
int main() { Max(1,2,3); return 0;}
```

A. Max 2 B. Max 3

Answer:

Q 2. What's the output of the source code?

```
int main () { int a[4]={2,4,6}; int y=0, *p=a;
for(; *p++;) if(*p) y+=*p; cout << y<< endl; }
```

A. 10 B. 11 a+1 y+4+6

Answer:

Q4. In inline functions, you can include switch statements and loop bodies.

A. True B. False ☆ 不可以!!

Answer:

第一次见耶!! ☺☺

Q5. What's the output of the source code? 记住它.

```
int main(){ int a; int &b=a; b=10; cout << a << endl;}
```

Answer: 10 a为b的值?!

Week10

Q1. What's the output of the source code?

```
namespace { void func() { cout<<"a=10"<<endl; } }
int main() { func(); return 0; }
```

A. compile error B. a=10

Answer: 运行3!

Q3. What's the output of the source code?

```
namespace A { int a=100; void func(); }
void A::func() { cout<<"a = "<<a<<endl; }
void funb() { int a = 200; cout<<"a = "<<A::a<<endl; }
int main() { funb(); return 0; }
```

A. a=100 B. a=200;

Answer:

Q 2. The automatic storage duration is created when the program enters the block in which it is defined, and destroyed when it leaves the block.

A. True B. False 函数的未 Auto

Answer:

Q4. The static duration is allocated from the beginning of the program execution to when the program is completed.

A. True B. False 程序的未 static

Answer:

都没有~

Q5. Dynamic allocation will be destroyed immediately after execution. A. True B. False

Answer:

Week11

Q1. What's the output of the source code?

```
class Pair{ int X, Y;   public: Pair (int initX, int initY) :
  X(initX), Y(initY){} int sumXY() {return X+Y;}};
int main() { Pair A1(5,3);  cout << A1.sumXY(); return 0;}
```

Answer: 8 对象

Q3. What's the output of the source code?

```
#include <iostream>
using namespace std;
class info { public: info(int age) { this->age = age;
      cout << age << endl;} private: int age; };
int main(void) { info a(8); return 0; }
```

Answer: 8

w=24%16=8
u=16; v=8

Q 2. What's the output of the source code? w=0; u=8; v=0; false, u=8

```
int abc(int u, int v) {int w; while(v){w=u%v; u=v;v=w;}
  return u;} int main(){int a=24, b=16, c;
c=abc(a,b); printf("%d\n", c);}
```

Answer: 8 取余

Q4. When defining a destructor, it has no formal parameters and can not be overloaded.

A. True B. False

Answer: 析构器 不被重载

Q5. this pointer exists in every function.

A. True B. False

this在类里~

Week12

Q1. In a class contains the following function prototypes, "VOLUME operator- (VOLUME) const", which one is wrong?

A. This is an overloaded operator function of the operator "-"

B. This function is overloaded with a unary operator

Answer: B 这个是二元运算符,不是一元的

Q3. What's the output of the source code?

```
class Date{private: int month=5; int day=6;
  public: Date operator++(int); void display();};
Date Date::operator++(int){day++; return *this;}
void Date::display(){cout<<month<<"-"<<day<<endl; }
int main(){ Date d1; d1++; d1.display();}
```

Answer: 5-7

类 后置++

Q 2. The "++" and "*" in the expression "++ i * k" are both overloaded friend operators. If the operator function call format is used, the expression can also be expressed as

A. operator*(i.operator++(),k) B. operator*(operator++(i), k)

Answer: B 5(b<)> 记住它

Q4. Which of the following functions cannot overload the operator.

A. friend function B. constructor

Answer: constructor to destructor 构造的最不能重载操作符

Q5. Which one cannot be used for friend function overloading.

A. = B. ==

Answer: 赋值这是

Week13

Q1. If a class has pointer variables and dynamic memory allocation, it must have a copy constructor. What form does this constructor take?

A. deep copy B. shallow copy

Answer:

Q3. What's the output of the source code?

```
class Foo { public: Foo(); Foo( int );  private: int val; };
Foo::Foo() { cout << "0"<< endl; }
Foo::Foo(int i) { cout << "1"<< endl; }
class Bar { public:  Foo foo;  char *str;  };
void foobar(){ Bar bar; } int main() { foobar(); return 0; }
```

Answer: 0

没法拷值来复制!

Q 2. What constructor is called by the following statement?

```
class A; A a; A b; b = a ;
```

A. copy constructor

B. assignment operators ✓

Answer:

构造. 若 A b=a; 是copy const

Q4. The constructor can be overloaded and have parameters.

A. True B. False

Answer:

可以带参! ✓ ~() ~(int) ~(int int)

Q5. You can call default constructor without any parameters.

A. True B. False

Answer: ✓

Week 14

emm...

Q1. When the object is released, what is the execution order of the destructor in the compilation system?

A. derived class, sub-object class in derived class, base class

B. sub-object class in derived class, derived class, base class

Answer:

Q 2. When creating a derived class object, what is the execution order of the constructor?

A. base class, object member, and derived class constructor

B. base class, derived class, and object member constructor

Answer:

Q3. What's the output of the source code?

```
class base{public: int a, b;void get(int i, int j) {a = i; b=j;};};
class A: public base{ public: void get(int i,int j){base obj3;
obj3.get(5,6); u=i+j+obj3.a;}void print(){ cout << u;} private:
int u; }; int main()
{base ob1; A ob2; ob1.get(1,2);ob2.get(3,4);ob2.print();}
```

Answer:

Q4. Derived classes can define data and functions that are not available in the base class.

A. True B. False

Answer:

Q5. The inheritance relationship between classes is symmetrical.

A. True B. False

Week 15

Q1. What kind of inheritance method can the derived class adopt to make the member variable become its own private member?

A. Private member B. Protected member

Answer:

Q 2. When multiple inheritance derived class constructors construct an object, which one is called first?

A. Constructor of virtual base class

B. The constructor of the sub-object class in the derived class

Answer:

Q3. What's the output of the source code?

```
class Base {int i; public: Base(int n) {i =n;} ~Base(){}
void showi(); int Geti() {return i;} };
class Derived:private Base { int j; Base aa; public:
Derived(int n, int m, int p):Base(m), aa(p){j=n;}
~Derived(){} void show() { cout << j << "," << aa.Geti() <<
endl;} };int main() { Derived obj(8,13,24); obj.show(); }
```

Answer:

Q4. A derived class can be used as the base class of another derived class.

A. True B. False

Answer:

Q5. Which one can be inherited by derived classes?

A. Constructor B. Virtual function