C++入门

第一个C++程序

```
#include<iostream>
//标准输入输出流
using namespace std;
//std C++标准库的命名空间
int main()
{
    //cout 标准的输出
    //endl 结束换行
    cout << "hello world" << endl;
    std::cout << "hello world" << std::endl;
    //:: 作用域运算符
    return 0;
}
```

- 使用cout标准输出和cin标准输入时，必须包含头文件和std标准命名空间；
- 使用C++输出输入，不需要增加数据格式控制；
- :: 作用域运算符，可以再函数中调用全局变量；

命名空间

命名空间主要是用来解决命名冲突得问题。

```
namespace N
{
    //函数
    //变量
    //结构体
    //类
    //命名空间
}
```

使用命名空间的方式

```
namespace N
{
    int a=10;
    int b=20;
    int Add(int left, int right)
    {
        return left+right;
    }
    int Sub(int left, int right)
    {
        return left-right;
    }
}
```

```
1.加命名空间名称及作用域限定符
int main()
{
    printf("%d\n",N::a);
    return 0;
}
2.使用using将命名空间中成员引入
using N::b;
int main()
{
    printf("%d\n",b);
    return 0;
}
3.使用using namespace将命名空间引入
using namespace N;
int main()
{
    printf("%d\n",Add(10,20));
    return 0;
}
```

- 命名空间必须定义在全局作用域下；
- 命名空间下可以放函数、变量、结构体、类；
- 命名空间可以嵌套命名空间；
- 同一个工程中语序存在多个相同的名称的命名空间，编译器最后会合成到同一个命名空间中；
- 命名空间是开放，可以随时加入新的成员，最后编译器会将他们合并成一个命名空间；

缺省参数

缺省参数是**声明或定义函数时**为函数的**参数指定一个默认值**。在调用该函数时，如果没有指定实参则采用该默认值，否则使用指定的实参 。

全缺省参数

函数的每一个参数都带有缺省值。

```
//全缺省参数
void Fun(int a=10,int b=20,int c=30)
{
    cout<<"a="<<a<<endl;
    cout<<"b="<<b<<endl;
    cout<<"c="<<c<<endl;
}
```

半缺省参数

函数一部分参数带有缺省值。

```
//半缺省参数
void Fun(int a,int b,int c=30)
{
    cout<<"a="<<a<<endl;
    cout<<"b="<<b<<endl;
    cout<<"c="<<c<<endl;
}
```

1. 半缺省必须从右往左连续缺省
2. 缺省值必须是常量或者全局变量
3. 调用时，如果要传参必须从左往右依次传参，不能空缺

函数重载

函数重载是指**在同一作用域内**，可以有一组**具有相同函数名**，**不同参数列表**(个数/类型/顺序)的函数，这组函数被称为重载函数。

```
//函数重载
//函数名相同，参数不同(类型/个数/顺序)
int Add(int left,int right)
{
    return left+right;
}
double Add(double left,double right)
{
    return left+right;
}
long Add(long left,long right)
{
    return left+right;
}
int main()
{
    return 0;
}
```

- 只有返回值不同，是不能构成重载的；

函数名修饰

为什么C++支持函数重载，C语言不支持?

- 在C/C++中一个程序需要运行起来，需要经历的阶段
    - 1、预处理——头文件的展开、宏替换、条件编译、去掉注释；生成 .i文件
    - 2、编译——检查语法，生成汇编代码；生成 .s文件
    - 3、汇编——汇编代码转成二进制的机器码；生成 .o文件
    - 4、链接——将目标文件链接到一起；生成可执行程序

```
#include <stdio.h>
int add(int a,int b)
{
        return a+b;
}
int main()
{
        return 0;
}
```

```
00000000004004cd <add>:
  4004cd:       55                      push   %rbp
  4004ce:       48 89 e5                mov    %rsp,%rbp
  4004d1:       89 7d fc                mov    %edi,-0x4(%rbp)
  4004d4:       89 75 f8                mov    %esi,-0x8(%rbp)
  4004d7:       8b 45 f8                mov    -0x8(%rbp),%eax
  4004da:       8b 55 fc                mov    -0x4(%rbp),%edx
  4004dd:       01 d0                   add    %edx,%eax
  4004df:       5d                      pop    %rbp
  4004e0:       c3                      retq
```

- C语言在编译完成后，汇编代码中函数名没有发生任何改变；

```
#include <stdio.h>
int add(int a,int b)
{
        return a+b;
}
int main()
{
        return 0;
}
```

```
00000000004004fd <_Z3addii>:
  4004fd:       55                      push   %rbp
  4004fe:       48 89 e5                mov    %rsp,%rbp
  400501:       89 7d fc                mov    %edi,-0x4(%rbp)
  400504:       89 75 f8                mov    %esi,-0x8(%rbp)
  400507:       8b 45 f8                mov    -0x8(%rbp),%eax
  40050a:       8b 55 fc                mov    -0x4(%rbp),%edx
  40050d:       01 d0                   add    %edx,%eax
  40050f:       5d                      pop    %rbp
  400510:       c3                      retq
```

- C++在编译完成后，汇编代码中函数名的修饰发生改变，编译器讲函数参数类型信息添加到修改后的名字中；
- C语言不支持函数重载，因为同名函数没办法区别；而C++中通过函数名修饰规则来区分，只要参数不同，修饰出来的函数名就不同，就可以支持重载；

extern “C”

由于C++支持函数重载，编译器在编译时会对函数名进行修饰，而C语言不支持函数重载，函数名不会进行修饰；**为了能够正确实现C++代码调用其他的C语言代码，加上extern “C”后，会指示编译器这部分代码按照C语言的方式进行编译。**

引用

引用就是给已存在变量取了一个别名，编译器不会为引用变量开辟内存空间，它和它引用的变量共同用一块内存空间。

类型& 引用变量名(对象名)=引用实体;

```
void Test()
{
    int a=1;
    //引用
    int& ra=a;
    printf("%d\n",a);
    printf("%d\n",ra);
}
```

- 引用类型和引用实体必须是同种类型

引用特性

1. 引用必须在定义的时候初始化
2. 一个变量可以有多个引用
3. 引用一旦引用一个实体，在不能引用其他实体

```
void Test()
{
    int a=10;
    int &ra;//编译时会报错
    int &b=a;
    int &c=a;
    int &d=b;//一个变量可以有多个引用
    int &ra=a;//引用在定义的时候初始化
}
```

常引用

```
 void Test()
 {
     const int a=10;
     //a现在是一个int类型的常量
     int& b=a;
     //编译出错
     //a是const int类型(只读),b是int类型(可读可写),所以编译出错
     const int& b=a;
     //常引用

     int c=20;
     int& d=c;
     const int& e=c;
     //常引用
 }
```

- 引用去别名时，变量访问的权限可以缩小，但是不能放大

使用场景

1. 引用做参数

```
void swap_cpp(int& left, int& right)
    //引用做参数
    //left相当于a的别名，right相当于b的别名
{
    int temp = left;
    left = right;
    right = temp;
}
int main()
{
    int a=10,b=20;
    swap_cpp(a,b);
}
```

2. 引用做返回值

```
int Count1()、
    //传值返回
{
    //static只是改变了变量的生命周期
    static int n = 0;
    n++;
    return n;
}
//返回过程中产生一个临时变量，将n的值拷贝进去
int& Count2()
    //引用做返回值
{
    //static只是改变了变量的生命周期
    static int n = 0;
    n++;
    return n;
}
//返回过程中产生一个n的别名
int main()
{
    const int& a=Count1();
    //临时变量具有常性，所以需要加上const
    int& b=Count2();
    return 0;
}
```

- 一个函数要使用引用返回，返回变量出了这个函数的作用域还存在就可以只用引用返回，否则就不安全。

引用和指针的区别

从语法概念上讲，引用就是一个别名，没有独立空间，和其引用实体共同用一块空间；但在底层实现上，引用是按照指针的方式来实现的。

1. 引用在定义时必须初始化，指针没有要求；
2. 引用在初始化引用一个实体后，就不能再引用其他实体，而指针可以在任何时候指向任何一个同类型的实体；
3. 没有NULL引用，但是有NULL指针；
4. 没有多级引用，但是有多级指针
5. 在sizeof中，引用的结果为引用类型的大小，指针始终为地址空间所占字节个数；

6. 引用自加即引用的实体增加1，指针自加即指针向后偏移一个类型的大小；

内联函数

内联函数是指用inline关键字修饰的函数。编译时C++会在调用内联函数的地方展开，没有函数压栈的开销，以空间换时间来提示程序的运行效率。

- 一般内联函数适用于函数体比较短的函数，代码比较长/含有递归的函数不适宜作为内联函数。

1. C++有哪些技术替代宏?
    1. 常量定义，换用const
    2. 短小没有递归的函数，换用内联函数
2. 宏的优缺点?
    1. 优点
        - 增强代码的复用性；
        - 提高性能；
    2. 缺点
        - 不方便调试宏(因为预编译阶段就进行了替换)
        - 导致代码可读性差，可维护性差
        - 没有类型安全的检查

auto关键字(11)

auto声明的变量必须是由编译器在编译时期推导而得，**使用auto定义变量时必须对其进行初始化。**

```
int main()
{
    int a=10;
    auto b=a;
    //这里b的类型是根据a的类型推导出来的
    auto& c=a;
    //int类型
    auto d=&a;
    //int *类型
    return 0;
}
```

- 用auto声明指针类型时，用auto和auto*没有任何区别，但是auto声明引用类型时则必须加&；
- 当在同一行用auto声明多个变量时，这些变量必须是相同的类型，否则编译器将会报错，因为编译器实际只对第一个类型进行推导，然后用推导出来的类型定义其他变量；
- auto不能作为函数的参数，auto不能直接用来声明数组；

范围for(11)

```
int main()
{
    int array[]={1,2,3,4};
    //访问并打印数组中的内容
    for(auto e : array)
    {
        cout<<e<<" ";
    }
    return 0;
}
```

使用条件

1. for循环迭代的范围必须是确定的
2. 迭代的对象要实现++和==的操作

指针空值nullptr(11)

在C++98中，字面常量0既可以表示一个整形数字，也可以表示无类型的指针(void *)常量；所以在C++11中，引入了nullptr来解决这一矛盾。

- 在使用nullptr表示指针空值时，不需要包含头文件
- C++11中，sizeof(nullptr)和sizeof((void*)0)所占字节数相同