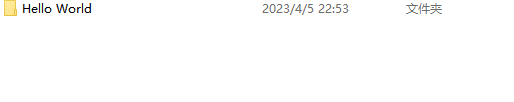
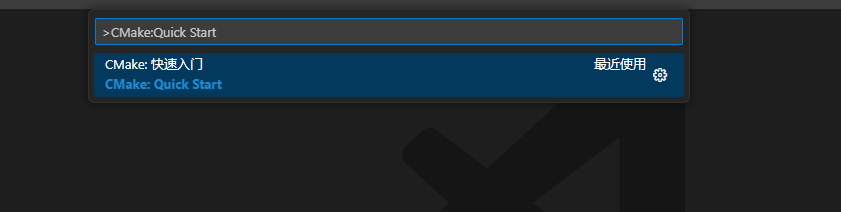
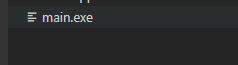
## 第一课总结

## **使用VSCode开发第一个C++项目**

1. 新建一个文件夹管理多个项目
2. 
3. C++中放置一个文件夹管理一个单独的项目
4. 
5. 用VSCode打开此文件夹，此时文件夹为空。
6. 
7. 按下Ctrl+Shift+P打开命令面板，输入并点击CMake:Quick Start，（选择gcc...），输入项目名回车后，点击Executable，会生成CMakeLists.txt和main.cpp。
8. 
9. 然后可以进行编译，由于编译环境之前搭好了，不在赘述。
10. 在终端打印出Hello World并且生成科执行文件Hello World.exe
11. 
12. 问题和答案（对C++的理解）：
13. 问题：我们能忽略警告吗？
14. 答：编译器有时候会发出警告信息，警告和错误时有区别的，警告的这一行在语法上是正确的并且有编译的价值。然而，我们对于警告的这一行代码有许多更好的代码来代替这一行的代码，并且一个好的编译器会以一个代码的修复建议来发出警告
15. 问题：解释语言和编译语言的区别？
16. 像JavaScript, Ruby, and Python 这样的语言就是解释语言。语言使用解释器读取(解释)脚本文件中的代码并执行所需的操作。编译语言像C++这样的语言使用Build（链接）这个步骤来生成可执行文件用来知道处理器执行代码。
17. 问题：什么是运行时错误，编译时错误和运行时错误有什么不同。
18. 答：当你在执行一个应用程序时所发生的错误叫运行时错误。在Windows系统下，你也许经历过臭名昭著的非法访问，这是由一个有bug的可执行文件所引起的运行时错误。编译时错误发生在Build(链接过程中)，他们表示语法错误，需要程序员在编译之前完成。

# 第二课总结

## 预处理指令#include:

1. 像这样的指令表示处理器运行代码是在编译开始之前，预处理指令总是要求预处理器以#开始（也呗称为哈希符号），#include<Filename>告诉预处理器去连接对应的文件，iostream是一个标准头文件，包含我们后能够使用里面所包含的代码，像std::cout这些代码，因为头文件iostream里面对std::cout有定义。
2. 许多应用在编译时有各种各样的文件，一个文件需要包含其他文件，如果一个函数或者变量被声明在A文件时，而B文件需要使用时，你需要在B文件中包含A文件，”#include...A对B的相对路径路径\FileA”。
3. 当我们使用我们自己编写的头文件时用””英文引号把头文件包起来，当我们使用标 准头文件时使用尖括号（<>）把头文件包起来，像<iostream>,<string>和<algorithm>这样 的标准头文件。

## 程序框架：main()

1. 预处理指令后面跟着的是程序的框架由函数mian()来表示，C++代码的执行总是从这里开始，标准的用法是在main()函数之前用一个int来声明main()函数（既int main()），int是main()函数的返回值的类型是整型的意思。

## 返回值：

1. 在c++中一个函数需要返回一个指，除非有特别的声明。main()是一个函数总是返回一个整型值，这个值是被返回给操作系统，该返回值取决于你的应用的性质，这种返回方式特别有用，在许多操作系统通过查询返回值来判断程序的正常结束。在许多情况中，一个应用被引用到另一个应用中，这个父应用（应用别的应用的应用）想知道子应用（被引用的那个应用）是否成功的完成任务，程序员可以使用main()的返回值来传达父应用的成功还是失败。
2. 程序员一般用返回值0表示事件的成功，用-1表示事件的失败，返回值是一个整型，程序员可以灵活的通过整型的取值范围取不同的值来表示各种成功和失败的状态。
3. C++是非常敏感的，我们必须区分大小写。

## 命名空间：

1. 当我们在程序中使用std::cout时不仅仅使用的是cout这个对象（这中对象被称为artifacts），而是标准命名空间（std），对一些代码进行了标准名称定义，using namespace声明，能够帮助我们避免重复使用，

1: // Preprocessor directive

2: #include <iostream>

3:

4: // Start of your program

5: int main()

6: {

7: // Tell the compiler what namespace to search in

8: using namespace std;

9:

10: /\* Write to the screen using std::cout \*/

11: cout << “Hello World” << endl;

12:

13: // Return a value to the OS

14: return 0;

15: }

1. 上面的代码中使用了 using namespace 对std进行声明，我们在调用cout时不需要在重复写std这个命名空间。

1: // Preprocessor directive

2: #include <iostream>

3:

4: // Start of your program

5: int main()

6: {

7: using std::cout;

8: using std::endl;

9:

10: /\* Write to the screen using std::cout \*/

11: cout << “Hello World” << endl;

12:

13: // Return a value to the OS

14: return 0;

15: }

1. 我们还可以使用using std::cout和using std::endl来代替上面的using namespace std，这种不包括完整的命名空间的使用方法更加严格，你只需要特别声明你想使用的那个对象（artifacs）就行，但是最后我们没有对命名空间进行消除，这仅仅局限于std::cout。

## C++中的函数

1. 我们有时候需要把我们的代码划分成多个函数构成的逻辑单元，我们如果把我们的子函数定义在调用函数（父函数）之后那么我们需要在调用之前（父函数之前）把函数的头声明一下（例如：int Function()），如果我们在调用之前已经定义了子函数，那么我们不需要再对该函数进行定义。

## 输入输出std::cin和std::cout:

1. cin可以实现人机交互列如：

#include <iostream>

#include <string>

int InputNum;

using namespace std;

int main() {

    cout<<"Please Input Your InputNum ";

    cin>>InputNum;

    string Name;

    cout<<"Please Input Your Name ";

    cin>>Name;

    cout<<Name<<" ENter Num "<<InputNum<<endl;

    return 0;

}

1. Exercise:
2. Answer:

#include <iostream>

int main() {

    std::cout << "Is there a bug here?";

    return 0;

}

1. Analysis:返回类型必须是int类型，左边缺少双引号std::Cout中的C应该变为小写。

# 第三课总结

## 内存和地址简介：

1. 一台电脑，或者智能手机，或者其他可编程的设备包含一个微处理器和一定量的内存用作临时存储，被称为随机存取存储器（RAM）。另外许多设备还允许数据存储再存储设备上，例如硬盘。微处理器实现的命令是由你写的应用构成的。在这总情况下，微处理器使用RAM找到二进制代码和相关的程序去执行。
2. RAM本身就可以想像成存储区，就像一个宿舍中的一排储物柜，每个柜子有一个号码，一个号码就是一个地址，为了找到内存中的一个位置，就像位置587，这个处理器需要被编程使用一个命令来拿到该位置中的值，或者向内存中的这个位置写入一个值。

## sizeof()的使用：

1. sizeof(变量类型或者变量名)函数可以把变量所占的字节数返回给程序员。

## 自动类型的使用auto：

使用auto来定义变量时，我们的编译器会核查该变量的初始值来决定该变量最可能的类型，然后就会把该变量自动定义为该类型。

#include <iostream>

using namespace std;

int main() {

    int variableA=0;

    int SizeOfVariableA = sizeof(variableA);

    auto VariableB=true;

    int SizeOfVariableB = sizeof(VariableB);

    cout<<" SizeOfVariableA= "<<SizeOfVariableA<<'\n';

    cout<<" SizeOfVariableB = "<<SizeOfVariableB<<endl;

}

## 使用typedf来定义变量：

1. typedf是变量的类型定义，比如：typdef usinged int UINT;这时候UINT就表示usigned int。我们可以用这个直接定义一个无符号整型变量，UINT VariableA=0;这就相当于usinged int Variable=0;既增强了代码的可读性又使得代码简单整洁方便。

## 常量：

1. 常量就是不可该变数值的变量，用const作为常量关键字，定义方式为：const 常量类型 常量名=初值；这就是const的定义，注意由于常量是不可改变的量，必须在定义是一定要给值。
2. 常量表达式，使用关键字constexpr来定义常量表达式，像函数就可以定义为一个常量表达式函数表达式为：constexpr 返回值类型 函数名(){函数内容}，返回值为一个常量，当常量函数的参数是常量时，常量函数才会被编译器认为是一个常量表达式，当参数是一个变量时，编译器会把这个函数认为是普通函数在运行时被执行。

## 即时函数：

1. C++20定义即时函数用constval，即时函数定义表达式：constval 返回值类型 函数名(){内容}，即时函数只能使用常量参数来调用，如果用变量作为参数调用该函数就会出错。

## 枚举类型：

1. 枚举类型定义变量用enum，定义式：enum 枚举类型变量名 {常量1；常量2...}，枚举类型的元素为多个常量，这样就可以定义大量的常量，不用分开对每个常量单独定义，如果我们不对第一个常量赋值时就会默认第一个常量的值为0，我们也可以对每一个元素都进行赋值，当我们只对第一个常量赋值时那么我们的其他元素按照位置分别被赋值为上一个元素的值加1，我们可以直接对枚举中的每一个元素单独使用，也可以使用枚举变量来使用表达式为：枚举类型名 变量名=枚举中的某一个元素名。
2. 枚举类型还可以枚举类(class)和结构体(struct)，在定义变量时我们可以进行转换，既可以用变量类型来接枚举变量表达式为：变量类型（可以是非枚举变量） 变量名=枚举中的常量，但是如果是类或者是结构体时，我们不能进行这样的转换，只能用对应的变量来接我们的类中的对象，或者结构体中的元素，表达式为：类名 变量名=类名::对象名。

## 使用#define来定义常量：

1. #define是预编译宏定义，表达式为#define A B，在编译之前通过预处理器把A代替为B这只是简单的文本替换，编译器即不知道也不在乎这个常量的实际类型。

## 第四课总结

## 什么是数组：

1. 一个数组是一堆元素的集合，所有的元素包含在一个相同类型的素组中，这个集合完形成了一个完整的设置。

## 静态数组的声明和和初始化

1. 数组定义式：数组元素类型 数组名[静态元素量]={元素的初始值1，元素的初始值2...}；
2. 数组的初始化：我们可以对数组的每一个元素进行初始化，例如int MyNumArry[5]={1,2,3,4,5};我们也可以不对数组中的元素进行初始化例如：MyNumArry[5]={};这样我们的数组中的每个元素都是0，我们也可以对数组进行部分初始化例如：MyNumArry[5]={1,2};这样就初始化了我们的数组中的第一个和第二个元素，元素值分别为1和2，如果我们知道每个元素的值时，我们还可以不对元素的个数进行说明例如：MyNumArry[]={1,2,3};那么该数组的元素个数为3个，注意数组的第一个元素的坐标为0。

## 数组的元素怎么存储的：

1. 我们之前知道每个类型的变量所占的空间，数组所占空间等于数组长度乘以单个变量所占的空间，例如：int MyNumArry[5]={...};那么MyNumArry所占空间等于5\*4Byte=20Byte(设单个int类型变量所占空间为4个字节)。

## 多维数组：

1. 多维数组如同多个一维数组一样，我们以二维数组为例，二维数组定义：

int MyNumArry[2][3]={{1,2,3},{4,5,6}};数组总的元素个数为行长度\*列长度，数组初始化时是先行后列。

1. 注意：当我们定义多维数组时，内存中对数据的存储仍然是以一维的方式存储的，只不过的编译器的映射，在映射时数据仅仅在一个方向扩展，我们还可以对数组进行这样的初始化int MyNumArry[2][3]={1,2,3,4,5,6};不过之前的那种方法更容易让人理解多维数组。

## 多维数组的访问：

以二维数组为例：访问某一个元素只需要写出数组名和坐标就行：MyNumArry[0][1]那么输出值为2。代码如下：

#include <iostream>

using namespace std;

int main() {

   int MyNumArry[2][3] ={1,2,3,4,5,6};

   cout<<MyNumArry[0][1]<<endl;

}

## 动态数组：

1. 当我们需要实时改变数组的长度时我们如果使用静态数组我们就得预留一定得空间，而且该空间得足够大，是程序员根据实际情况评估，但是如果人的评估总会出错，当我们预留空间太小时，我们会出现不够用，那么我们得系统就会崩溃，給用户带来不小损失，如果空间太大那么造成内存资源的浪费，所以我们需要一个可以实时根据用户需求改变数组长度的数组，所以C++引入了动态数组，动态数组和静态数组在功能上最大的区别就是数组长度可以根据需求随时改变，动态数组由std::vector这个对象提供。
2. 动态数组定义和初始化以整型为例，定义式：vector<int> DynArry(3)，这样就定义了一个一维动态数组，初始长度为3。DynArry.size()可以计算数组长度，DynArry.push\_back，push\_back这个函数可以把我们要添加的数据存放到动态数组的最后，且数组长度加一，这相当于入栈，出栈这里没有讲，后面应该会讲更多，动态数组需要包含头文件<vector>

## 字符串：

1. 在C语言中字符串中’\0’表示结束字符，当我们在数组中初始化字符串时最后一个字符为’\0’，所以在使用字符串数组时数组长度至少要大于实际字符串一个字符，我们也可以自己使用’\0’来表示字符串的结束。
2. C++字符串的使用std::string，字符串声明方式：string greetString=(“Hello”);这个可以完成字符串的定义和初始化，当然也可以不用初始化，直接定义string Firstline;就可以;可以用getline(cin,Firstline);让用户输入字符串，我们还可以对字符串进行拼接getline(cin,Secondline);

string ConcatString =Firstline + ” “ + Secondline;这样就完成了字符串的拼接string acopy = ConcatString可以2完成字符串的copy。

# 第五课总结

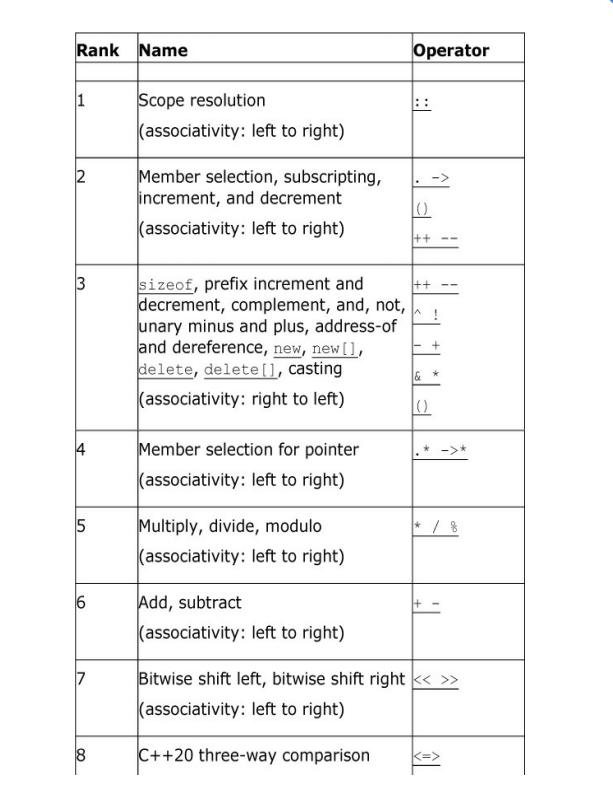
1. 当我们需要字符串换行时我们得加一个\来换行，否则显示错误，或者输出两个字符串例如:cout<<”Hello”

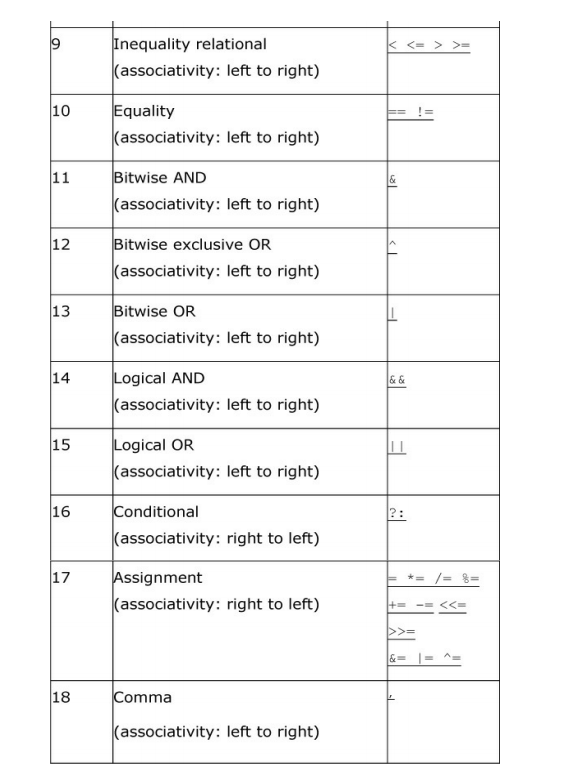
“world”<<endl;

## 三种方式比较运算符：

1. Auto result = var1 <=> var2 ，如果vresult小于0则var1小于var2，如果result大于0则var1大于var2，如果等于0则var1等于var2，这种运算符得优先级比别的运算符都高。
2. 异或：就是模二加（^）这个符号。
3. 移位运算符，不是循环移位，在移位中左移十进制算法为左移n位那么就是就是十进制数乘以2^n，如果时右移n位则十进制数就除以2^n。

## 运算符优先级：





# 第六课总结

## 判断语句：

1. if和else if的区别：if和if之间是相互独立的，互不影响，而if和else if是互斥的当执行if时就不会执行else if执行else if后就证明if没有被执行，也有可能两个都不执行，当if没有花括号时就相当于if中只有离它最近的那一条代码。
2. switch语句当有匹配项时就执行匹配项命令，然后break，如果我们在执行一个匹配项后没有break那么我们的程序会忽略剩下的case中的条件按顺序执行每一个语句，当程序遇到break时就跳出整个switch程序，我们一般以防程序出错会添加default调剂，当没有对应的case时我们的switch会执行default中的语句，防止程序出错而没有任何提示和反应。
3. goto代码，goto代码和起始标签配合使用例如：

#include <iostream>

#include <string>

using namespace std;

int main() {

    char repeat='y';

    Stat1:

        cout<<"Hello World! \n";

        cout<<"repeat again? \n";

        cout<<"Enter char y/n: \n";

        cin>>repeat;

    if(repeat=='y'){

         goto Stat1;

    }

    else{cout<<"GoodBye!";}

}

输出：

Hello World!

repeat again?

Enter char y/n:

y

Hello World!

repeat again?

Enter char y/n:

n

GoodBye!

## 循环语句：

1. for循环第一个是初始化命令只在第一次循环时执行，第二个条件在每次循环开始时都要判断，第三个表达式在每次循环结束后执行。当第二个条件不满足时就不再循环，循环结束。

## break和continue

1. break只跳出当前多分支语句和循环体，不能直接和if使用，if嵌套在循环中时才可以使用break，这时跳出得是循环不是跳出if继续循环，当多个循环嵌套使用时只跳出当前循环，外层循环仍然继续，循环和switch嵌套使用时switch中的break也只跳出switch。
2. continue是忽略continue以下的语句结束本次循环直接开始下一次循环例如：

#include <iostream>

using namespace std;

int main() {

    for(int i=0;i<5;i++){

        if(true){

            cout<<"Hello World \n";

            continue;

            cout<<"Continue Exercise \n";

        }

    }

}

输出结果：

Hello World

Hello World

Hello World

Hello World

Hello World

同样continue不能直接和if使用，它是用来处理循环的，当if嵌套在循环中时才可以使用。

1. break和continue一般都是在循环和if嵌套中使用。

# 第七课总结

## 函数：

1. 当函数遇到return时函数就结束了，如果函数为void类型时则可以直接return;。
2. 函数超载机制：不同函数使用同一个名字，但是参数不同或者参数的设置不同，称为超载，例如：

#include <iostream>

#define Pi 3.1415926

using namespace std;

double Area(double Radius);

double Area(double Radius,double Height);

int main() {

    double Radius=0;

    char UserSelection = 'z';

    cout<<"Enter z For Cylinder,c For Circle: \n";

    cin>>UserSelection;

    cout<<"Please Enter Radius: \n";

    cin>>Radius;

    if(UserSelection == 'z')

    {

        double Height = 0;

        cout<<"Please Enter Height For Cylinder: \n";

        cin>>Height;

        cout<<"The SurfaceArea Of Cylinder is : \n";

        cout<<Area(Radius,Height);

    }

    else

    {

        cout<<"The Area Of Circle is : \n";

        cout<<Area(Radius);

    }

    return 0;

}

double Area(double Radius){

    return Pi\*Radius\*Radius;

}

double Area(double Radius,double Height){

    return 2\*Pi\*Radius\*Height+2\*Pi\*Radius\*Radius;

}

1. 函数可以通过取地址返回数据给我们，比如我们参数中有一个参数取的是地址&，这个符号表示取地址，比如double& result表示取result的地址那么我们在传参数时我们的函数会对我们的变量取地址(传递的时候像穿普通参数就行，不需要我们再去取地址)，然后把变量的值存在对应的地址框中，例如：

#include <iostream>

#define Pi 3.1415926

using namespace std;

int Area(double Radius,double& Result);

int main() {

    double Result=0;

    double Radius=0;

    cout<<"Please Enter Radius For Circle: \n";

    cin>>Radius;

    Area(Radius,Result);

    cout<<"The Circle Area is: \n";

    cout<<Result;

}

int Area(double Radius,double& Result){

    Result = Pi\*Radius\*Radius;

    return 0;

}

# 第八课总结

## 指针：

1. 指针在定义时要么给个初始值，要么指向NULL，否则指针会随机指向一个地址，非常危险。在定义指针时且赋给指针变量地址时给的是变量的首地址，在定义指针时我们的指针所占的内存空间不是根据类型定的，而是系统固定的，32bit系统的所有指针都占用内存是4byte，64bit的系统给的是8Byte。当我们对指针的使用时应该注意：当我们访问变量采用偏执加指针首地址访问数组中的变量时我们的指针还是指向变量的，当我们采用指针自加加访问变量时我们的指针地址会变成我们访问的最后一个变量的地址，这时候我们如果要想指针返回到便利最初的首地址时应该采取自减减回到指针最初的位置。
2. new和delete的使用，和普通变量以及动态数组的区别，第一区别就是：new申请的这个空间和普通变量申请的空间的生命周期不一样，new所申请的空间不管在哪里申请的如果没delete掉那么我们申请的这个空间相当于全局变量，直到整个程序结束才释放这段内存空间，而我们的普通变量申请的空间声明周期只在一个块内，出了块就释放掉（猜想：我们vector提供的动态数组就和普通变量生命周期一样，但是在同样的生命周期内功能差不多，我们也可以自己写一个动态数组的访问，C++之所以比C方便是因为把大量我们需要自己写的代码进行集成封装了）。第二个区别是：我们的变量可以通过变量名访问该变量的值，通过&取地址符号读取地址，但是我们的new产生的变量空间没有普通变量名，只能通过指针访问值和地址。当我们使用完该变量时必须立即释放该变量，避免资源浪费。第三个是静态和动态的区别：我们可以申请动态数组来有效的利用内存资源，之前讲过动态数组，不在赘述。

new申请空间形势：Type\* PointerName = new Type，当申请空间失败时会返回一个例外，我们这时候要和catch配合使用;

申请块时：Type\* PointerName = new Type[ElementsNum];

delete的使用：delete PointerName;，释放块时delete[] PointerName;我们释放得时候释放的也应该是指针当初的地址，在使用块空间时应该注意返回到最初位置后才使用

delete[] PointerName，例如：

#include <iostream>

using namespace std;

int main() {

    int NumEntris = 0;

    cout<<"How Much Integrets You Wish To Store: \n";

    cin>>NumEntris;

    int\* PointedIntegersPointer = new int [NumEntris];

    //Output The Initial PointedIntegersPointer Of Adress.

    cout<<"The Initial PointedIntegersPointer Adress: \n";

    cout<<hex<<PointedIntegersPointer<<'\n';

    /\*The Pointer Still Stay Here The Initial Position Of Arry,

    Because We Adapted The Pointer+Offset Way To Access The Variable.\*/

    for(int i=0;i<NumEntris;i++)

        {

            cout<<"Please Enter Your Integrets That You Wish To Store Number: "<<i<<'\n';

            cin>>\*(PointedIntegersPointer+i);

        }

    cout<<"Output The  Secoend PointedIntegersPointer Adress: \n";

    cout<<hex<<PointedIntegersPointer<<'\n';

    for(int i=0;i<NumEntris;i++)

        {

            cout<<"Your Stored "<< i<<" Elements is: \n";

            cout<<\*(PointedIntegersPointer++)<<'\n';

        }

    /\*Make The Pointer Back To The Initial Position Of Arry，

    Because We Adapted The Pointer++ Way To Access The Variable.

    This Way Make The Pointer Arrived The Top Of Arry.\*/

    cout<<"Output The Third PointedIntegersPointer Adress: \n";

    cout<<hex<<PointedIntegersPointer<<'\n';

    PointedIntegersPointer -= NumEntris;

    cout<<"Output The Fourth PointedIntegersPointer Adress: \n";

    cout<<hex<<PointedIntegersPointer<<'\n'<<endl;

    delete[] PointedIntegersPointer;

    return 0;

}

输出：

How Much Integrets You Wish To Store:

2

The Initial PointedIntegersPointer Adress:

0xd81640

Please Enter Your Integrets That You Wish To Store Number: 0

1

Please Enter Your Integrets That You Wish To Store Number: 1

2

Output The Secoend PointedIntegersPointer Adress:

0xd81640

Your Stored 0 Elements is:

1

Your Stored 1 Elements is:

2

Output The Third PointedIntegersPointer Adress:

0xd81648

Output The Fourth PointedIntegersPointer Adress:

0xd81640

1. 有三种情况对指针的定义（以整型为例）：

int\* const Pointer = &Variable;①

const int\* Pointer = &Variable;②

const int\* const Pointer = &Variable;③

①定义的是一个常量指针，指向一个变量。

②定义的是一个指针，该指针指向一个常量。

③定义的是一个常量指针指向的是一个常量。

所以我们的修饰符修饰的是离它最近的右边的一个量或者类型。

常量指针（简称常指针如：①）它所指向的变量的地址不能被修改，始终指向该变量。指向常量类型的指针如②：它不能通过指针改变所指向的变量的值，但是可以修改它所指的地址（既可以指向别的变量）。在③中：既不能通过指针改变所指变量的值，也不能改变自己所指的地址。（注意：②和③中只是不能通过指针修改变量的值，与变量自己能不能修改自己的值无关既：②和③中既可以指向const int Variable，也可以指向int Variable）。

1. new(nothrow)所申请的空间，这个空间申请失败时会返回一个NULL而不是例外，如果成功则会返回一个正常的地址，所以在使用new(nothrow)做一个申请内存较大的项目的时候我们的日志中必须要有new(nothrow)申请的地址。

## 取地址符的用法：

1. 变量类型& 相同变量1 = 变量名;

当我们要对地址进行赋值时：变量类型& 人为定义变量地址名2 = 相同变量1;

这时候：相同变量2完全等同于人为定义变量地址名2，变量名。既：三者的地址和数值都一样，这个和指针很像但是有区别，指针是有一个独立的变量，只是存了别的变量的地址，自己本身也有自己的地址，不过我们上面两个的定义是可以取到变量的地址和值，但是我们不能在对其进行取地址既&&相同变量1或者&&相同变量2，因为它们只是一个变量而不是一个指针它们的地址都一样的，存储的是变量值，地址就是本身变量的地址（实际这里叫做取地址符的用法不对，但是没有好的翻译，就用原本英文reference代替了）。

## 使用const修饰reference

1. 当我们共同开发项目在数据传递时不想被别人修改产生错误时，就可以用const来修饰，例如：我们的函数在传递地址时，不想让别人对我们的输入参数进行修改时就可以在对reference定义时前面加const修饰如：变量类型& 相同变量1 = 变量名，注意变量有没有被const修饰与我们相同变量能否被修饰无关，虽然相同变量1值不能被修改，但是变量的值还是可以变化的，我们最终变化完后才传输到我们的相同变量1，所以这里并不会矛盾，例如：

#include <iostream>

using namespace std;

int main() {

    int Variable1 = 30;

    const int& Reference1 = Variable1;

    Variable1 =60;

    cout<<"Reference1 Value: \n";

    cout<<Reference1<<'\n';

    cout<<"Reference1 Adress: \n";

    cout<<&Reference1<<'\n';

    cout<<"Variable1 Value: \n";

    cout<<Variable1<<'\n';

    cout<<"Variable1 Adress: \n";

    cout<<&Variable1<<'\n'<<endl;

    return 0;

}

输出：

Reference1 Value:

60

Reference1 Adress:

0x61fe14

Variable1 Value:

60

Variable1 Adress:

0x61fe14

分析：虽然我们的变量在Reference1被const修饰后又修改值，但是最后我们两个变量是完全相同的，是因为在对Variable1修改完后才对Reference1进行覆盖，所以我们只是不能对Reference1直接进行修改，如果没有const的修改那么两个变量除了名字不一样其他的都一样，包括功能，在对Reference1进行修改时Variable也会被修改，所以为了变量的安全，我们一般在进行数据传递时都会用const，以免被别人修改。

# 第九课总结

1. 类的定义：

class 类名{

变量1;

变量2;

...

}

1. 普通对象的定义：类名 对象名

使用动态分配定义对象：类名 \* 对象名 = new 类名();

普通对象的访问方式：对象名.成员变量;

动态分配定义对象的访问方式：(\*对象名).成员变量;或者对象名->成员变量;注意当我们使用完动态分配变量时我们必须删除变量。

public和private：public表示用可以直接访问的变量，而private是不可直接访问的变量，可能会由public中的变量操作后输出的变量，也有可能直接无法访问。

1. 构造函数（constructor）:

构造函数是定义在类中的一个函数，且该函数的函数名和类是一个名字没有返回值，函数的声明可以在类的内部也可以在外部。该函数在当对象被定义时就被调用，可以用来初始化变量。构造函数也可以重载，在定义对象时传入不同的参数决定使用哪一个构造函数，当我们只有有参构造函数时，这时构造函数需要传入参数如果我们在定义对象时没有传入参数那就意味着在定义对象时不使用构造函数就造成冲突导致错误。

构造函数的使用：类名 对象名(参数1，参数2);其中的参数是构造函数需要传入的参数。

1. 析构函数（destructor）：

析构函数的定义：~类名();析构函数也是自动调用的是在对象空间被回收之前才调用，一般在mian函数执行最后一句语言之前，除非特别说明要提前回收对象的空间。同样析构函数的声明可以在类的内部也可以在外部，注意析构函数没有参数，不能重载，析构函数内部代码主要是重置变量和释放掉动态变量占用的内存空间以及其他空间资源的回收。

1. 拷贝构造函数：

深度拷贝：我们在拷贝对象时是这样的：类名 对象名（要拷贝的对象）;深度拷贝构造函数其实就是拷贝整个对象。我们如果单纯直接这样拷贝，其实是浅拷贝，在内存空间上，其实是两个指针指向同一个对象而已。

当我们类中有重载构的造函数且重载的函数的参数是对象的地址，这时我们的代码：

类名 对象名（被拷贝的对象），这时不再是浅拷贝了，而是重新定义了一个对象，被拷贝的对象被当作进入另一个重载的参数，这时我们就可以把第一个对象的变量的值或地址赋值到对应的第二个对象中去。

具体做法：在类中重载一个这样的构造函数：类名(const 类名& 对象名){用传进来的第一个对象对第二个对象变量进行拷贝}，其中const是不允许在传参时改变传入参数的数据。然后这样我们在mian中再对对象进行传参：类名 对象名（被拷贝的对象），这样就完成了深度拷贝。（代码过多见：Copy Constructor）

1. 一个类只能创建一个对象的方法：

我们知道对象是通过构造函数来实例化对象的，我们如果没有自己写构造函数系统会默认帮我们跑一个空的构造函数来产生实例，如果有自己的构造函数时系统就会覆盖原来的那个空构造函数，所以如果我们把构造函数或者复制构造函数私有在一个类中，那么我们的这个类就无法直接产生对象，这时我们只能间接产生，我们就在公共的部分写一个函数来产生一个静态对象（因为私有部分只能在类的内部进行访问，这时候在类的内部实例化对象可以调用构造函数），这时候在返回所产生的对象，这时我们就可以通过类来调用这个函数就可以得到一个对象，为了防止赋值而改变我们的唯一对象，我们需要声明对象的赋值运算为常量。（代码见P:268页）

1. 一个类禁止在栈上实例化：

当一个类需要释放时是需要调用一个析构函数的，如果我们把我们的析构函数私有化的话，我们就无法在堆上直接对类进行实例化，可以使用new这样的自由空间来实例化一个对象，但是我们要想用delete来删除这个对象时我们会无法直接删除，因为析构函数被私有化，所以我们的main()函数不能自动调用这个析构函数，我们只能在类的内部使用一个静态函数来删除我们的对象，因为在内部我们的类能够调用私有化的析构函数。（代码见p271）

1. 构造函数初始化表达式：

构造函数(参数):要初始化的变量(参数){}，把参数的值赋值给要初始化的变量。

1. friend关键字：

friend修饰的变量可以访问该类中所有的成员包括私有成员。

# 第十课总结

1. 继承只能继承public中的成员
2. public继承：

class 子类名：public父类名，其中子类相当于独立拥有了父类的成员，可以进行独立访问，各个子类之间互不影响。

1. 继承中访问protected：

当被保护时我们只能在子类的内部对被保护的成员进行访问，不能进行外部访问。

1. 子类覆盖父类成员：

当子类重新定义一个变量与父类名字一样有冲突时子类覆盖父类变量，但是我们额可以通过父类来调用被冲突的变量，这时就会调用父类中的变量了。当父类中有重载函数时，我们子类中的函数只要和一个函数冲突那么我们的子类就会隐藏父类中的其他重载函数，这时我们就无法直接从子类中访问重载函数解决方式有：

1. 子类对象.父类名::重载函数();
2. 在子类中重载然后在函数中调用父类。

using 父类::函数名；这样就不会隐藏父类的重载函数了。

1. 不管有多少构造函数和析构函数记住：先构造的后析构。
2. private继承：

私有继承时相当于把父类中public的成员作为子类的私有，子类只能在内部访问。

1. 当我们把继承的子类对象的值传递给父类的对象的值时会导致只传递了父类的值，子类会被切片，所以我们在遇到这样的情况时应该用地址或者指针传递。
2. 多重继承制：

我们要对多个父类进行继承时可以用逗号隔开：class 子类名:public 父类1,

public 父类2......

1. final禁止重写，禁止继承，禁止函数重载