第1章

C++回顾

函数

函数定义

```
int abc(int a, int b, int c)
{
    return a + b * c;
}
```

形式参数

函数调用

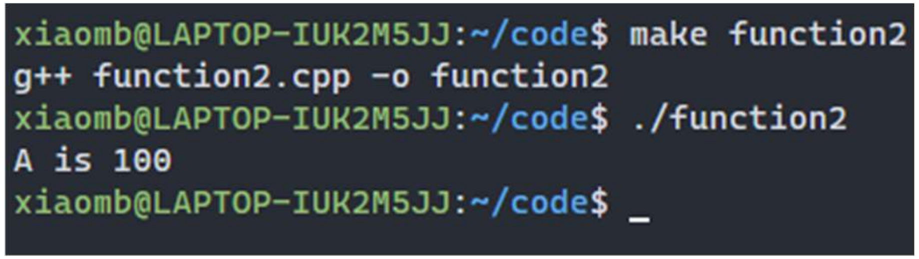
```
int main(int argc, char *argv[])
{
    int x = 3, y = 4;
    int z = abc(2, x, y);
}
```

实际参数

函数参数

- 函数参数以值传递的形式进行

```
#include <iostream>

using namespace std;

void incr1(int a)
{
    a += 1;
}

int main(int argc, char *argv[])
{
    int A = 100;
    incr1(A);
    cout << “A is “ << A << endl;

    return 0;
}
```

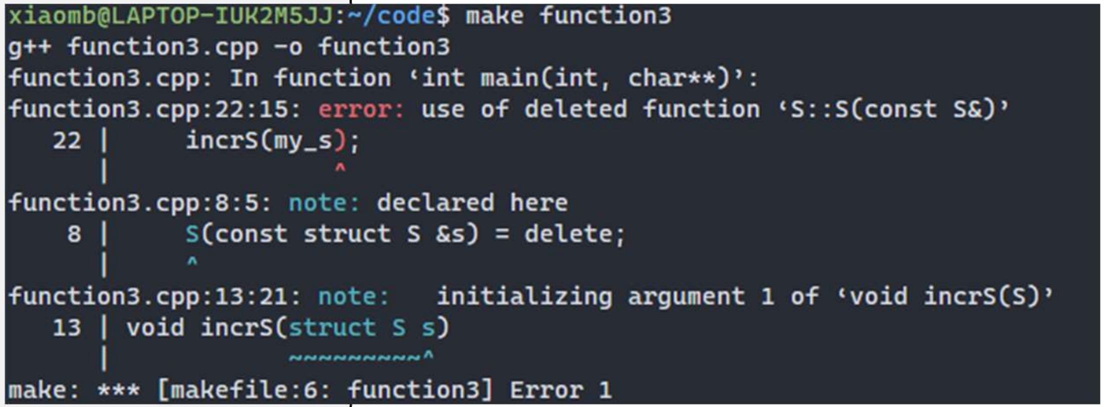


```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ make function2
g++ function2.cpp -o function2
xiaomb@LAPTOP-IUK2M5JJ:~/code$ ./function2
A is 100
xiaomb@LAPTOP-IUK2M5JJ:~/code$ _
```

复制构造体

- 函数调用时默认参数调用复制构造体

```
#include <iostream>

using namespace std;

struct S {
    int a;
    S(int _a) { a = _a; }
    S(const struct S &s) = delete;
};

void incrS(struct S s)
{
    s.a += 1;
}

int main(int argc, char *argv[])
{
    struct S my_s(100);
    cout << “my_s.a is “ << my_s.a << endl;
    incrS(my_s);

    return 0;
}
```

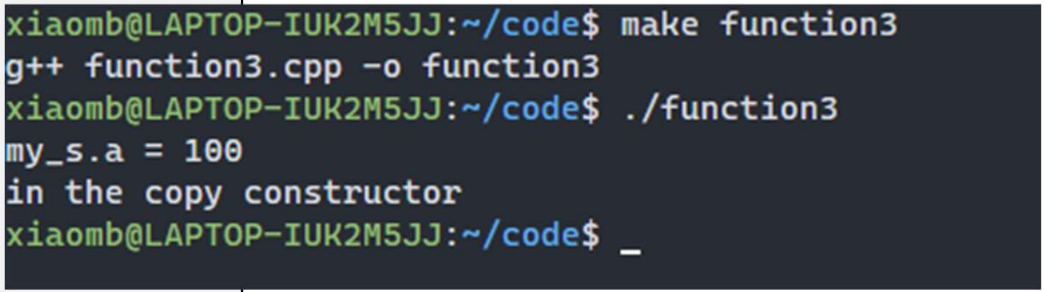


```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ make function3
g++ function3.cpp -o function3
function3.cpp: In function ‘int main(int, char**)’:
function3.cpp:22:15: error: use of deleted function ‘S::S(const S&)’
   22 |     incrS(my_s);
      |               ^
function3.cpp:8:5: note: declared here
    8 |     S(const struct S &s) = delete;
      |     ^
function3.cpp:13:21: note:   initializing argument 1 of ‘void incrS(S)’
   13 | void incrS(struct S s)
      |            ~~~~~~~~~^
make: *** [makefile:6: function3] Error 1
```

复制构造体

- 函数调用时默认参数调用复制构造体

```
#include <iostream>

using namespace std;

struct S {
  int a;
  S(int _a) { a = _a; }
  // S(const struct S &s) = delete;
  S(const struct S &s) {
    cout << “in the copy constructor” << endl;
    a = s.a;
  }
};

void incrS(struct S s)
{
  s.a += 1;
}

int main(int argc, char *argv[])
{
  struct S my_s(100);
  cout << “my_s.a is “ << my_s.a << endl;
  incrS(my_s);

  return 0;
}
```

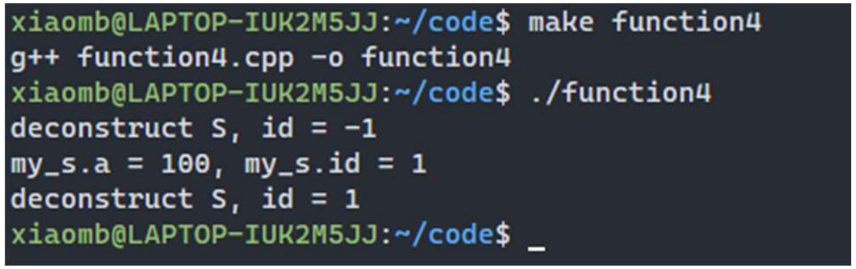


```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ make function3
g++ function3.cpp -o function3
xiaomb@LAPTOP-IUK2M5JJ:~/code$ ./function3
my_s.a = 100
in the copy constructor
xiaomb@LAPTOP-IUK2M5JJ:~/code$ _
```

析构函数

- 函数返回时调用参数的析构函数

```
#include <iostream>

using namespace std;

struct S {
  int a, id;
  S(int _a, int _id) : a(_a), id(_id) { }
  S(const struct S &s) { id = -1; }
  ~S() { cout << “desconstruct S, id = “
              << id << endl; }
};

void incrS(struct S s)
{
  s.a += 1;
}

int main(int argc, char *argv[])
{
  struct S my_s(100, 1);
  incrS(my_s);
  cout << “my_s.a = “ << my_s.a << “, “
       << “my_s.id = “ << my_s.id << endl;

  return 0;
}
```

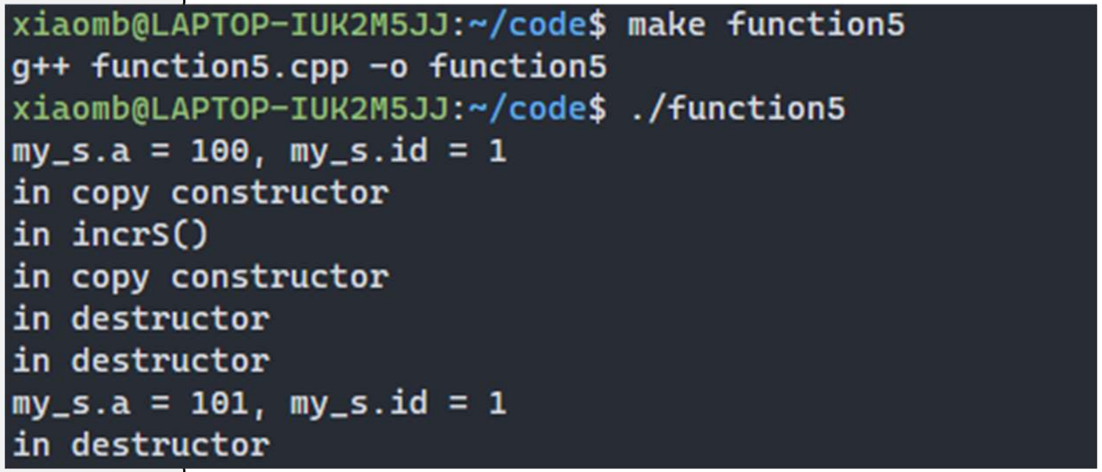


```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ make function4
g++ function4.cpp -o function4
xiaomb@LAPTOP-IUK2M5JJ:~/code$ ./function4
deconstruct S, id = -1
my_s.a = 100, my_s.id = 1
deconstruct S, id = 1
xiaomb@LAPTOP-IUK2M5JJ:~/code$ _
```

函数返回

- 通过赋值方法保留更改

```
#include <iostream>

using namespace std;

struct S {
  int a, id;
  S(int _a, int _id) : a(_a), id(_id) { }
  S(const struct S &s) {
    a = s.a;
    id = s.id;
    cout << “in copy constructor” << endl;
  }
  ~S() { cout << “in destructor“ << endl; }
};

struct S incrS(struct S s)
{
  s.a += 1;
  cout << “in incrS()” << endl;
  return s;
}

int main(int argc, char *argv[])
{
  struct S my_s(100, 1);
  cout << “my_s.a = “ << my_s.a << “, “
       << “my_s.id = “ << my_s.id << endl;
  my_s = incrS(my_s);
  cout << “my_s.a = “ << my_s.a << “, “
       << “my_s.id = “ << my_s.id << endl;
  return 0;
}
```



```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ make function5
g++ function5.cpp -o function5
xiaomb@LAPTOP-IUK2M5JJ:~/code$ ./function5
my_s.a = 100, my_s.id = 1
in copy constructor
in incrS()
in copy constructor
in destructor
in destructor
my_s.a = 101, my_s.id = 1
in destructor
```

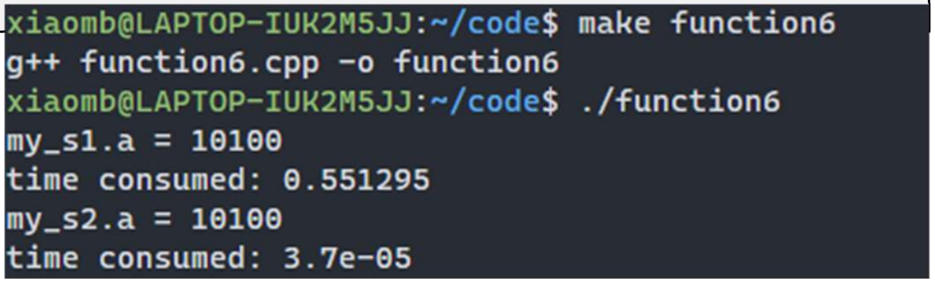
大量无意义的数据拷贝

引用参数

- 传入引用参数避免数据拷贝

```
#include <iostream>
#include <sys/time.h>

using namespace std;

double getRunningTime(struct timeval &begin,
                      struct timeval &end)
{
  return (end.tv_sec - begin.tv_sec) +
         (end.tv_usec - begin.tv_usec)*1e-6;
}
struct S {
  int a;
  int b[100000];
  S(int _a) : a(_a) { }
};

struct S incrS(struct S s){
  s.a += 1;
  return s;
}

void incrSr(struct S& s){
  s.a += 1;
}
```

```
#define LOOP_NR 10000
int main(int argc, char *argv[])
{
  struct timeval begin, end;

  struct S my_s1(100);
  gettimeofday(&begin, 0);
  for(int i = 0; i < LOOP_NR; i++)
    my_s1 = incrS(my_s1);
  gettimeofday(&end, 0);
  cout << “my_s1.a = ” << my_s1.a << endl;
  cout << “time consumed: ”
       << getRunningTime(begin, end) << endl;

  struct S my_ss(100);
  gettimeofday(&begin, 0);
  for(int i = 0; i < LOOP_NR; i++)
    my_s1 = incrSr(my_ss);
  gettimeofday(&end, 0);
  cout << “my_s2.a = ” << my_s2.a << endl;
  cout << “time consumed: ”
       << getRunningTime(begin, end) << endl;

  return 0;
}
```



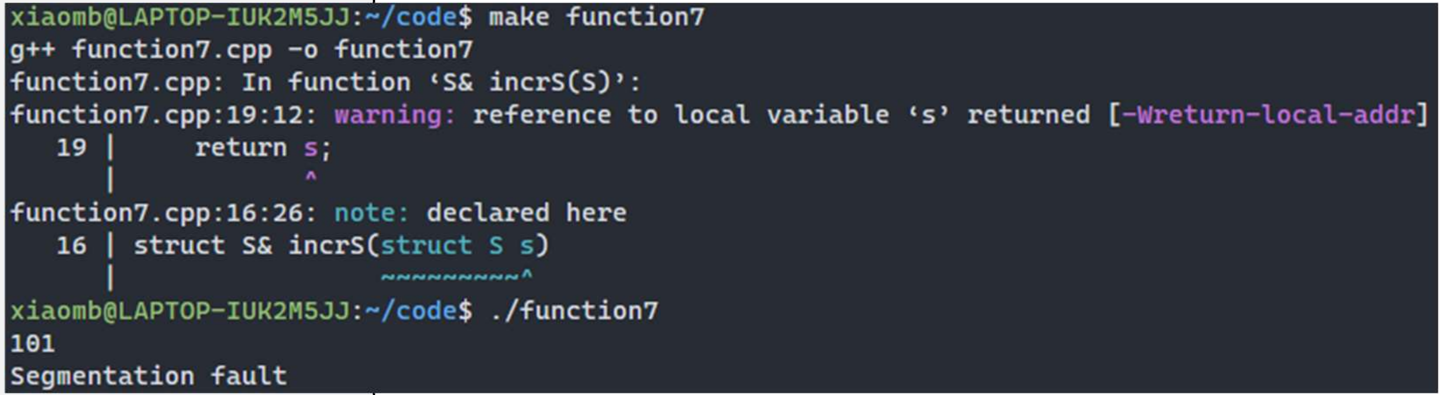
```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ make function6
g++ function6.cpp -o function6
xiaomb@LAPTOP-IUK2M5JJ:~/code$ ./function6
my_s1.a = 10100
time consumed: 0.551295
my_s2.a = 10100
time consumed: 3.7e-05
```

操纵传入的实参本身，没有拷贝构造和析构函数调用

引用返回

- 函数的返回值也可以是引用

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

struct S {
  int a;
  S(int _a) : a(_a) { }
};

struct S& incrSr(struct S& s)
{
  s.a += 1;
  return s;
}

struct S& incrS(struct S s)
{
  s.a += 1;
  return s;
}

int main(int argc, char *argv[])
{
  struct S my_s(100);
  my_s = incrSr(my_s);
  cout << my_s.a << endl;

  my_s = incrS(my_s);
  cout << my_s.a << endl;
  return 0;
}
```



```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ make function7
g++ function7.cpp -o function7
function7.cpp: In function ‘S& incrS(S)’:
function7.cpp:19:12: warning: reference to local variable ‘s’ returned [-Wreturn-local-addr]
   19 |     return s;
      |            ^
function7.cpp:16:26: note: declared here
   16 | struct S& incrS(struct S s)
      |                 ~~~~~~~~~^
xiaomb@LAPTOP-IUK2M5JJ:~/code$ ./function7
101
Segmentation fault
```

引用局部变量在退出其作用范围后会引起程序错误

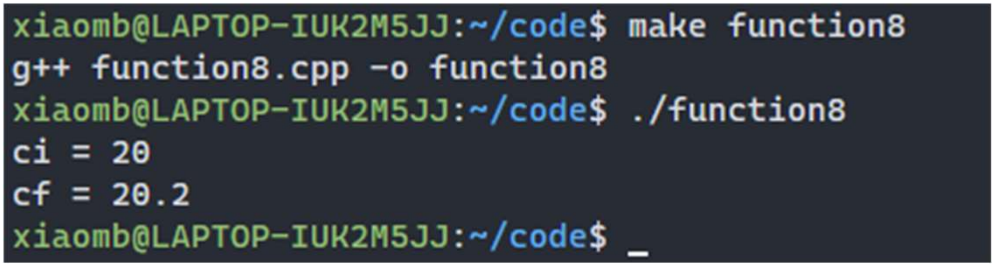
重载函数

- 编写程序时，往往出现算法相同，数据不同的情况
  - 加/减/乘/除
  - 排序算法
- 使用重载函数来简化代码编写
  - 编译器中，函数签名是由其参数个数和参数类型决定的
  - 同名函数含有不同参数为不同函数

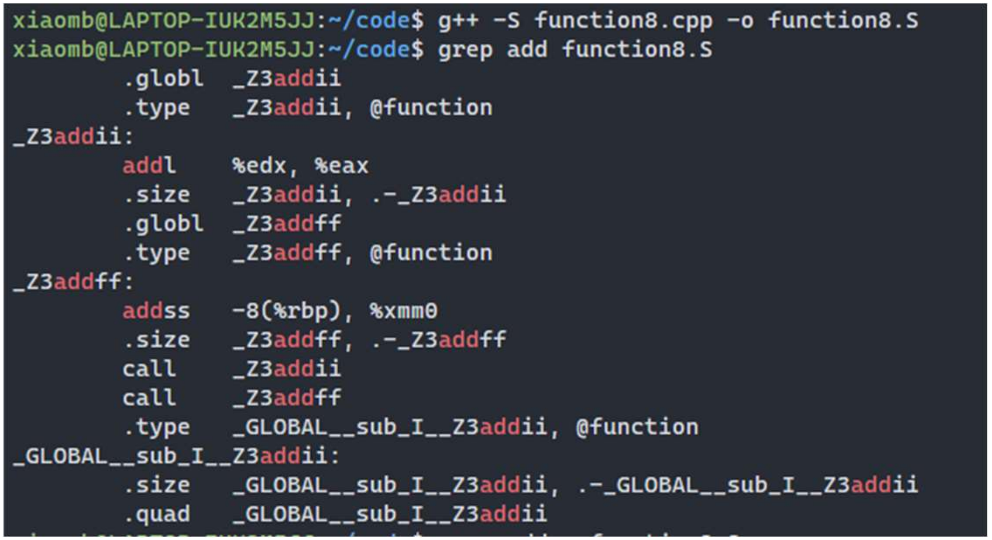
重载函数

- 重载函数让编译器决定应该调用哪个实现

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

int add(int a, int b){
  return a + b;
}

float add(float a, float b){
  return a + b;
}

int main(int argc, char *argv[])
{
  int ai = 10, bi = 10;
  int ci = add(ai, bi);
  cout << “ci = ” << ci << endl;

  float af = 10.1, bf = 10.1;
  float cf = add(af, bf);
  cout << “cf = ” << cf << endl;

  return 0;
}
```



```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ make function8
g++ function8.cpp -o function8
xiaomb@LAPTOP-IUK2M5JJ:~/code$ ./function8
ci = 20
cf = 20.2
xiaomb@LAPTOP-IUK2M5JJ:~/code$ _
```

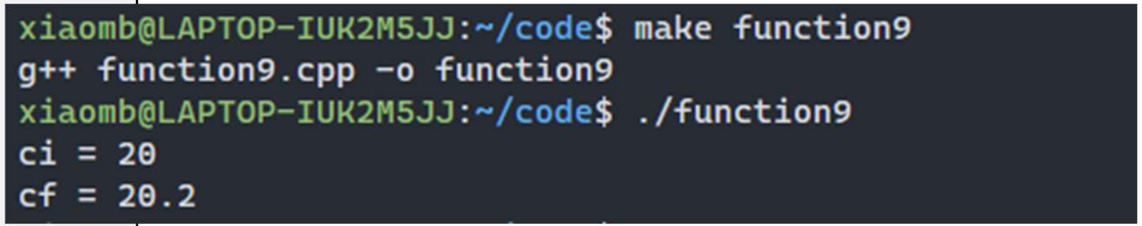


```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ g++ -S function8.cpp -o function8.S
xiaomb@LAPTOP-IUK2M5JJ:~/code$ grep add function8.S
        .globl  _Z3addii
        .type   _Z3addii, @function
_Z3addii:
        addl    %edx, %eax
        .size   _Z3addii, .-_Z3addii
        .globl  _Z3addff
        .type   _Z3addff, @function
_Z3addff:
        addss   -8(%rbp), %xmm0
        .size   _Z3addff, .-_Z3addff
        call    _Z3addii
        call    _Z3addff
        .type   _GLOBAL__sub_I__Z3addii, @function
_GLOBAL__sub_I__Z3addii:
        .size   _GLOBAL__sub_I__Z3addii, .-_GLOBAL__sub_I__Z3addii
        .quad   _GLOBAL__sub_I__Z3addii
```

模板函数

- 模板函数进一步简化程序的编写

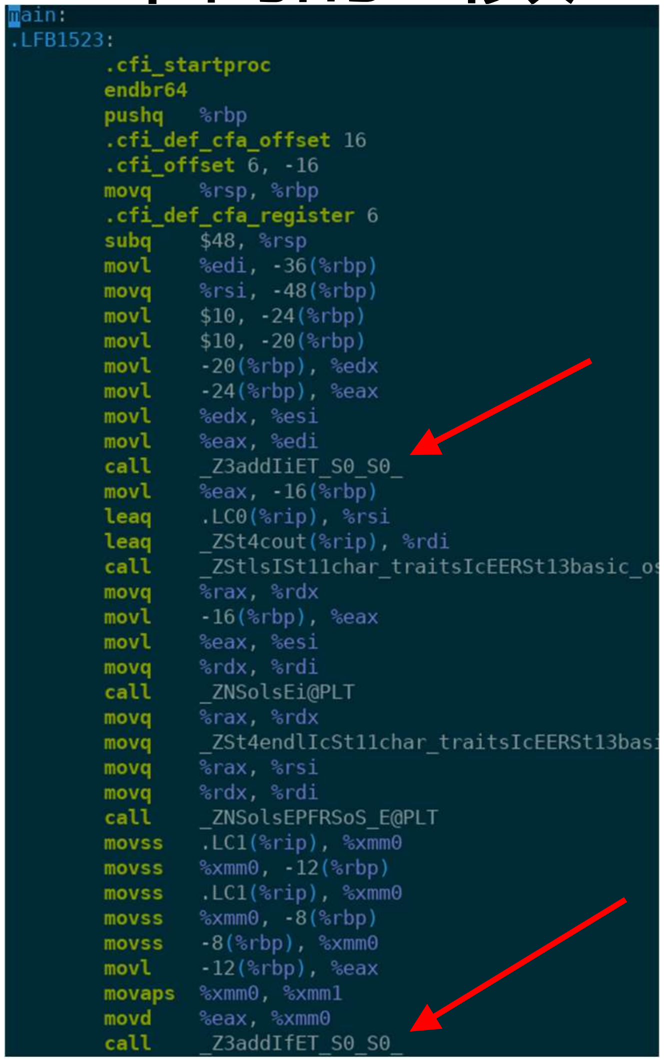
```
#include <iostream>
using namespace std;

template<typename T>
T add(T a, T b){
  return a + b;
}

int main(int argc, char* argv[])
{
  int ai = 10, bi = 10;
  int ci = add(ai, bi);
  cout << “ci = “ << ci << endl;

  float af = 10.1, bf = 10.1;
  float cf = add(af, bf);
  cout << “cf = “ << cf << endl;

  return 0;
}
```



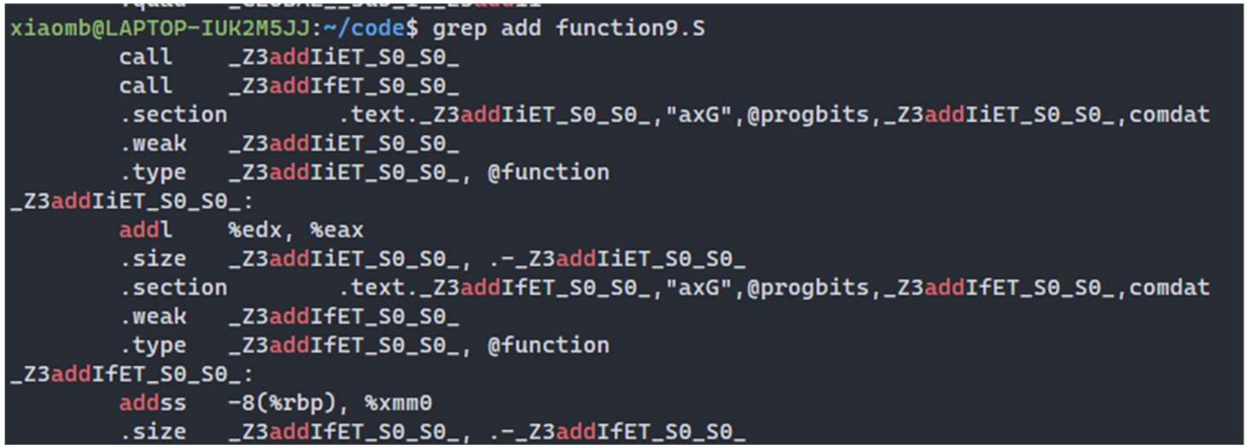
```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ make function9
g++ function9.cpp -o function9
xiaomb@LAPTOP-IUK2M5JJ:~/code$ ./function9
ci = 20
cf = 20.2
```

模板函数

- 模板函数由编译器自动生成代码，实际还是不同的函数



```
main:
.LFB1523:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $48, %rsp
        movl    %edi, -36(%rbp)
        movq    %rsi, -48(%rbp)
        movl    $10, -24(%rbp)
        movl    $10, -20(%rbp)
        movl    -20(%rbp), %edx
        movl    -24(%rbp), %eax
        movl    %edx, %esi
        movl    %eax, %edi
        call    _Z3addIiET_S0_S0_
        movl    %eax, -16(%rbp)
        leaq    .LC0(%rip), %rsi
        leaq    _ZSt4cout(%rip), %rdi
        call    _ZStlsISt11char_traitsIcEERSt13basic_os
        movq    %rax, %rdx
        movl    -16(%rbp), %eax
        movl    %eax, %esi
        movq    %rdx, %rdi
        call    _ZNSolsEi@PLT
        movq    %rax, %rdx
        movq    _ZSt4endlIcSt11char_traitsIcEERSt13bas
        movq    %rax, %rsi
        movq    %rdx, %rdi
        call    _ZNSolsEPFRSoS_E@PLT
        movss   .LC1(%rip), %xmm0
        movss   %xmm0, -12(%rbp)
        movss   .LC1(%rip), %xmm0
        movss   %xmm0, -8(%rbp)
        movss   -8(%rbp), %xmm0
        movl    -12(%rbp), %eax
        movaps  %xmm0, %xmm1
        movd    %eax, %xmm0
        call    _Z3addIfET_S0_S0_
```

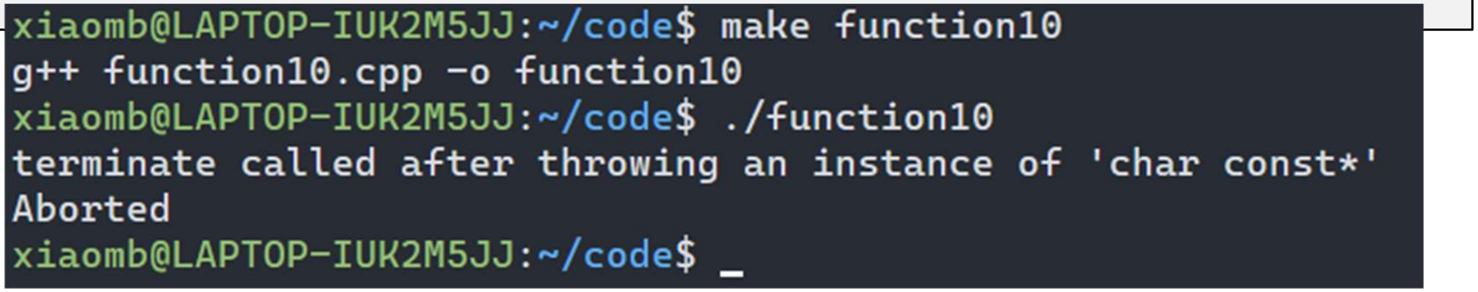


```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ grep add function9.S
        call    _Z3addIiET_S0_S0_
        call    _Z3addIfET_S0_S0_
        .section        .text._Z3addIiET_S0_S0_,"axG",@progbits,_Z3addIiET_S0_S0_,comdat
        .weak   _Z3addIiET_S0_S0_
        .type   _Z3addIiET_S0_S0_, @function
_Z3addIiET_S0_S0_:
        addl    %edx, %eax
        .size   _Z3addIiET_S0_S0_, .-_Z3addIiET_S0_S0_
        .section        .text._Z3addIfET_S0_S0_,"axG",@progbits,_Z3addIfET_S0_S0_,comdat
        .weak   _Z3addIfET_S0_S0_
        .type   _Z3addIfET_S0_S0_, @function
_Z3addIfET_S0_S0_:
        addss   -8(%rbp), %xmm0
        .size   _Z3addIfET_S0_S0_, .-_Z3addIfET_S0_S0_
```

异常

- 异常会终止程序执行并给出抛出的异常对象的类型信息

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

int add(int a, int b){
  if(a <= 0 || b <= 0)
    throw “All parameters should be greater than 0”;
  return a + b;
}

int main(int argc, char* argv[])
{
  cout << “10 + -1 = “ << add(10, -1) << endl;
  return 0;
}
```

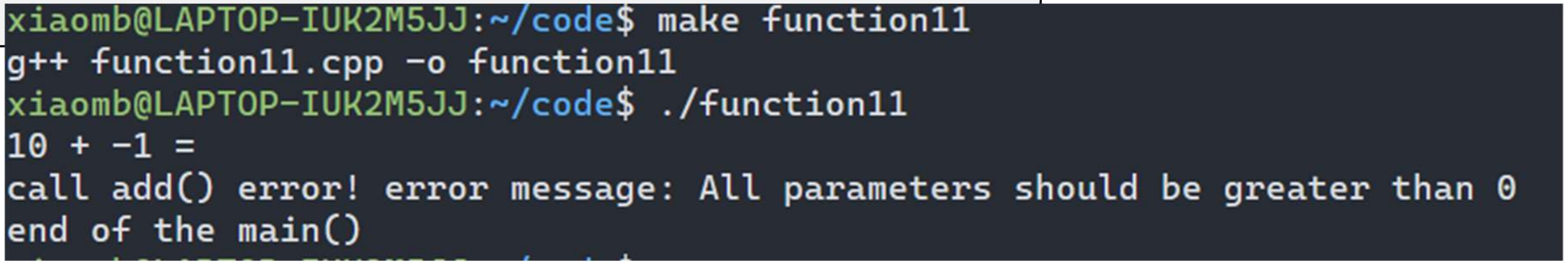


```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ make function10
g++ function10.cpp -o function10
xiaomb@LAPTOP-IUK2M5JJ:~/code$ ./function10
terminate called after throwing an instance of 'char const*'
Aborted
xiaomb@LAPTOP-IUK2M5JJ:~/code$ _
```

异常

- 异常也可以被捕捉后让程序继续执行

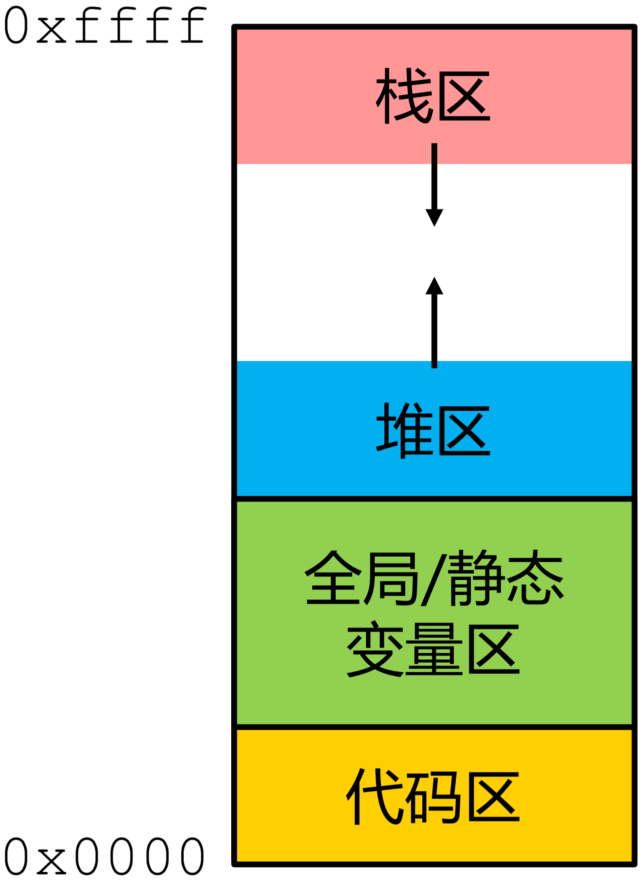
```
#include <iostream>
using namespace std;

int add(int a, int b){
  if(a <= 0 || b <= 0)
    throw “All parameters should be greater than 0”;
  return a + b;
}

int main(int argc, char* argv[])
{
  try {
    cout << “10 + -1 = “ << add(10, -1) << endl;
  }catch(const char *e){
    cout << “\ncall add() error! error message: “
         << e << endl;
  }
  cout << “end of the main()” << endl;
  return 0;
}
```



```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ make function11
g++ function11.cpp -o function11
xiaomb@LAPTOP-IUK2M5JJ:~/code$ ./function11
10 + -1 =
call add() error! error message: All parameters should be greater than 0
end of the main()
```

内存中的C++程序

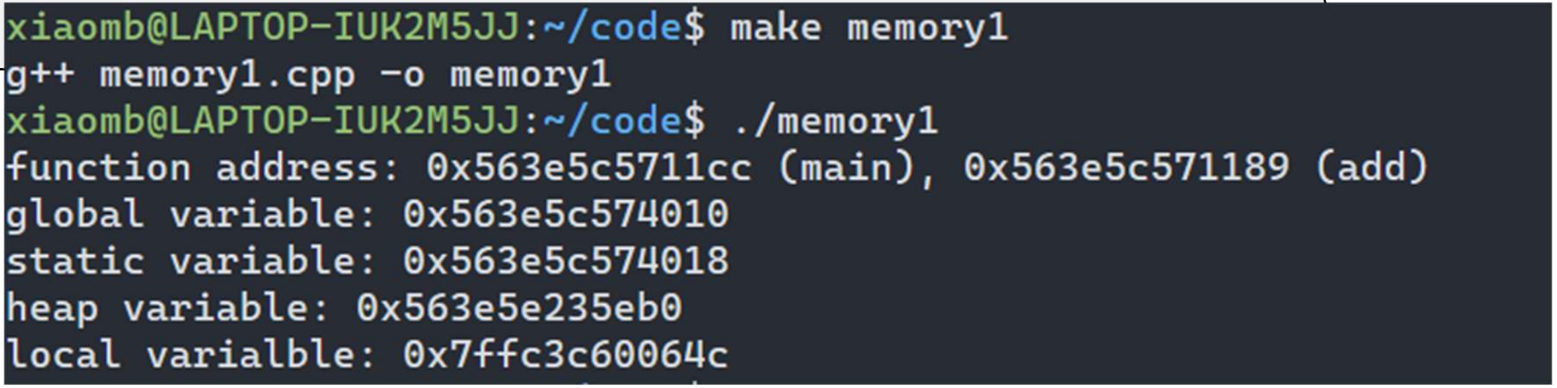
- C/C++程序定义了4个内存区间:
  - 代码区
  - 全局变量与静态变量区
  - 局部变量区(栈区)
  - 动态存储区，即堆(heap)区



内存中的C++程序

- 示例程序

```
#include <cstdio>

int global_var = 100;

int add(int a, int b){
  static int static_var = 0;
  static_var++;
  printf(“static variable: %p\n”, &static_var);
  return a + b;
}

int main(int argc, char* argv[])
{
  int local_var = 101;
  int *heap_var = new int;

  printf(“function address: %p (main), %p (add)\n”, main, add);
  printf(“global variable: %p\n”, &global_var);
  add(0, 0);
  printf(“heap variable: %p\n”, heap_var);
  printf(“local varialble: %p\n”, local_var);
  return 0;
}
```



```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ make memory1
g++ memory1.cpp -o memory1
xiaomb@LAPTOP-IUK2M5JJ:~/code$ ./memory1
function address: 0x563e5c5711cc (main), 0x563e5c571189 (add)
global variable: 0x563e5c574010
static variable: 0x563e5c574018
heap variable: 0x563e5e235eb0
local varialble: 0x7ffc3c60064c
```

静态储存与动态储存

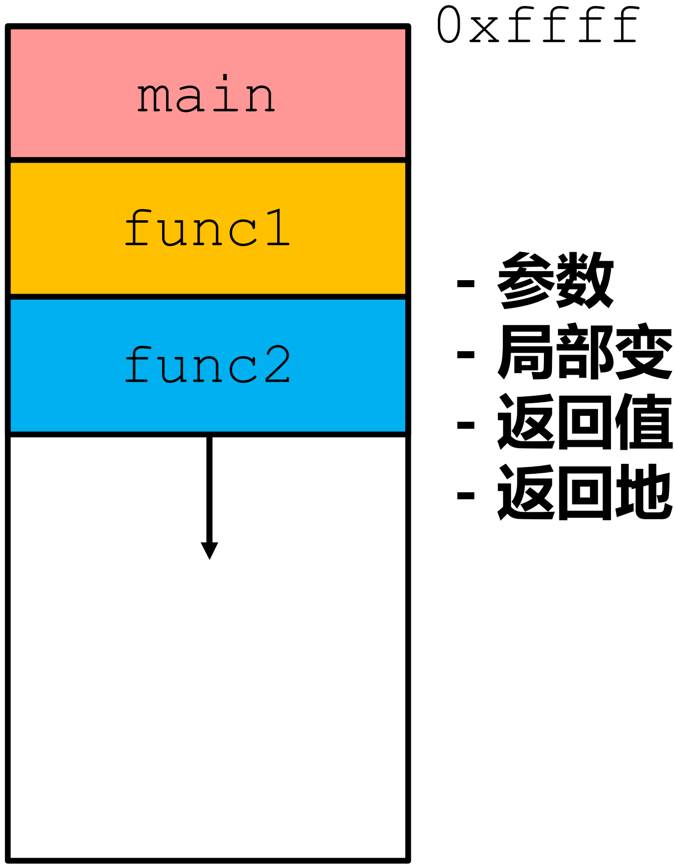
- **静态存储**：编译器在**编译时**知道所需内存空间的大小
  - 直接定义

- **动态存储：**在**程序运行时**才能确定所需内存空间
  - 堆中分配：`malloc(), calloc(), new`
  - 栈中分配：`alloca(), variable-length array (C99 standard)`

栈帧

- 栈区由栈帧组成

```
float func2(float arg){
  return arg;
}

int func1(int arg) {
  return (int) func2((float) arg);
}

int main(int argc, char* argv[])
{
  int ret = func1(100);
  return ret;
}
```



- 参数
- 局部变量
- 返回值
- 返回地址

堆区与栈区内存的使用

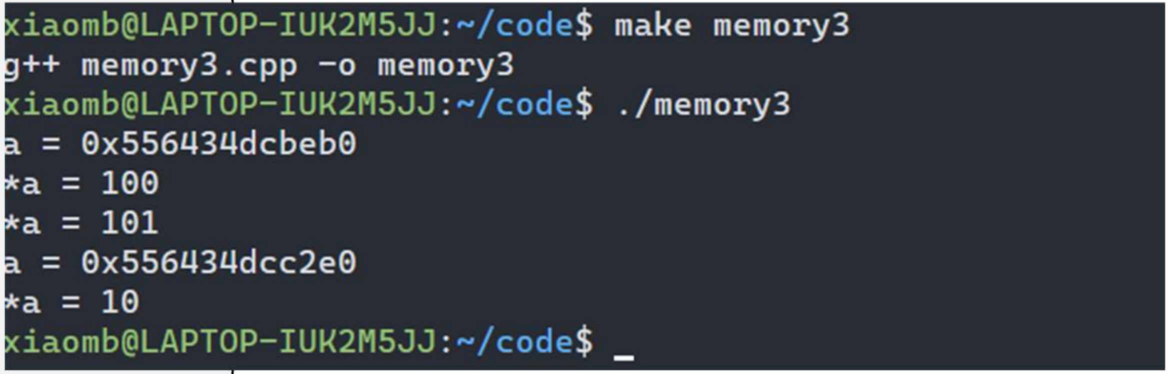
- **堆区**内存：需显式释放，可在不同函数中使用

- **栈区**内存：不用显式释放，只能在当前函数中使用

操作符 new

- 用于在堆区分配内存

```
#include <cstdio>

int main(int argc, char *argv[])
{
  int *a = new int;
  *a = 100;
  printf(“a = %p\n”, a);
  printf(“*a = %d\n”, *a);

  a[0] = 101;
  printf(“*a = %d\n”, *a);

  a = new int(10);
  printf(“a = %p\n”, a);
  printf(“*a = %d\n”, *a);
  return 0;
}
```

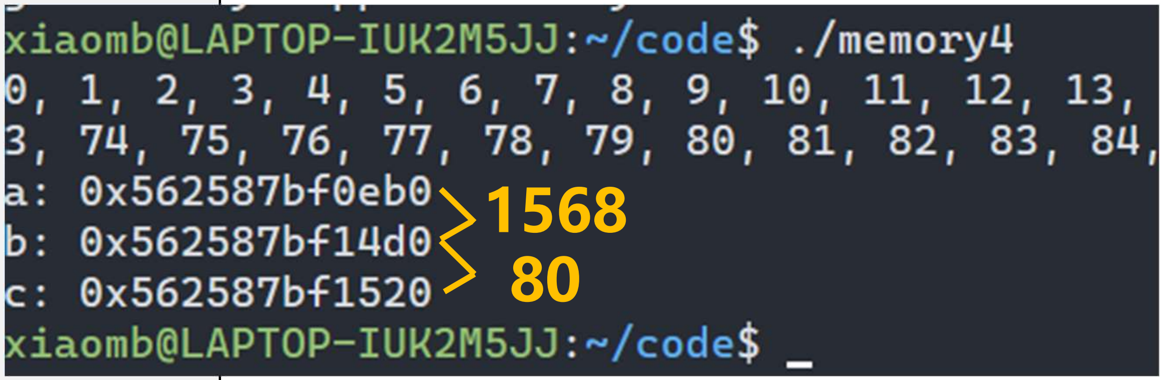


```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ make memory3
g++ memory3.cpp -o memory3
xiaomb@LAPTOP-IUK2M5JJ:~/code$ ./memory3
a = 0x556434dcbeb0
*a = 100
*a = 101
a = 0x556434dcc2e0
*a = 10
xiaomb@LAPTOP-IUK2M5JJ:~/code$ _
```

一维数组

- 在堆区用new动态分配内存

```
#include <cstdio>

int main(int argc, char *argv[])
{
  int n = 128;
  int *a = new int[n];
  for(int i = 0; i < n; i++)
    a[i] = i;

  for(int i = 0; i < n; i++)
    printf(“%d, ”, a[i]);
  printf(“\n”);

  printf(“a: %p\n”, a);
  int *b = new int[16];
  printf(“b: %p\n”, b);
  int *c = new int;
  printf(“c: %p\n”, c);
  return 0;
}
```

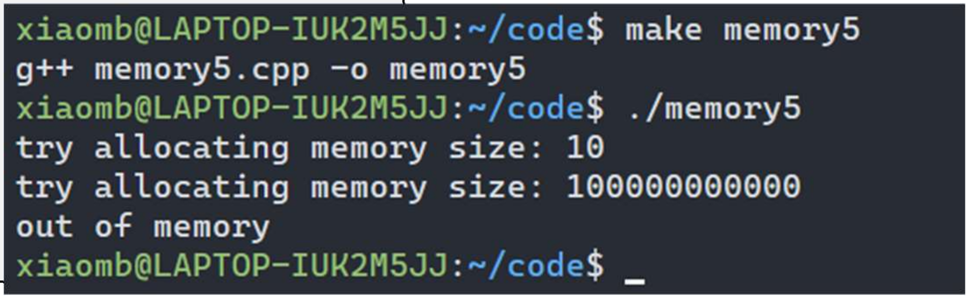


```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ ./memory4
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
3, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
a: 0x562587bf0eb0
b: 0x562587bf14d0
c: 0x562587bf1520
xiaomb@LAPTOP-IUK2M5JJ:~/code$ _
```

异常处理

- 所请求的内存空间过大时?
  - 异常对象：bad_alloc

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

int main(int argc, char *argv[])
{
  int *a;
  unsigned long long len = 10;
  cout << “try allocating memory size: << len << endl;
  try { a = new int[len]; }
  catch(bad_alloc) {
    cerr << “out of memory” << endl;
  }

  len = 100000000000; // ~ 100 GB
  cout << “try allocating memory size: << len << endl;
  try { a = new int[len]; }
  catch(bad_alloc) {
    cerr << “out of memory” << endl;
  }

  return 0;
}
```



```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ make memory5
g++ memory5.cpp -o memory5
xiaomb@LAPTOP-IUK2M5JJ:~/code$ ./memory5
try allocating memory size: 10
try allocating memory size: 100000000000
out of memory
xiaomb@LAPTOP-IUK2M5JJ:~/code$ _
```

操作符 delete

- 使用delete及时释放内存，否则易出现内存泄露

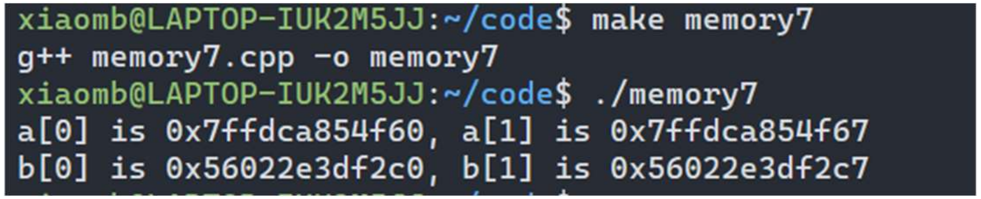
```
#include <iostream>
using namespace std;

int main(int argc, char *argv[])
{
  // Initialization …

  // Routines
  while(true){
    // allocate resources for this round of operation
    int *a = new int(10);
    int *b = new int[16];

    // some operations …

    // memory leak without following deallocations
    delete [] b;
    delete a;
  }

  return 0;
}
```

二维数组

- 静态分配二维数组
- 动态分配二维数组
  - 列数已知

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

int main(int argc, char *argv[])
{
  char a[5][7];
  printf(“a[0] is %p, a[1] is %p\n”, a[0], a[1]);

  int n = 5;
  char (*b)[7];
  b = new char[n][7];
  printf(“b[0] is %p, b[1] is %p\n”, b[0], b[1]);

  return 0;
}
```

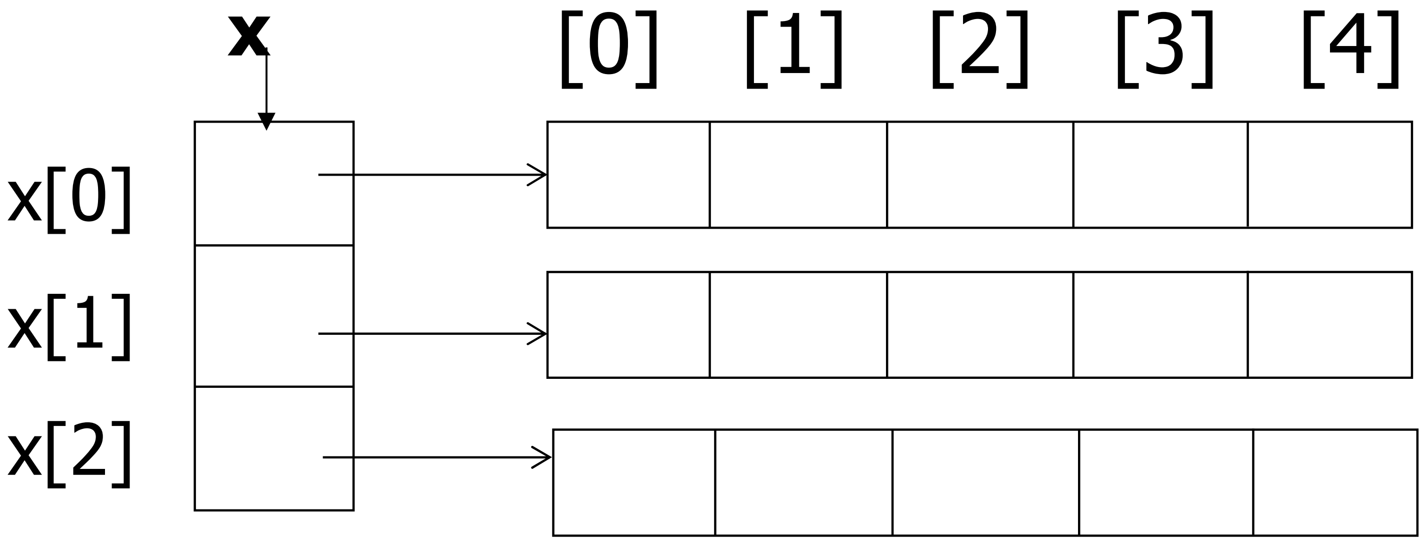
均为大小为35的连续内存空间



```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ make memory7
g++ memory7.cpp -o memory7
xiaomb@LAPTOP-IUK2M5JJ:~/code$ ./memory7
a[0] is 0x7ffdca854f60, a[1] is 0x7ffdca854f67
b[0] is 0x56022e3df2c0, b[1] is 0x56022e3df2c7
```

二维数组

- 编译时数组的行数与列数均未知
  - 分配非连续空间
  - `char **x`

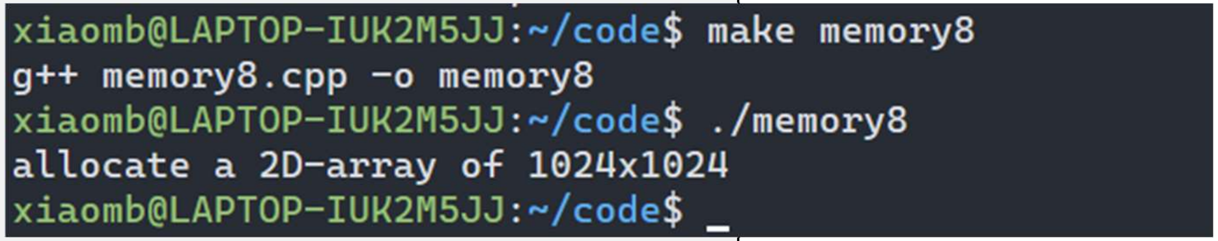


内存空间不连续

二维数组

- 为任意类型数据设计二维数组创建函数

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

template<class T>
bool make2dArray(T **&x, int NOfRow, int NOfCol)
{
  try{
    x = new T*[NOfRow];
    for(int i = 0; i < NOfRow; i++)
      x[i] = new T[NOfCol];
    return true;
  }catch(bad_alloc){
    return false;
  }
}
int main(int argc, char *argv[])
{
  float **float2DArray
  bool ret;

  ret = make2dArray(float2DArray, 1024, 1024);
  if(ret) cout << “allocate a 2D-array of 1024x1024” << endl;
  else cerr << “fail to allocate the 2D-array” << endl;
  return 0;
}
```



```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ make memory8
g++ memory8.cpp -o memory8
xiaomb@LAPTOP-IUK2M5JJ:~/code$ ./memory8
allocate a 2D-array of 1024x1024
xiaomb@LAPTOP-IUK2M5JJ:~/code$ _
```

二维数组

- 为创建的二维数组释放空间

```cpp
template<class T>
bool delete2dArray(T **&x, int NOfRow)
{
   for(int i = 0; i < NOfRow; i++)
      delete [] x[i];
   delete [] x;
   x = nullptr;
}

int main(int argc, char *argv[])
{
   float **float2DArray
   bool ret;

   ret = make2dArray(float2DArray, 1024, 1024);

   delete2dArray(float2DArray, 1024);
   return 0;
}
```

- 逆序释放
- 置空指针

自有数据类型（类）

- Currency类：$2.33，-$1.23
  - 类（class）与对象（object）
  - 成员与方法
    - 成员：符号（枚举），美元（长整型），美分（整型）

```
enum signType {_plus, _minus};

class Currency {
private:
   signType sign;
   unsigned long dollars;
   unsigned int cents;
};
```

Currency类方法

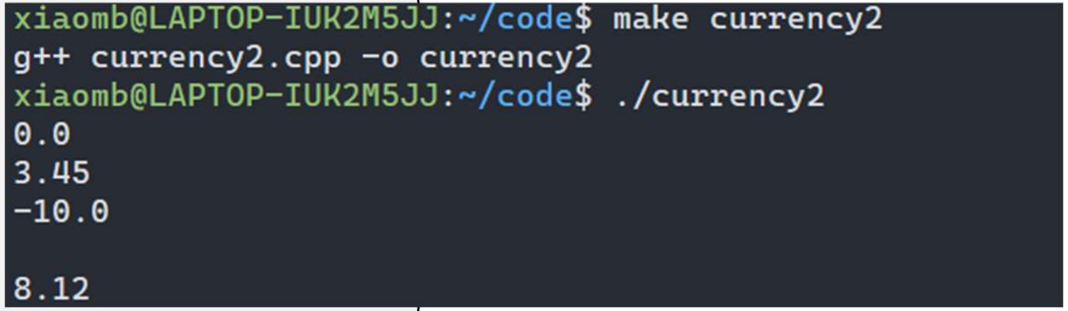
- Currency类：$2.33，-$1.23
  - 方法
    - 构造/析构
    - 设置成员值
    - 读取各成员值
    - 两个对象相加
    - 增加成员的值
    - 输出

```
enum signType {_plus, _minus};

class Currency {
public:
  Currency(signType theSign = _plus,
           unsigned long theDollars = 0,
           unsigned int theCents = 0);
  ~Currency() { }
  bool setValue(signType, unsigned long, unsigned int);
  bool setValue(double);
  signType getSign() const { return sign; }
  unsigned long getDollars() const { return dollars; }
  unsigned int getCents() const { return cents; }
  Currency add(const Currency&) const;
  Currency& increment(const Currency&);
  void output() const;
private:
  signType sign;
  unsigned long dollars;
  unsigned int cents;
};
```

方法实现

构造函数与输出函数

```
Currency::Currency(signType theSign,
                   unsigned long theDollars,
                   unsigned int theCents)
{
  sign = theSign;
  dollars = theDollars;
  cents = theCents;
}

void Currency::output() const{
  cout << (sign == _plus ? “” : “-”)
       << dollars << “.”
       << cents << endl;
}

int main(int argc, char *argv[])
{
  Currency f, g(_plus, 3, 45), h(_minus, 10);
  f.output();
  g.output();
  h.output();

  cout << endl;
  Currency *m = new Currency(_plus, 8, 12);
  m->output();

  return 0;
}
```

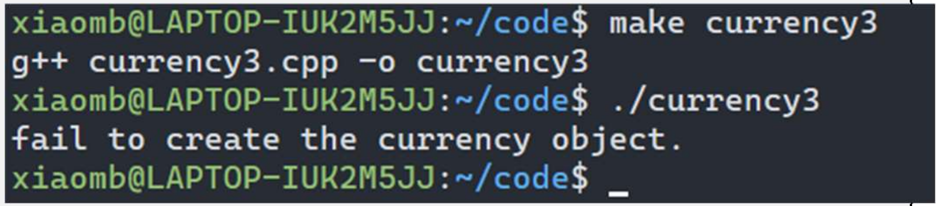


```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ make currency2
g++ currency2.cpp -o currency2
xiaomb@LAPTOP-IUK2M5JJ:~/code$ ./currency2
0.0
3.45
-10.0

8.12
```

处理非法输入

- 输入的美分可能大于有效范围，用异常处理

```
#include <cstring>
class illegalParameterValue
{
public:
  illegalParameterValue(const char *s) { strncpy(errMsg, s, 128); }
private:
  char errMsg[128];
};

Currency::Currency(signType theSign,
                   unsigned long theDollars,
                   unsigned int theCents)
{
  if(theCents > 99)
    throw illegalParameterValue("Cents should be <= 99");
  sign = theSign;
  dollars = theDollars;
  cents = theCents;
}

int main(int argc, char *argv[])
{
  try{ Currency f(_plus, 100, 100); }
  catch(illegalParameterValue e) {
    cout << "fail to create the currency object." << endl;
  }
  return 0;
}
```

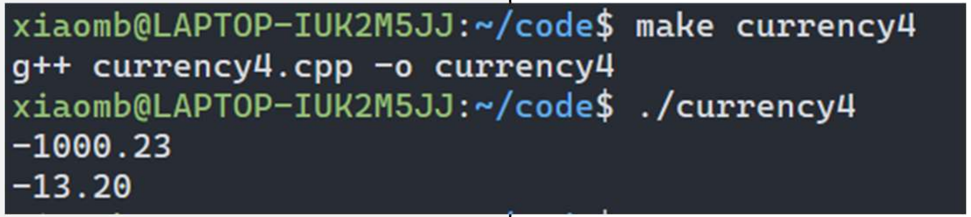


```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ make currency3
g++ currency3.cpp -o currency3
xiaomb@LAPTOP-IUK2M5JJ:~/code$ ./currency3
fail to create the currency object.
xiaomb@LAPTOP-IUK2M5JJ:~/code$ _
```

setValue

- 接收两种类型的输入

```
bool Currency::setValue(signType, unsigned long, unsigned int)
{  ……  }

bool Currency::setValue(double theAmount)
{
  if(theAmount < 0){
    sign = _minus;
    theAmount = -theAmount;
  }else
    sign = _plus;
  dollars = (unsigned long) theAmount;
  cents = (unsigned int) ((theAmount + 0.001 - dollars) * 100);
  return true;
}

int main(int argc, char *argv[])
{
  Currency f, g;
  f.setValue(_minus, 1000, 23);
  g.setValue(-13.2);

  f.output();
  g.output();
  return 0;
}
```



```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ make currency4
g++ currency4.cpp -o currency4
xiaomb@LAPTOP-IUK2M5JJ:~/code$ ./currency4
-1000.23
-13.20
```

add

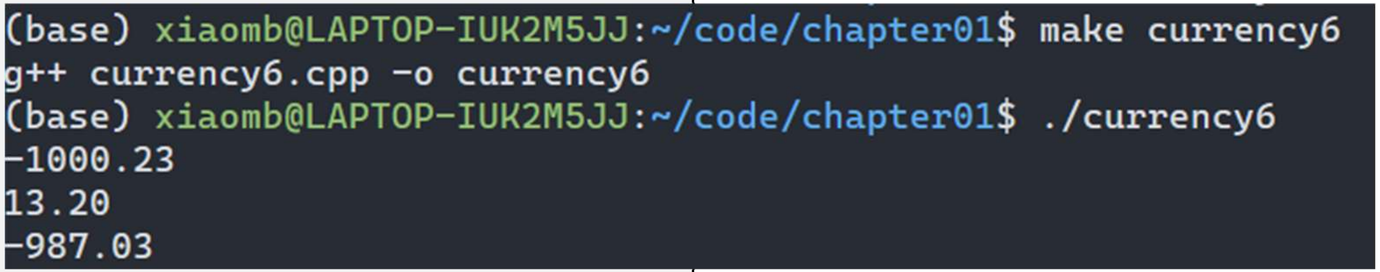
- 求两个Currency对象的和（新的对象）

```
Currency Currency::add(const Currency& other) const
{
  long a1, a2, a3;
  Currency result;

  a1 = dollars * 100 + cents;
  if(sign == _minus) a1 = -a1;

  a2 = other.dollars * 100 + other.cents;
  if(other.sign == _minus) a2 = -a2;

  a3 = a1 + a2;

  if(a3 < 0) {
    result.sign = _minus;
    a3 = -a3;
  }else
    result.sign = _plus;

  result.dollars = a3 / 100;
  result.cents = a3 - result.dollars * 100;
  return result;
}
```

increment

- 更新Currency对象本身

```
Currency &Currency::increment(const Currency &other)
{
  *this = add(other);
  return *this;
}

int main(int argc, char *argv[])
{
  Currency f,g;
  f.setValue(_minus, 1000, 23);
  g.setValue(13.2);

  f.output();
  g.output();

  f.increment(g);
  f.output();

  return 0;
}
```



```
(base) xiaomb@LAPTOP-IUK2M5JJ:~/code/chapter01$ make currency6
g++ currency6.cpp -o currency6
(base) xiaomb@LAPTOP-IUK2M5JJ:~/code/chapter01$ ./currency6
-1000.23
13.20
-987.03
```

类currency的应用

- Currency可以实现在头文件中

currency.h

```
#ifndef CURRENCY_H
#define CURRENCY_H

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

enum signType {_plus, _minus};

class Currency {
public:
  ……
};

……

#endif
```

main.cpp

```
#include "currency.h"

int main(int argc, char *argv[])
{
  Currency g, h(_plus, 10, 0), i, j;

  ……

  return 0;
}
```

不同的实现

- 已经有许多程序使用Currency类

- 修改Currency类，使其频率最高的add运行的更快

- 不影响代码的正确性

不同的实现

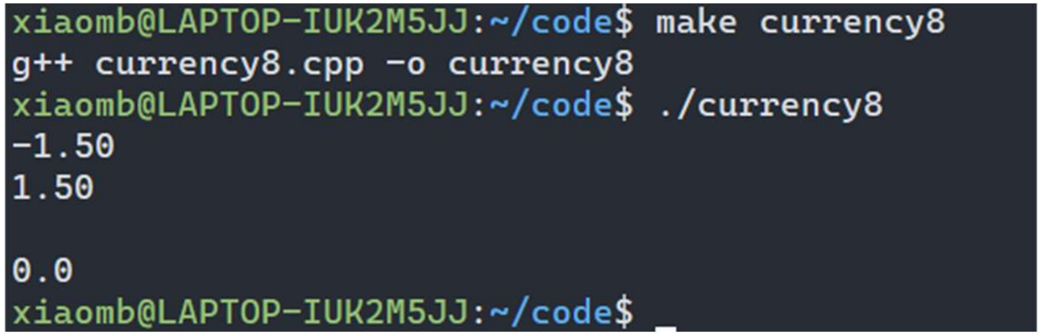
```
class Currency {
public:
  Currency(signType theSign = _plus,
           unsigned long theDollars = 0,
           unsigned int theCents = 0);
  ~Currency() { }
  bool setValue(signType,
                unsigned long,
                unsigned int);
  bool setValue(double);
  signType getSign() const {
    if(amount < 0) return _minus;
    else return _plus;
  }
  unsigned long getDollars() const {
    if(amount < 0) return (-amount) / 100;
    else return amount / 100;
  }
  unsigned int getCents() const {
    if(amount < 0)
      return -amount - getDollars() * 100;
    else
      return amount - getDollars() * 100;
  }
  Currency add(const Currency&) const;
  Currency& increment(const Currency& other) {
    amount += other.amount;
    return *this;
  }
  void output() const;
private:
  long amount;
};
```

类的接口不变

```
Currency::Currency(signType theSign,
                   unsigned long theDollars,
                   unsigned int theCents)
{
  if(theCents > 99)
    throw illegalParameterValue("Cents should be < 99");
  amount = theDollars * 100 + theCents;
  amount = theSign == _minus ? -amount : amount;
}

void Currency::output() const
{
  cout << (amount >= 0 ? "" : "-")
       << getDollars() << "."
       << getCents() << endl;
}
```

不同的实现

```
int main(int argc, char *argv[])
{
    Currency f(_minus, 1, 50), g(_plus, 1, 50);
    f.output();
    g.output();

    cout << endl;
    f.increment(g);
    f.output();
}
```



```
xiaomb@LAPTOP-IUK2M5JJ:~/code$ make currency8
g++ currency8.cpp -o currency8
xiaomb@LAPTOP-IUK2M5JJ:~/code$ ./currency8
-1.50
1.50

0.0
xiaomb@LAPTOP-IUK2M5JJ:~/code$ _
```

操作符重载

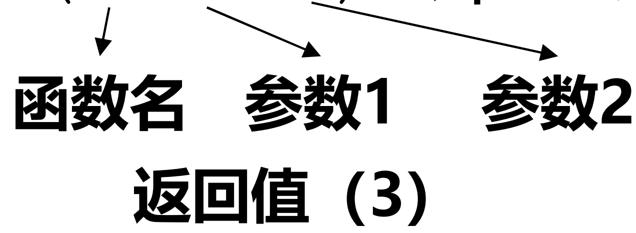
- 更简洁自然的方式表示add与increment

```
h1 = f1.add(g1);
f1.increment(g1);
```

⇒

```
h1 = f1 + g1;
f1 += g1;
```

- 操作符即函数
  - Lisp: (+ 1 2) 即 C/C++: 1 + 2



函数名 参数1 参数2

返回值（3）

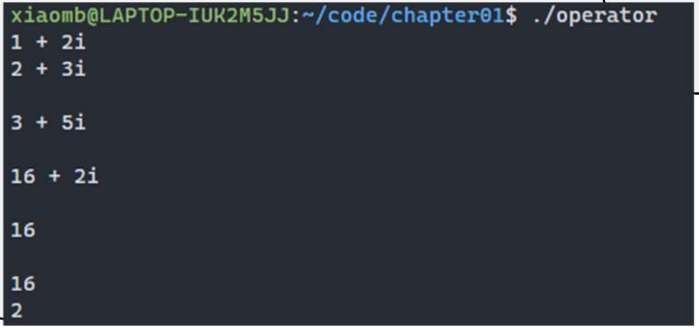
- C/C++可以重载类之间的操作符

操作符重载

- 可重载的操作符
  - 运算符/类型转换/函数调用

```
class Complex {
public:
    Complex(double r = 0.0, double i = 0.0) : real(r), image(i) {}
    ~Complex() {}
    void output() { cout << real << " + " << image << "i" << endl; }

    Complex operator+(Complex &other) const {
        Complex result;
        result.real = real + other.real;
        result.image = image + other.image;
        return result;
    }

    Complex &operator+=(Complex &other) {
        real += other.real;
        image += other.image;
        return *this;
    }

    Complex &operator+=(int other) {
        real += other;
        return *this;
    }
    operator double() { return real; }
    double operator()(int which) {
        if(which == 0)
            return real;
        else
            return image;
    }
private:
    double real;
    double image;
};
```

```
int main(int argc, char *argv[])
{
    Complex a(1.0, 2.0), b(2.0, 3.0);
    a.output();
    b.output();

    cout << endl;
    (a + b).output();

    cout << endl;
    a += 15;
    a.output();

    cout << endl;
    cout << (double) a << endl;

    cout << endl;
    cout << a(0) << endl;
    cout << a(1) << endl;

    return 0;
```



```
xiaomb@LAPTOP-IUK2M5JJ:~/code/chapter01$ ./operator
1 + 2i
2 + 3i

3 + 5i

16 + 2i

16

16
2
```

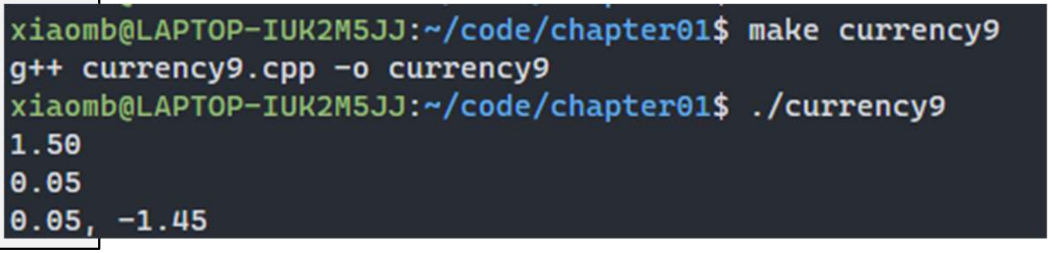
Currency中操作符重载

- 重载+=， <<两个操作符

```
  Currency &operator+=(const Currency& other) {
    amount += other.amount;
    return *this;
  }
```

```
std::string Currency::getString() const
{
  char strbuf[128];
  snprintf(strbuf, 128, "%s%lu.%.2u",
           amount >= 0 ? "" : "-", getDollars(), getCents());
  return std::string(strbuf);
}

ostream& operator<<(ostream &out, const Currency &c)
{
  out << c.getString();
  return out;
}
```

```
int main (int argc, char *argv[])
{
  Currency f1(_plus, 1, 50), g1(_minus, 1, 45);
  f1.output();
  f1 += g1;
  f1.output();

  cout << f1 << ", " << g1 << endl;

  return 0;
}
```

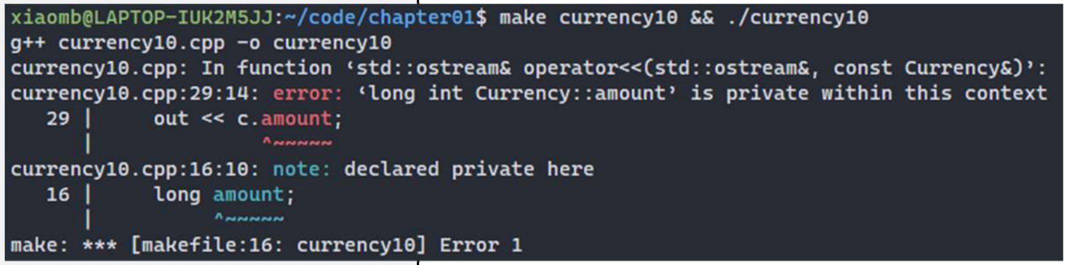


```
xiaomb@LAPTOP-IUK2M5JJ:~/code/chapter01$ make currency9
g++ currency9.cpp -o currency9
xiaomb@LAPTOP-IUK2M5JJ:~/code/chapter01$ ./currency9
1.50
0.05
0.05, -1.45
```

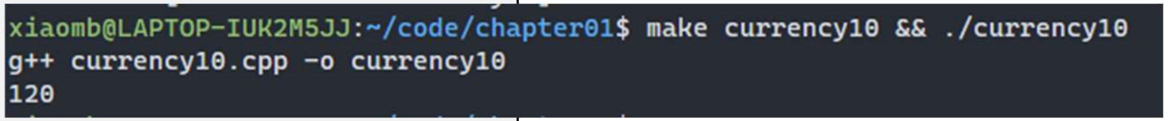
友元和保护性类成员

- 可以设置某些函数为友元来允许其访问内部变量

```
class Currency {
    // commenting this will cause compiling errors
    friend ostream& operator<<(ostream&, const Currency&);
public:
    Currency(signType theSign = _plus,
             unsigned long theDollars = 0,
             unsigned int theCents = 0);
    ~Currency() { }
private:
    long amount;
};

ostream& operator<<(ostream &out, const Currency &c)
{
    out << c.amount;
    return out;
}

int main(int argc, char* argv[])
{
    Currency g(_plus, 1, 20);

    cout << g << endl;;
    return 0;
}
```



```
xiaomb@LAPTOP-IUK2M5JJ:~/code/chapter01$ make currency10 && ./currency10
g++ currency10.cpp -o currency10
currency10.cpp: In function ‘std::ostream& operator<<(std::ostream&, const Currency&)’:
currency10.cpp:29:14: error: ‘long int Currency::amount’ is private within this context
   29 |     out << c.amount;
      |              ^~~~~~
currency10.cpp:16:10: note: declared private here
   16 |     long amount;
      |          ^~~~~~
make: *** [makefile:16: currency10] Error 1
```



```
xiaomb@LAPTOP-IUK2M5JJ:~/code/chapter01$ make currency10 && ./currency10
g++ currency10.cpp -o currency10
120
```

类成员权限

- 类成员的权限可以是public, protected, 和private
  - Protected可以被子类访问

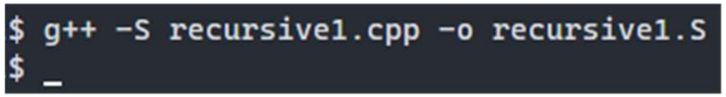
```
class Currency
{
public: // default to struct
     …
protected: // default to class
     …
private:
     …
}
```

递归函数

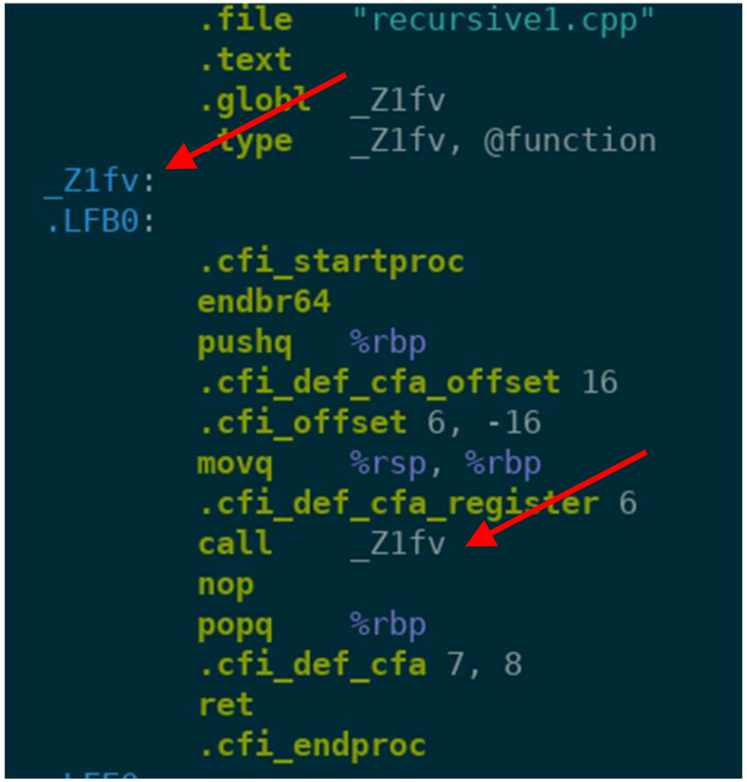
- 递归函数：自己调用自己的函数

- 直接递归：f(){ f(); }

```
void f()
{
  f();
}

int main(int argc, char *argv[])
{
  f();
  return 0;
}
```



```
$ g++ -S recursive1.cpp -o recursive1.S
$ _
```



```
        .file   "recursive1.cpp"
        .text
        .globl  _Z1fv
        .type   _Z1fv, @function
_Z1fv:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        call    _Z1fv
        nop
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

- 间接递归：f(){ g(); }, g(){ f(); }

如何停止?

数学函数的递归定义

- 数学函数中的递归定义：阶乘函数 **n!**
  - n! = n*(n-1)*(n-2)*…*2*1

$$f(n)=\begin{cases}1 & n\leqslant 1\\ nf(n-1) & n>1\end{cases} \longleftarrow$$

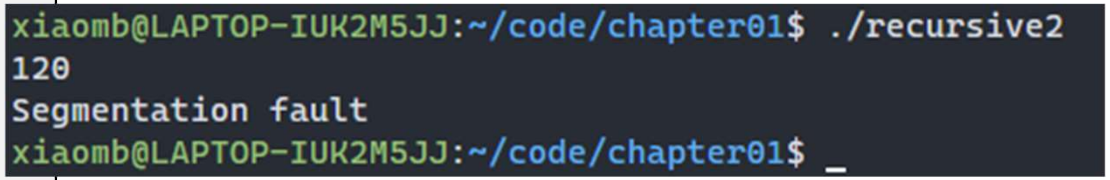
停止条件

- 完整的递归定义必须包含：
  - 基本部分：直接定义
  - 递归部分：函数被自己定义，参数向基本部分变化
  - **f(5)**=5f(4)=20f(3)=60f(2)=120f(1)=**120**

C++递归函数

- 一个正确的C++递归函数也必须包括基本部分和递归部分
  - 递归部分向基本部分靠近

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

int factorial(int n)
{
    if(n == 1)
        return 1;
    return n * factorial(n-1);
}

int factorial_error(int n)
{
    return n * factorial_error(n-1);
}

int main(int argc, char *argv[])
{
    cout << factorial(5) << endl;

    cout << factorial_error(5) << endl;

    return 0;
}
```

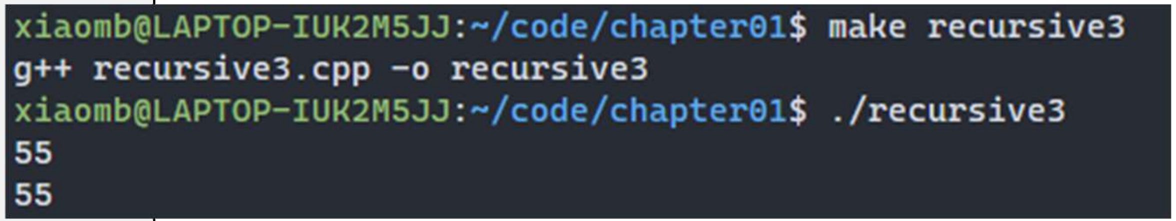


```
xiaomb@LAPTOP-IUK2M5JJ:~/code/chapter01$ ./recursive2
120
Segmentation fault
xiaomb@LAPTOP-IUK2M5JJ:~/code/chapter01$ _
```

sum的递归实现

- 求和函数即可以循环实现，也可以递归实现
  - sum(n) = n + (n-1) + … + 2 + 1
  - sum(n) = n + sum(n-1); sum(1) = 1;

```
template<typename T>
T sum(T a[], int n)
{
  T sum = 0;
  for(int i = 0; i < n; i++)
    sum += a[i];
  return sum;
}

template<typename T>
T sumR(T a[], int n)
{
  if(n == 1)
    return a[0];
  return a[0] + sumR(a+1, n-1);
}

int main(int argc, char *argv[])
{
  int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

  cout << sum(a, 10) << endl;
  cout << sumR(a, 10) << endl;

  return 0;
}
```



```
xiaomb@LAPTOP-IUK2M5JJ:~/code/chapter01$ make recursive3
g++ recursive3.cpp -o recursive3
xiaomb@LAPTOP-IUK2M5JJ:~/code/chapter01$ ./recursive3
55
55
```

n个数的排列

- n个不同元素的排列，如a，b，c三个元素
  - abc，acb，bac，bca，cab，cba
  - n!种排列

- 求排列的递归表示
  - 一个元素x的排列：x本身
  - 多个元素{x_1,…,x_n}的排列:
    - x_1为首元素，拼接{x_2,…,x_n}的所有排列
    - x_2为首元素，拼接{x_1,x_3,…,x_n}的所有排列
    - …
    - x_n为首元素，拼接{x_1,x_2,…,x_{n-1}}的所有排列

n个数的排列

- 元素{a, b, c}, perm({…})输出输入元素集合的所有排列 **abc,acb,bac,bca,cab,cba**
- perm({a, b, c})
  - a与perm({b, c})输出的所有排列拼接 **abc,acb**
  - b与perm({a, c})输出的所有排列拼接 **bac,bca**
  - c与perm({a, b})输出的所有排列拼接 **cab,cba**
- perm({b, c}) **bc, cb**
  - b与perm({c})输出的所有拼接排列 **bc**
  - c与perm({b})输出的所有拼接排列 **cb**
- perm ({c}) 输出c **c**

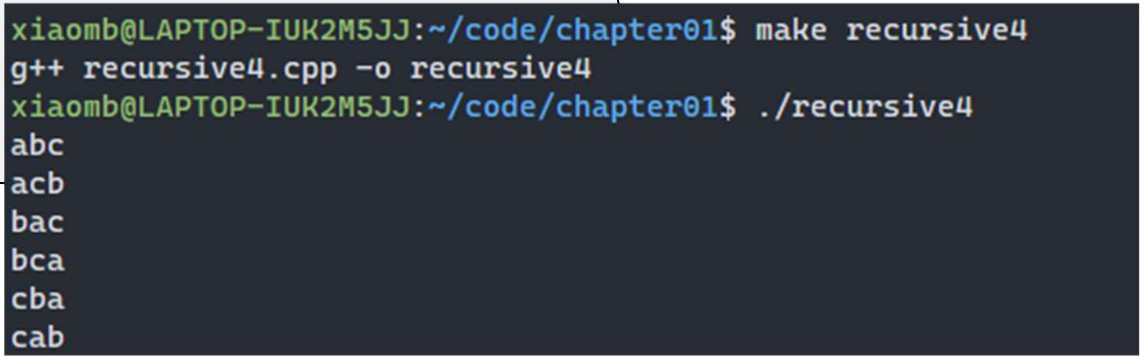
n个数的排列

```
template<typename T>
void permutations(T list[], int k, int m)
{

  if(k == m) {
    copy(list, list+m+1, ostream_iterator<T>(cout, ""));
    cout << endl;
  }else{
    for(int i = k; i <= m; i++){
      swap(list[k], list[i]);
      permutations(list, k+1, m);
      swap(list[k], list[i]);
    }
  }
}


int main(int argc, char *argv[])
{
  char abcd[] = {'a', 'b', 'c'};
  permutations(abcd, 0, 2);
  return 0;
}
```

3. 处理到最后一个元素时，打印整个数组

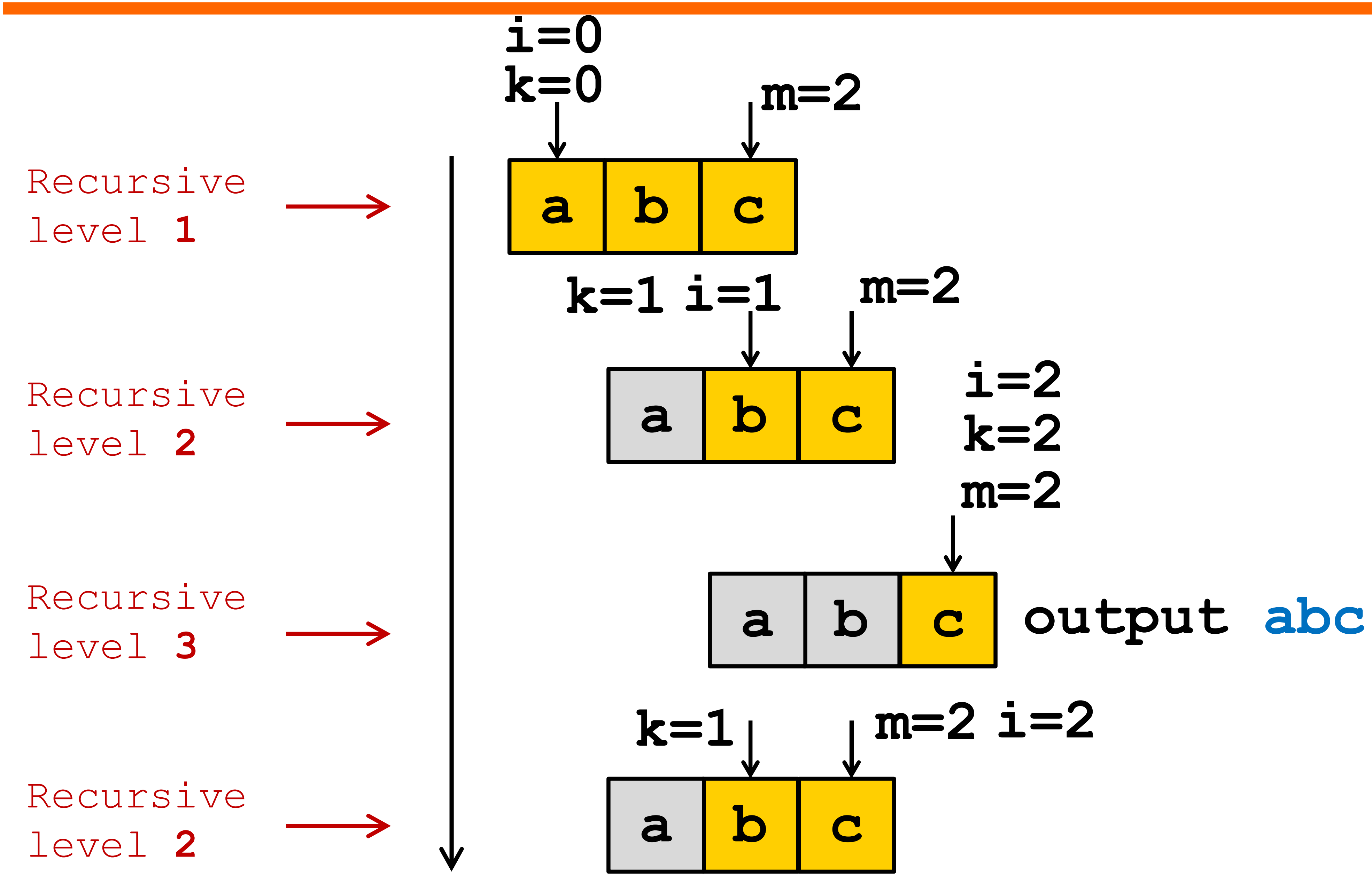
1. 对于当前子数组，逐个将元素置换到子数组首位

2. 处理从第二个元素开始的子数组



```
xiaomb@LAPTOP-IUK2M5JJ:~/code/chapter01$ make recursive4
g++ recursive4.cpp -o recursive4
xiaomb@LAPTOP-IUK2M5JJ:~/code/chapter01$ ./recursive4
abc
acb
bac
bca
cba
cab
```

n个数的排列



i=0
k=0
m=2

Recursive level 1

a b c

k=1 i=1
m=2

Recursive level 2

a b c

i=2
k=2
m=2

Recursive level 3

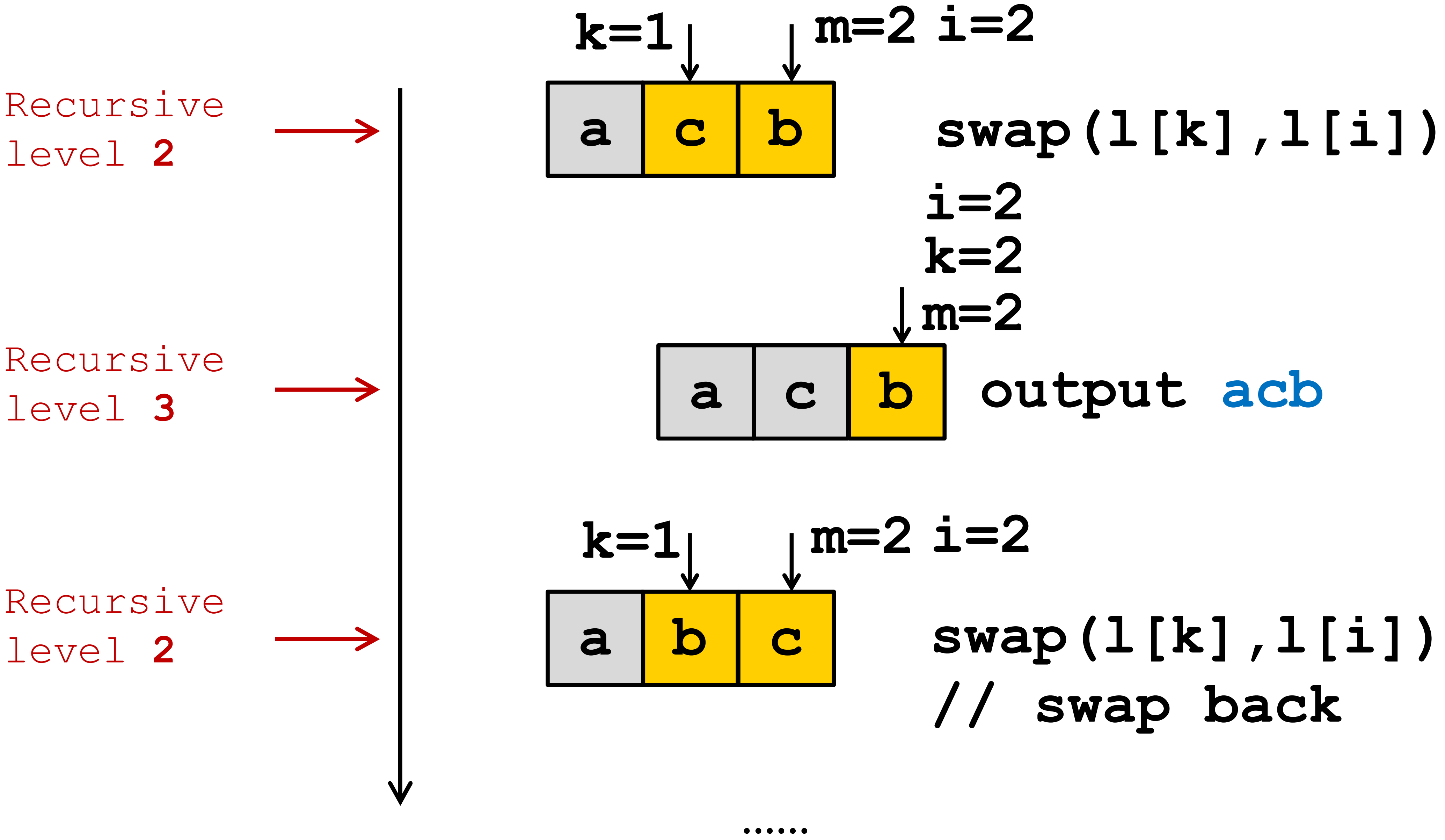
a b c output abc

k=1
m=2 i=2

Recursive level 2

a b c

n个数的排列



Recursive level 2

k=1 m=2 i=2

a c b

swap(l[k],l[i])

Recursive level 3

i=2

k=2

m=2

a c b

output acb

Recursive level 2

k=1 m=2 i=2

a b c

swap(l[k],l[i])

// swap back

……

C++标准模板库（STL）

- std::copy

```
#include <algorithm>
template<class InputIt, class OutputIt>
OutputIt std::copy(InputIt first, InputIt last, OutputIt d_first);
```

  - 完成通用拷贝功能
  - 通过迭代器（iterator）定位数据

- std::swap

```
#include <algorithm>
template<class T>
void swap(T &a, T &b);
```

  - 完成通用交换功能

C++标准模板库（STL）

- C++ STL (C++ Standard Template Library)
  - 容器 (containers) : vector<int>
  - 迭代器 (iterators) : vector<int>::iterator
  - 算法 (algorithms) : adjacent_find()
  - 函数对象 (functors) : not_equal_to

C++标准模板库（STL）

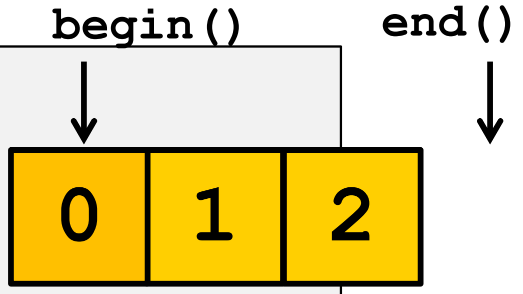
- 示例

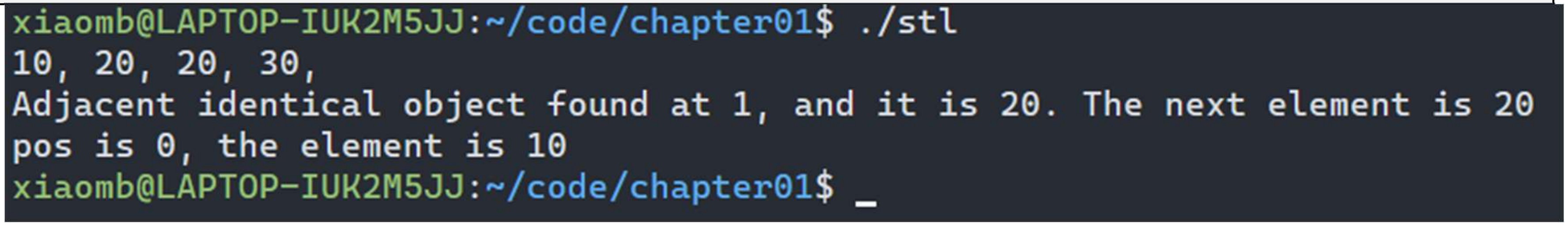
```
#include <vector>
#include <iostream>
#include <functional>
using namespace std;

int main(int argc, char *argv[])
{
  vector<int> m_array = {10, 20, 20, 30};
  for(vector<int>::iterator iter = m_array.begin(); iter != m_array.end(); iter++)
      cout << *iter << ", ";
  cout << endl;

  vector<int>::iterator pos = adjacent_find(m_array.begin(), m_array.end());
  if(pos == m_array.end())
    cout << "Failed to find adjacent identical objects" << endl;
  else
    cout << "Adjacent identical object found at " << (pos - m_array.begin())
         << ", and it is " << *pos
         << ". The next element is " << (*++pos)
         << endl;

  pos = adjacent_find(m_array.begin(), m_array.end(), not_equal_to<int>());
  cout << "pos is " << (pos - m_array.begin()) << ", the element is " << *pos << endl;
  return 0;
}
```

C++ 11/17/20 新特性

auto

lambda





```
xiaomb@LAPTOP-IUK2M5JJ:~/code/chapter01$ ./stl
10, 20, 20, 30,
Adjacent identical object found at 1, and it is 20. The next element is 20
pos is 0, the element is 10
xiaomb@LAPTOP-IUK2M5JJ:~/code/chapter01$ _
```

程序测试

- 测试程序的正确性
  - 使用测试数据为输入运行程序，与预期结果作比较
  - 单次正确不代表程序正确
  - 尽可能发现多的缺陷和问题

- 设计测试数据的方法
  - 黑盒法：尽可能覆盖所有的输入
  - 白盒法：尽可能覆盖所有的执行路径

课后作业

- P23 16
- P29 23 24