§4. 函数

4.0. 为什么要引入函数

★ 目前为止的所讲的内容及作业都是只有一个main函数，
负责完成一个程序的所有功能

例：输入两数求max

```
#include <iostream>
using namespace std;
int main()
{
    int a,b,m;
    cin >> a >> b;
    m = a > b ? a : b;
    cout << "max=" << m;
    return 0;
}
```

```
//例：函数形式求两数最大值
#include <iostream>
using namespace std;
int max(int x, int y)
{
    int z;
    if (x>y) z=x;
    else z=y;
    return (z);
}
int main()
{
    int a,b,m;
    cin >> a >> b;
    m=max(a,b);
    cout << "max=" << m;
    return 0;
}
```

例：人民币转大写

① 输入一个浮点数

② 分解各位

③ 每位依次转为汉字

（大量重复，仅小部分不同）

```
switch(shiyi) {
    case 9:
        cout << "玖拾";
        break;
    ...
    case 1:
        cout << "壹拾";
        break;
    case 0:
        break;
    }
switch(yi) {
    case 9:
        cout << "玖亿";
        break;
    ...
    case 1:
        cout << "壹亿";
        break;
    case 0:
        if (shiyi>0)
            cout << "亿";
        break;
    }
```

```
void daxie(int num, int flag)
{
    switch(num) {
        case 0:
            if (flag)
                cout << "零";
            break;
        case 1:
            cout << "壹";
            break;
        ...
        case 9:
            cout << "玖";
            break;
        }
}
在main函数中：
    daxie(shiyi, ***);
    ... //可能需要的其它语句
    daxie(yi, ***);
    ... //可能需要的其它语句
```

- 能否不同功能分开，使程序逻辑更明确？
- 重复的代码能否只写一遍？
（如何体现差别部分）

§4. 函数

4.1. 概述

★ C/C++程序的基本组成单位

★ 一个函数实现一个特定的功能

★ 有且仅有一个main函数，程序执行从main开始

★ 函数平行定义，嵌套调用

★ 一个源程序文件由多个函数组成，一个程序可由多个源程序文件组成

```
//a1.cpp                //a2.cpp              //a3.cpp
int fun1( )             float fun5( )         double fun4( )
{                       {                     {
}                       }                     }
short fun2( )                                 int main()
{                                             {
}                                             }
long fun6( )
{                                   一个程序由3个源程序文件组成
}                                   共6个函数，有且仅有一个main
```

★ 函数的分类

用户使用角度
- 标准函数（库函数）由系统提供
- 自定义函数　　用户自己编写

● 在使用上无任何的区别

函数形式
- 无参　调用与被调用函数间无数据传递
- 有参　调用与被调用函数间有数据传递

§4. 函数

4.2. 函数的定义

4.2.1. 无参函数的定义

函数返回类型 函数名（[void]）{ () | (void) }

{

函数体 { 声明语句 | 执行语句 }

}

```
int fun()
{
    cout << "***" << endl;
    return 0;
}
int fun(void)
{
    cout << "***" << endl;
    return 0;
}
```

★ 函数名的命名规则同变量

★ 返回类型与数据类型相同

```
int fun(void)
{
    …
}
long fun2()
{
    …
}
```

★ 返回类型可以是void，表示不需要返回类型

```
void fun3()
{
}
```

★ C缺省返回类型为int(不建议缺省，int也写)，C++不支持默认int，必须写

```
fun3(…)
{
}//C++编译报错
```

§4. 函数

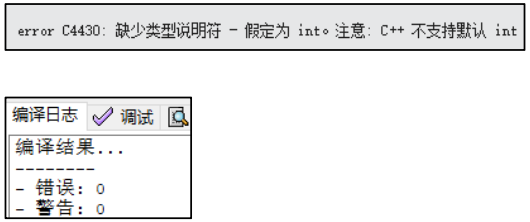
4.2. 函数的定义

4.2.1. 无参函数的定义

★ ANSI C++要求main函数的返回值只能是int并且不能缺省不写，否则编译会报错；但部分编译器可缺省不写；
VS系列还允许void等其它类型(建议唯一int)

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

main()
{
   return 0;
}
//VS2019报error
//Dev正确
```

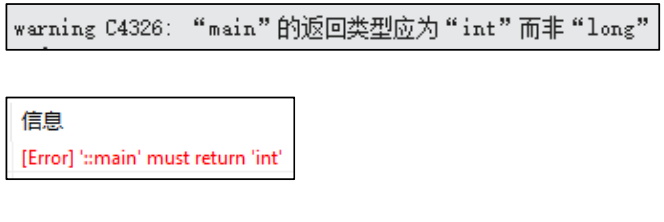



```
#include <iostream>
using namespace std;

void main()
{
    return;
}
//VS2019报warning
//Dev报error
```


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

long main()
{
    return 0L;
}
//VS2019报warning
//Dev报error
```



§4. 函数

4.2. 函数的定义

4.2.2. 有参函数的定义

函数返回类型 函数名（形式参数表）

{

函数体 { 声明语句 / 执行语句

}

```
int max(int x, int y)
{
    int z;          /* 声明语句 */
    if (x>y)
        z=x;
    else
        z=y;
    return z;
}
```

```
return x>y?x:y;
```

★ 函数名的命名规则同变量

★ 返回类型与数据类型相同

★ 返回类型可以是void，表示不需要返回类型

★ C缺省返回类型为int(不建议缺省，int也写)，C++不支持默认int，必须写

§4. 函数

4.3. 函数参数与函数的值

4.3.1. 形式参数与实际参数

形式参数：在被调用函数中出现的参数

实际参数：在调用函数中出现的参数

```
int main()
{
   int i=15,j=10,m;
   m = max(i, j);
   cout<< "max=" <<m;
   return 0;
}
```

i,j为实参

```
int max(int x, int y)
{
   int z;
   z = x>y ? x : y;
   return z;
}
```

x,y为形参

```
int main()
{
   int x=15,y=10,m;
   m = max(x, y);
   cout<< "max=" <<m;
   return 0;
}
```

x,y为实参

```
int max(int x, int y)
{
   int z;
   z = x>y ? x : y;
   return z;
}
```

x,y为形参

允许同名

★ 实参与形参分别占用不同的内存空间，实形参名称既可以相同，也可以不同

★ 参数的传递方式是"单向传值"，即将实参的值复制一份到形参中(理解为 形参=实参 的形式)

§4. 函数

4.3. 函数参数与函数的值

4.3.1. 形式参数与实际参数

★ 实参与形参分别占用不同的内存空间

★ 参数的传递方式是"单向传值"，即将实参的值复制一份到形参中(理解为 形参=实参 的形式)

★ 执行后，形参的变化不影响实参值

```
#include <iostream>
using namespace std;
void fun(int x)
{   cout << "x1=" << x << endl;
    x=5;
    cout << "x2=" << x << endl;
}
int main( )
{   int k=15;
    cout << "k1=" << k << endl;
    fun(k);
    cout << "k2=" << k << endl;
    return 0;
}
```

```
k1=15
x1=15
x2=5
k2=15
```

★ 实参可以是常量、变量、表达式，形参只能是变量

```
int main()
{   int k=10;
    fun(2+k*3);
    return 0;
}
```

```
void fun(int x)
{
    ...
}
```

x = 2+k*3

§4. 函数

4.3. 函数参数与函数的值

4.3.1. 形式参数与实际参数

★ 形参在使用时分配空间，函数运行结束后释放空间

```
int main()        void f1(int x)     void f2(int y)
{  f1(10);        {  ...              { ...
   f2(15);        }                   }
   ...
}
```

x和y可能共用4个字节的空间

```
int main()       void f1(int x)
{  ...            {  ...
   f1(..);        }
   ...
   f1(..);
   ...
   f1(..);
   ...
}
```

1、假设main中调用10000次f1()，
则x的分配释放会重复10000次

2、每次x分配的4字节不保证是同一个空间

§4. 函数

4.3. 函数参数与函数的值

4.3.1. 形式参数与实际参数

★ 实参、形参类型必须一致，否则结果可能不正确

```
#include <iostream>
using namespace std;
int fun(short x)
{
    cout << "x=" << x << endl;
    return 0;
}
int main()
{
    long k=70000;
    fun(k); //编译有警告
    cout << "k=" << k << endl;
    return 0;
}
```

x=4464

实形参类型不一致时，
转换规则同赋值（形参 = 实参）

warning C4244: “参数”: 从“long”转换到“short”，可能丢失数据

k=70000

§4. 函数

4.3. 函数参数与函数的值

4.3.1. 形式参数与实际参数

4.3.2. 函数的值（函数的返回值）

★ 通过return语句获得，若return类型与返回类型定义不一致，以返回类型为准进行数据转换

```
... f(...)                  int fun(...)
{                           {
    int k;                      int s;
    ...                         ...
    k = fun(...);               ...
    ...                         return s;
}                           }
```

若s=10
则k=10

理解为
调用函数中值=return后值的形式

```
long fun2()
{
  long a;
  …
  return a;
} 正确
```

```
long fun2()
{
   short a;
   …
   return a;
} 正确
```

```
short fun3()
{
   long a;
   …
   return a;
} 可能不正确
```

```
//问1：运行结果（d的值是多少？）
//问2：哪句会有warning错？
#include <iostream>
using namespace std;

short fun3()
{
    long a = 70000;
    return a;
}
int main()
{
    long d;
    d = fun3();
    cout << d << endl;

    return 0;
}
```

warning C4244: “参数”: 从“long”转换到“short”，可能丢失数据

0000000000000001 0001000101110000

0001000101110000

0000000000000000 0001000101110000

§4. 函数

4.3. 函数参数与函数的值

4.3.1. 形式参数与实际参数

4.3.2. 函数的值（函数的返回值）

★ return后可以是变量、常量、表达式，有两种形式(带括号、不带括号)

```
return a;          return k*2;
return (a);        return (k*2);
```

★ 若函数不要求有返回值，则指定返回类型为void

```
void fun1()        int main()     int fun()
{                  {              {
   ...               ...            ...
   return;           return 0;      return 0;
}                  }              }
```

无return语句
空return语句

return int 型

return int型

返回类型非void的函数，如果不带return语句，
不同编译器表现不同(error/warning/不报错)
VS2019：main无return不报错，其余函数报error

=> 推论：① 返回类型为void的函数不能出现在除逗号表达式外的任何表达式中
② 若逗号表达式要参与其它运算，则不能做为最后一个表达式出现

```
error C2186: "+": "void"类型的操作数非法
error C2679: 二元"<<": 没有找到接受"void"类型的右操作数的运算符(或没有可接受的转换)
```

```
#include <iostream>
using namespace std;
void f()
{   int x=10;
}
int main()
{
    int k=10;
    k=k+f();  //编译错
    k,f();    //可编译通过，无意义
    cout << (k, f()) << endl;       //编译错
    cout << (k, f(), k+2) << endl; //可编译通过
    return 0;
}
```

§4. 函数

4.3. 函数参数与函数的值

4.3.1. 形式参数与实际参数

4.3.2. 函数的值（函数的返回值）

★ 一个return只能带回一个返回值

★ 函数中可以有多个return语句，但只能根据条件执行其中的一个，执行return后，函数调用结束(return后的语句不会被执行到)

```
int fun(void)
{
    if (...)
        return ...;
    else
        return ...;
    ....;  //无法被执行到
}
```

★ 如果函数中有分支语句，但return未覆盖全部分支，则VS2019会报warning错(不会判断条件是否覆盖!)

```
int fun(int x)
{
    if (x>1) {
        if (x>10)
            return 1;
        }
    else
        return 2;
} //报warning
```

```
int fun(int x)
{
   if (x>1)
        return 1;
   else if (x<=1)
        return 2;
} //仍会报warning
```

为什么？计算机思维如何理解？
人的思维如何理解？

warning C4715: “f”: 不是所有的控件路径都返回值

§4. 函数

4.4. 函数的调用

函数的编写方法：

通过第2-3章的基本知识，定义不同数据类型的变量，采用顺序、分支、循环等基本结构，按照函数的预期功能来编写每个函数

§4. 函数

4.4. 函数的调用

4.4.1. 基本形式

函数名()　　：适用于无参函数

函数名(实参表列)：适用于有参函数，用，分开

与形参表的个数、顺序、类型一致

★ 若同一变量同时出现在一个函数的多个参数中，且有自增、赋值、复合赋值等改变变量值的操作，则不同编译器处理的方式可能不同（不在讨论，也不建议深入）

fun(i, ++i)

- 从左至右：fun(3,4)
- 从右至左：fun(4,4)

注意：fun(i++, --j) 这种不同变量是必须讨论的

printf/scanf等函数有参数个数、类型不等的情况出现，称为可变参数方式，本课程暂不讨论

```
printf("%d\n", a);          //2个参数
printf("%d %d\n", a, b);    //3个参数
scanf("%d", &a);            //2个参数
scanf("%d %d", &a, &b);     //3个参数
```

§4. 函数

4.4. 函数的调用

4.4.2. 调用方式

函数语句：函数调用+；

```
printf("Hello.\n");
putchar('A');
```

函数表达式：出现在某个表达式中

```
c=max(a,b)+4;
k=sqrt(m);
```

函数参数：作为另一个函数的参数

```
printf("max=%d", max(a,b));
putchar( getchar() );
sqrt( fabs(x) );
```

函数返回类型不能是void

问题：其它函数的返回值可由调用函数使用，main的返回值给谁？

★ 函数调用时，不能写返回类型

定义及实现时：	调用时：
long f1() { ... } int max(int x, int y) { ... }	k = f1(); ✓ k = long f1(); × k = max(i, j); ✓ k = int max(i,j); ×

★ 无参函数调用时，参数位置不能写void

定义及实现时：	调用时：

int fun() //空
{ ...
}
int fun(void) //写void
{ ...
}
	k = fun(); ✓
	k = fun(void); ×

★ 有参函数调用时，实参不能写类型

定义及实现时：	调用时：
int max(int x, int y) { ... }	int i=10, j=15; k=max(i, j); ✓ k=max(int i, int j); ×

§4. 函数

4.4. 函数的调用

4.4.1. 基本形式

4.4.2. 调用方式

4.4.3. 对被调用函数的说明

★ 对库函数，加相应的头文件说明

```
#include <cstdio>   输入输出函数
#include <cmath>    数学运算函数
#include <cstring>  字符串运算函数
```

注意：<cstdio>和<cmath>这两个头文件在VS2019中缺省可以不加，其它编译器一般需要加

★ 对自定义函数，在调用前加以说明，位置在调用函数前/整个函数定义前

两种方法：

返回类型 函数名(形参类型);

返回类型 函数名(形参类型 形参表);

```
int max(int, int);
int main()
{
    k=max(i,j);
}
int max(int x, int y)
{
    ...
}
```

```
int max(int x, int y);
int main()
{
    k=max(i,j);
}
int max(int x, int y)
{
    ...
}
```

```
int max(int p, int q);
        //pq不要求与实现中xy一致
int main()
{
    k=max(i,j);
}
int max(int x, int y)
{
    ...
}
```

§4. 函数

4.4. 函数的调用

4.4.1. 基本形式

4.4.2. 调用方式

4.4.3. 对被调用函数的说明

★ 对库函数，加相应的头文件说明

★ 对自定义函数，在调用前加以说明，位置在调用函数前/整个函数定义前

★ 若被调用函数出现在调用函数之前，可以不加说明(有些编译器可能必须加)

```
//可以没有说明
float fun()
{   ...
}
int main()
{
    float k;
    k=fun();
    return 0;
}
```

```
float fun(); //必须有说明
int main()
{
    float k;
    k=fun();
    return 0;
}
float fun()
{   ...
}
```

§4. 函数

4.4. 函数的调用

4.4.3. 对被调用函数的说明

★ 调用说明可以在函数外，针对后面所有函数均适用；也可在函数内部，只对本函数有效

```
int max(int x,int y);
int main()
{
    ..max(...); ✓
}
int f1()
{
    ..max(...); ✓
}
int max(int x, int y)
{
    ....
}
```

```
int main()
{
    int max(int,int);
    ..max(...); ✓
}
int f1()
{
    ..max(...); ×
}
int max(int x, int y)
{
    ....
}
```

```
int main()
{
    ..max(...); ?
}
int max(int x, int y);
int f1()
{
    ..max(...); ?
}
int max(int x, int y)
{
    ....
}
```

```
int main()
{
    int max(int,int);
    ..max(...); ?
}
int max(int x, int y)
{
    ....
}
int f1()
{
    ..max(...); ?
}
```

§4. 函数

4.5. 函数的嵌套调用

4.5.1. C++程序的执行过程

例：程序如下

```
void b()
{
    ...
}
void a()
{   ...
    b();
    ...
}
int main()
{   ...
    a();
    ...
    return 0;
}
```

//左例，9步

(1) 执行main函数的开头部分

(2) 遇到调用a函数的语句，流程转去a函数

(3) 执行a函数的开头部分

(4) 遇到调用b函数的语句，流程转去b函数

(5) 执行b函数，如果再无其他嵌套的调用，则完成b函数的全部操作

(6) 返回原来调用b函数的位置，即返回a函数

(7) 继续执行a函数中尚未执行的部分，直到a函数结束

(8) 返回main中调用a函数的位置

(9) 继续执行main函数的剩余部分直到结束

如何返回？

§4. 函数

4.5. 函数的嵌套调用

4.5.1. C++程序的执行过程(通用描述)

（1）从main函数的第一个执行语句开始依次执行

（2）若执行到函数调用语句，则保存调用函数当前的一些系统信息（保存现场）

（3）转到被调用函数的第一个执行语句开始依次执行

（4）被调用函数执行完成后返回到调用函数的调用处，恢复调用前保存的系统信息（恢复现场）

（5）若被调用函数中仍有调用其它函数的语句，则嵌套执行步骤（2）-（4）

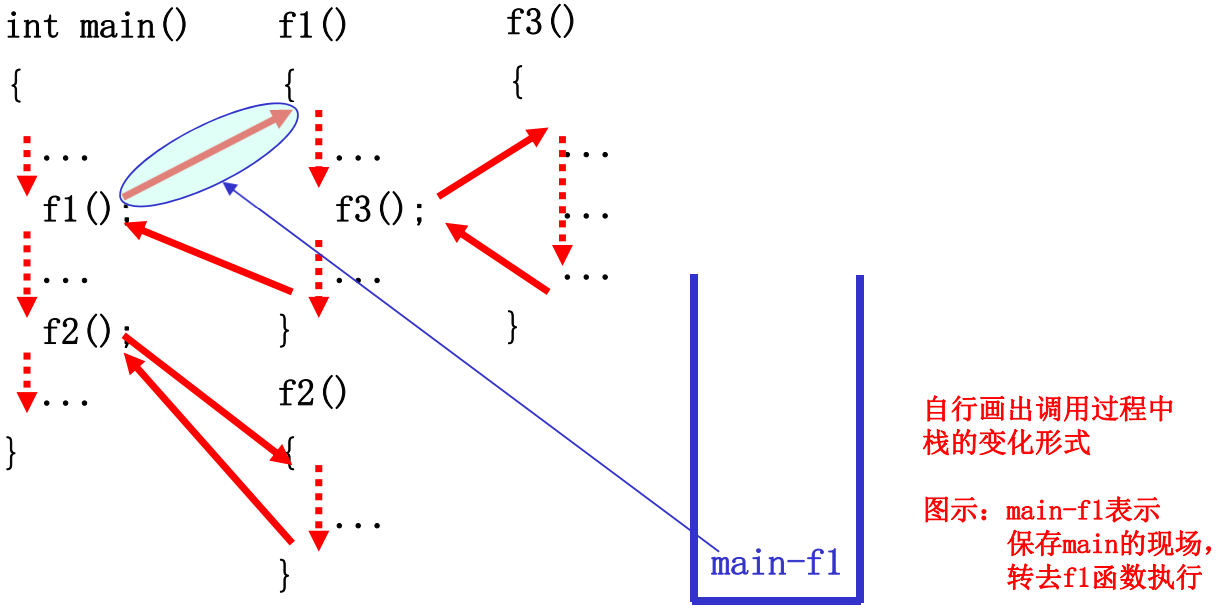
（6）所有被调用函数执行完后，顺序执行main函数的后续部分直到结束

4.5.2. 特点

★ 嵌套的层次、位置不限

★ 遵循后进先出的原则(栈)

★ 调用函数时，被调用函数与其所调用的函数的关系是透明的，适用于大程序的分工组织



自行画出调用过程中栈的变化形式

图示：main-f1表示保存main的现场，转去f1函数执行

§4. 函数

4.5. 函数的嵌套调用

4.5.3. 实例

例1：求四个整数的最大值

```
//方法1
int max2(int a, int b)
{
    if (a>b)
        return a;
    else
        return b;
}

int max4(int a, int b, int c, int d)
{
    int m;
    m = max2(a, b);
    m = max2(m, c);
    m = max2(m, d);
    return m;
}

int main()
{
   int a,b,c,d,m;

   ...输入a/b/c/d四个数字
   m = max4(a,b,c,d);
   ...输出最大值

   return 0;
}
```

```
//方法2
int max2(int a, int b)
{
    return (a>b ? a : b);
}

int max4(int a, int b, int c, int d)
{
    int m1, m2, m;
    m1 = max2(a, b);
    m2 = max2(c, d);
    m  = max2(m1, m2);
    return m;
}

int main()
{
   int a,b,c,d,m;

   ...输入a/b/c/d四个数字
   m = max4(a,b,c,d);
   ...输出最大值

   return 0;
}
```

```
int main()
{
   ...
   m = max2( max2( max2(a,b), c), d);
   ...
}
```

```
int main()
{
   ...
   m = max2( max2(a,b), max2(c,d) );
   ...
}
```

一个函数的返回值做为另一个函数的参数
(本例中函数名相同)

§4. 函数

4.5. 函数的嵌套调用

4.5.3. 实例

例2：写一个函数，判断某正整数是否素数

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

int prime(int n)
{
    int i;
    int k = int(sqrt(n));

    for(i=2; i<=k; i++)
       if (n%i == 0)
          break;

    return i<=k ? 0 : 1;
}

int main()
{
    int n;
    cin >> n; //为简化讨论，此处假设输入正确
    cout << n <<(prime(n) ? "是":"不是")<<"素数
"<<endl;
    return 0;
}
```

循环的结束有两个可能性：
1、表达式2（i<=k）不成立（是素数）
2、因为 break 而结束（不是素数）

```
//03模块例：求100~200间的素数
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int m, k, i, line=0;

    for(m=101; m<=200; m+=2) {
        k=int(sqrt(m));

        for(i=2; i<=k; i++)
            if (m%i==0)
                break;

        if (i>k) {
            cout << setw(5) << m;
            line++;
            if (line%10==0)
                cout << endl;
            }
        } //end of for

    return 0;
}
```

改写为用prime函数

```
//03模块例：求100~200间的素数
#include <iostream>
#include <iomanip>
using namespace std;
int prime(int n)
{   int i;
    int k = int(sqrt(n));

    for(i=2; i<=k; i++)
       if (n%i == 0)
          break;

    return i<=k ? 0 : 1;
}
int main()
{
    int m, line = 0;
    for(m=101; m<=200; m+=2) {
        if (prime(m)) {
            cout << setw(5) << m;
            line++;
            if (line%10==0)
                cout << endl;
            }
        }
    return 0;
}
```

§4. 函数

4.5. 函数的嵌套调用

4.5.3. 实例

例3：验证哥德巴赫猜想

```
#include <iostream>
#include <cmath>
using namespace std;
int prime(int n)
{
    int i;
    int k = int(sqrt(n));
    for(i=2; i<k; i++)
       if (n%i == 0)
          break;
    return i<k ? 0 : 1;
}
void gotbaha(int even)
{
    int x;
    for (x=3; x<=even/2; x+=2)
        if ( prime(x) + prime(even-x) == 2) {
            cout << x << "+" << even-x << "=" << even <<endl;
            break; //不要break则求出全部组合
         }
}
int main()
{
    int n;
    cin >> n; //为简化讨论，此处假设输入正确
    gotbaha(n);
    return 0;
}
```

一道题目的解可用于另一题中
强调过程的积累、经验的积累