第一节 输入输出

引导

c语言是面向过程的编程范式

一共四种编程范式．面向过程，面向对象，函数式编程，泛型编程

函数是一种映射，

数组是展开的函数，函数是压缩的数组

函数是计算式，数组是计算式．两种方式可以相互转换

深度神经网路可以逼近任何函数

输出函数说明

`printf`函数

- 头文件：　stdio.h
- 原型：　`int printf(const char *format, ...);`
- `format:`格式控制字符串
- ．．．可变参数列表
- 返回值：成功读入的参数个数

输入函数说明

`scanf`函数

- 头文件：　stdio.h
- 原型：　`int scanf(const char *format, ...);`
- `format:`格式控制字符串
- ．．．可变参数列表
- 返回值：成功读入的参数个数

随堂练习：

1. 使用`printf`输出一个数字的某一位

```
/*************************************************************************
    > File Name: wie.cpp
    > Author: ldc
    > Mail: litesla
    > Created Time: 2018年09月13日 星期四 18时29分42秒
 ************************************************************************/

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

```c
int main(){
    //1,2,3int a;
    char str[100];
    //2,FILE *fp = fopen("/dev/null","w");
    //1,2,3scanf("%d", &a);
    //1,printf(" %d", printf("%d", a));
    //2,fprintf(stdout,"%d", fprintf(fp, "%d",a));
    //3,printf("%d\n", sprintf(str,"%d",a));
    for (int i = 0,j = 0; i <= 20; i++) {
        j += sprintf(str + j,"%d",i);
    }
    printf("%s",str);
    return 0;
}
```

2. 读入一整行字符串

```c
/*************************************************************************
   > File Name: clang_test2.c
   > Author: hug
   > Mail:   hug@haizeix.com
   > Created Time: 四  9/13 19:13:09 2018
 ************************************************************************/

#include <stdio.h>

int main() {
    char str[100];
    scanf("%[^\n]s", str);
    int n = printf("%s", str);
    printf(" has %d chars\n", n);
    return 0;
}
```

第二节：数学运算

% / * (+ -) (& | ^~)运算符速度从低到高

pow函数说明

`pow`函数：指数函数

- 头文件：`math.h`
- 原型：`double pow(double a, double b);`
- a:底数
- b:指数
- 返回值：返回 a 的b次幂结果
- 例子：`pow(2,3) = 8`

`sqrt`函数：开平方数

- 头文件：`math.h`
- 原型：`double sqrt(double x);`
- x:被开方数
- 返回值：返回值根号ｘ结果
- 例子：`sqrt(16) = 4` 4

ceil向上取整函数

ｆｌｏｏｒ向下取整函数

ａｂｓ(stdlib.h)整数绝对值函数

ｆａｂｓ实数绝对值函数

ｌｏｇ以ｅ为底对数函数

ｌｏｇ１０以１０为底对数函数

ａｃｏｓ函数(arccos)

例子：`acos(-1) = 3.1415926`

随堂练习　mysqrt newton

```
/*************************************************************************
    > File Name: mysqrt.cpp
    > Author: ldc
    > Mail: litesla
    > Created Time: 2018年09月13日 星期四 20时03分55秒
 ************************************************************************/

#include<iostream>
#include <time.h>
#include <math.h>
using namespace std;

#define TEST_TIME(func)({\
    int begin=clock();\
    double ret=func;\
    int end=clock();\
    printf("%lfms\m",(end-begin) * 1.0/ CLOCK_PER_SEC * 1000);\
})

double _sqrt(double x){
    double head = -100,tail = 100, mid;
    #define EPS 1e-7
    while(tail - head > EPS){
        mid=(head + tail)/2.0;
        if(mid * mid < x) head = mid;
        else tail= mid;
    }
    #undef EPS
    return tail;
}
double f_func(double x,double n){
```

```
    return x * x - n;
}
double ff_func(double x){
    return 2 * x;
}

double newton(int n, double (*f)(double,double), double (*ff)(double)){
    double x = 1.0;
    #define EPS 1e-7

    while(fabs(f(x,n)) > EPS){
        x -= f(x,n)/ff(x);
    }
    #undef EPS
    return x;
}

int main(){
    int x;
    scanf("%d", &x);
    printf("%lf\n", _sqrt(x));
    printf("%lf\n", newton(x,f_func,ff_func));

    return 0;
}
```

第三节　流程控制

计算机比人勤奋不是比人聪明

关系运算符

`==, >=, >, <,<=,!, !=`

条件分支判断语句

if语句

switch语句:

switch内部无法定义局部变量

循环语句

while语句, do while

for语句

fast_read

```
/*************************************************************************
   > File Name: fast_read.cpp
   > Author: hug
   > Mail:   hug@haizeix.com
```

```
    > Created Time: 二  9/18 19:48:33 2018
 ************************************************************************/

#include <iostream>
using std::cin;
using std::cout;
using std::endl;

char ss[1<<17], *A = ss, *B = ss;
inline char gc() {
    return A == B && (B = (A = ss) + fread(ss, 1, 1 << 17, stdin), A == B) ? -1 : *A++;
}

template<typename T> inline void sdf(T &x) {
    char c;
    T y = 1;
    while (c = gc(), (c < 48 || c > 57) && c != -1) {
        if (c == '-') y = -1;
    }
    x = c ^ 48;
    while (c = gc(), (c <= 57 && c >= 48)) {
        x = (x << 1) + (x << 3) + (c ^ 48);
    }
    x *= y;
}

int main() {
    for (int i = 48; i <= 57; i++) {
        printf("%d ^ 48 = %d\n", i, i ^ 48);
    }
    int n, a;
    sdf(n);
    for (int i = 0; i < n; i++) {
        sdf(a);
    }
    return 0;
}
```

第四节　函数

函数是压缩的数组，数组是展开的函数

函数会增加程序的可读性

K＆R风格函数定义

```c
int is_prime(x)
int x;
{
    for (int i = 2; i * i <= x; i++) {
        if (x % i == 0) return 0;
    }
    return 1;
}
```

递归是一种编程技巧不是一种算法

程序调用自身的编程技巧叫做递归

1. 明确递归函数的语义
2. 设置边界条件
3. 实现递归过程和处理过程
4. 实现结果返回

规范化问题的思考方式

才能在遇到难题的时候寻找可行解

辗转相除法的证明

定理：a 和 b 两个整数的最大公约数等于 b 与 a % b 的最大公约数。

形式化表示：假设 a, b != 0 则，gcd(a, b) = gcd(b, a % b)

证明1：

1、设 c = gcd(a, b)，则 a = cx, b = cy

2、可知 a % b = r = a - k * b = cx - kcy = c(x - ky)

3、可知 c 也是 r 的因数

4、其中 x - ky 与 y 互素，见**证明2**

所以可知：gcd(a, b) = gcd(b, r) = gcd(b, a % b)

证明2：

1、假设 gcd(x - ky, y) = d

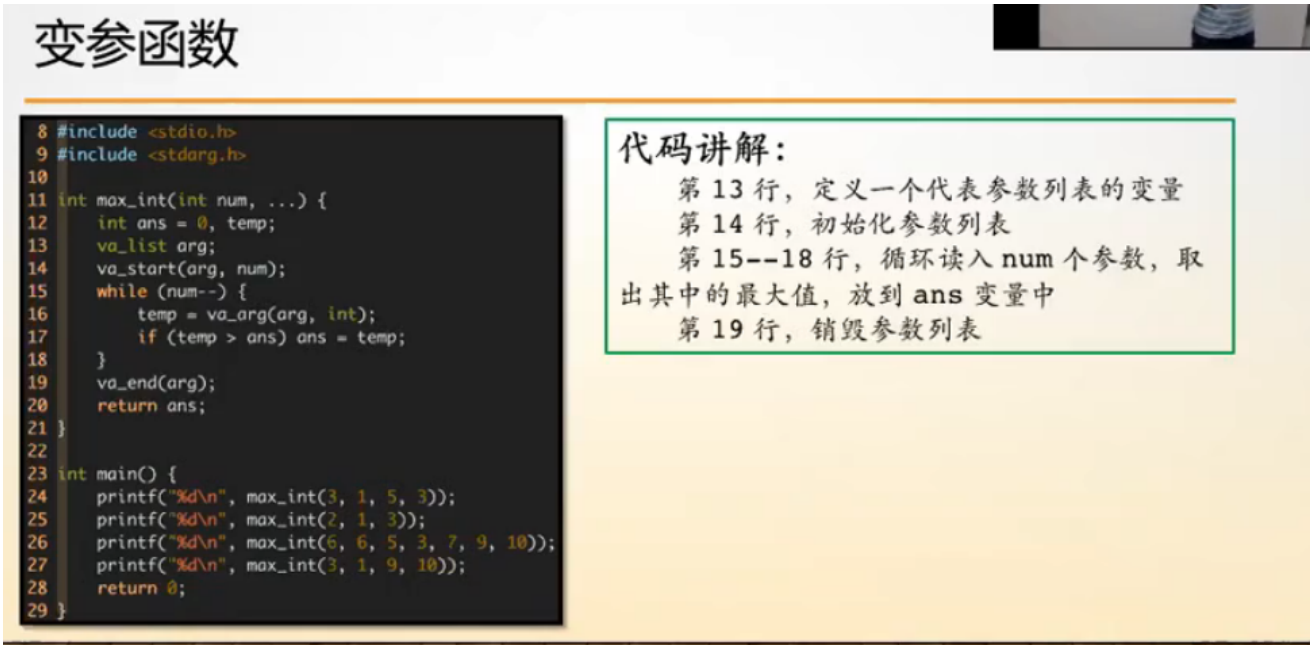
2、则 y = nd, x - ky = md，则 x = knd + md = d(kn + m)

3、重新表示 a, b，则有 a = cd(kn + m)，b = cdn

4、则可得 gcd(a, b)>=cd，又因为 gcd(a, b)=c，所以 d=1

变参函数

学技术一定要有敏锐的目光



变参函数代码演示

```c
/*************************************************************************
   > File Name: is_prime.c
   > Author: hug
   > Mail:   hug@haizeix.com
   > Created Time: 四  9/20 19:52:14 2018
 ************************************************************************/

#include <stdio.h>
#include <stdarg.h>
#include <inttypes.h>

int is_prime(int x) {
    for (int i = 2; i * i <= x; i++) {
        if (x % i == 0) return 0;
    }
    return 1;
}

int max_int(int n, ...) {
    va_list arg;
    va_start(arg, n);
    int ans = INT32_MIN;
    for (int i = 0; i < n; i++) {
        int temp = va_arg(arg, int);
        ans = temp > ans ? temp : ans;
    }
    return ans;
}
```

my_print

```c
/*************************************************************************
   > File Name: 3.my_print.c
   > Author: hug
   > Mail:   hug@haizeix.com
   > Created Time: 四  9/20 20:26:47 2018
 ************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <inttypes.h>

int print_int(int x, int flag) {
    if (x == 0) {
        flag && putchar('0');
        return !!(flag);
    }
    int temp = x, ret = 0, digit = 0;
    x = 0;
    if (temp < 0) temp = -temp, printf("-");
    while (temp) {
        x = x * 10 + temp % 10;
        temp /= 10;
        digit++;
    }
    while (digit--) {
        putchar(x % 10 + '0');
        x /= 10;
        ret++;
    }
    return ret;
}

int my_printf(const char *frm, ...) {
    int cnt = 0;
    va_list arg;
    va_start(arg, frm);
    for (int i = 0; frm[i]; i++) {
        switch (frm[i]) {
            case '%' : {
                i++;
                switch(frm[i]) {
                    case 'd' : {
                        int temp = va_arg(arg, int);
                        int p1 = temp / 10, p2 = temp % 10;
                        if (temp < 0) {
                            p1 = -p1, p2 = -p2;
                            putchar('-'); cnt++;
                        }
                        cnt += print_int(p1, 0);
                        cnt += print_int(p2, 1);
                    } break;
                    default :
```

```
                        fprintf(stderr, "error : unknow %%%c\n", frm[i]);
                        exit(1);
                }
            } break;
            default:
                putchar(frm[i]);
                cnt++;
        }
    }
    return cnt;
}

int main() {
    int n = 123;
    my_printf("hello world\n");
    my_printf("n = %d\n", n);
    my_printf("n = %d\n", 12000);
    my_printf("n = %d\n", 0);
    my_printf("n = %d\n", -567);
    my_printf("n = %d\n", INT32_MAX);
    my_printf("n = %d\n", INT32_MIN);
    return 0;
}
```

第五节：数组与预处理命令

数组是一片连续的存储空间

元素存在递推关系的化用数组存储比较号

函数是压缩的数组

素数筛

折半查找

```
/*************************************************************************
    > File Name: binary_search.cpp
    > Author: ldc
    > Mail: litesla
    > Created Time: 2018年09月23日 星期日 10时41分54秒
 ************************************************************************/

#include<iostream>
using namespace std;

int binary_search(int (*arr)(int), int x, int n){
    int head = 0,tail = n - 1;
    while(head <= tail){
        int mid = (head + tail) >> 1;
        if(arr(mid) == x) return mid;
        else if (arr(mid) < x) head = mid + 1;
```

```
        else tail = mid - 1;
    }
    return -1;
}



int main(){
    int arr[100];
    char *p1 = (char * )(arr + 1);
    char *p2 = (char * )(arr + 2);
    long long * p3 = (long long *)(arr + 1);
    p1[0] = 'a';
    p1[1] = 'b';
    p1[2] = 'c';
    p1[3] = 'd';
    for(int i = 0; i < 3; i++) p2[i] = p1[i];
    printf("p1 = %s, p1 = %lx\n", p1 ,*(long long*)p1);
    //printf("p3 = %ld",*p3);

    return 0;
}
/*
int main(){
    char arr[100];
    int *p1 = (int *)(arr + 1);
    int *p2 = (int *)(arr + 2);
    *p1 = 1;
    *p2 = 2;
    printf("*p1 = %d\n*p2 = %d\n", *p1, *p2);
}*/

/*
int main(){
    int ret;
    unsigned char * p = (unsigned char *)&ret;
    int a,b,c,d;
    scanf("%d%d%d%d",&a, &b, &d, &d);
    p[0] = a;
    p[1] = b;
    p[2] = c;
    p[3] = d;
    //p[0] = d;
    //p[1] = c;
    //p[2] = b;
    //p[3] = a;
    //注意高对高，低对低
    printf("ret: %u\n", ret);
    return 0;
}*/
```

开脑洞可以弥补经验上的不足

编译器预定义的宏

```
__DATA__
__TIME__
__LINE__
__func__
__FUNC__
........
```

max(a, b)宏

```
/*************************************************************************
    > File Name: max.cpp
    > Author: ldc
    > Mail: litesla
    > Created Time: 2018年09月23日 星期日 11时59分24秒
 ************************************************************************/

#include<iostream>
using namespace std;


#define P(func) {\
    printf("%s = %d\n", #func, func); \
}

//#define MAX(a,b) ((a) > (b) ? (a) : (b))

#define MAX(a,b) ({\
        __typeof(a) aa = (a);\
        __typeof(b) bb = (b);\
        aa > bb ? aa : bb;\
})


int main(){
    //P(MAX(2,3));
    //P(5 + MAX(2,3));
    //P(MAX(2,MAX(3,4)));
    //P(MAX(2,3 > 4 ? 3 : 4));
    int a = 7;
    P(MAX(a++,6));
    printf("a = %d\n", a);
    ( {int a = 3;});
    printf("a = %d\n",a);
    return 0;
}
```

条件式编译

函数	说明
#ifdef DEBUG	是否定义了 DEBUG 宏
#ifndef DEBUG	是否没定义 DEBUG 宏
#if MAX_N == 5	宏 MAX_N 是否等于5
#elif MAX_N == 4	否则宏 MAX_N 是否等于4
#else	
#endif	

DEBUG演示

```
/*************************************************************************
   > File Name: 4.define.cpp
   > Author: hug
   > Mail:   hug@haizeix.com
   > Created Time: 日  9/23 11:38:10 2018
 ************************************************************************/

#include <iostream>
using std::cin;
using std::cout;
using std::endl;


#ifdef DEBUG
#define P printf("%s : %d\n", __PRETTY_FUNCTION__, __LINE__)
#else
#define P
#endif

void testfunc() {
    P;
}

int main() {
    printf("%s %s\n", __DATE__, __TIME__);
    P;
    testfunc();
    return 0;
}
```

第六节　复杂结构和指针

结构体内部对齐效应

可以通过预编译命令进行调整 (这么做是为了效率)

共用体

一个共用体类型占用空间的大小是他里面存储的最大的数据类型

IP

```
/*************************************************************************
    > File Name: ip.cpp
    > Author: ldc
    > Mail: litesla
    > Created Time: 2018年10月04日 星期四 10时04分05秒
 ************************************************************************/

#include<iostream>
#include <string.h>
using namespace std;

union IP{
    struct{
        unsigned char arr[4];
    }ip;
    unsigned int num;
};

int main(){
    int a,b,c,d;
    IP ip;
    while(scanf("%d.%d.%d.%d",&a, &b, &c, &d) != EOF){
        ip.ip.arr[0] = a;
        ip.ip.arr[1] = b;
        ip.ip.arr[2] = c;
        ip.ip.arr[3] = d;
        printf("%u\n", ip.num);
       // memset(&ip,'\0',sizeof(ip));
    }
    return 0;
}
```

结构体代码演示

```
struct test{
    short a;//占用两个字节只能放在距离开始位置偶数个距离的地方
    char b;//占用一个字节，有个空就能差
    int c;//占用四个字节，只能放在距离开始位置４个字节的地方
    double e;//. . . .
}
```

扩充浮点型的表示形式

信息本身没有改变关键是你怎么取看待他

一个既可以存储整形又可以存储浮点型的数组

```
/*************************************************************************
   > File Name: 6.struct&union.cpp
   > Author: hug
   > Mail:   hug@haizeix.com
   > Created Time: 四 10/ 4 10:17:06 2018
 ************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

struct person {
    char name[20];
    int age;
    char gender;
    float height;
};

struct test {
    char b;
    short a;
    int c;
    double e;
};

struct Number {
    int type; // 0->int or 1->double
    union {
        int num1;
        float num2;
    } N;
};

void print(Number *n) {
    switch (n->type) {
        case 0: printf("%d\n", n->N.num1); break;
        case 1: printf("%f\n", n->N.num2); break;
    }
    return ;
}

void set(Number *n, float num) {
    n->type = 1;
    n->N.num2 = num;
}

void set(Number *n, int num) {
```

```
    n->type = 0;
    n->N.num1 = num;
}

int main() {
    srand(time(0));
    struct test a;
    printf("sizeof(person) : %d\n", sizeof(struct person));
    printf("%p %p %p", &a.a, &a.b, &a.c);
    #define MAX_N 20
    Number arr[MAX_N];
    for (int i = 0; i < MAX_N; i++) {
        int op = rand() % 2;
        switch (op) {
            case 0: {
                int value = rand() % 100;
                set(arr + i, value);
            } break;
            case 1: {
                float value = (1.0 * (rand() % 10000) / 10000) * 100;
                set(arr + i, value);
            } break;
        }
    }
    for (int i = 0; i < MAX_N; i++) {
        print(arr + i);
    }
    arr[0].N.num2 = 1.0;
    printf("arr[0].N.num1 = %d\n", arr[0].N.num1);
    printf("arr[0].N.num1 = %x\n", arr[0].N.num1);
    return 0;
}
```

变量的地址

指针变量存储的是变量的首地址

指针变量也是变量，也有地址

指针的类型决定了指针在解析运算符下影响了响应的操作

重点概念

- 指针变量也是变量

等价形式交换

*p = a

p+ 1 = &p[1]

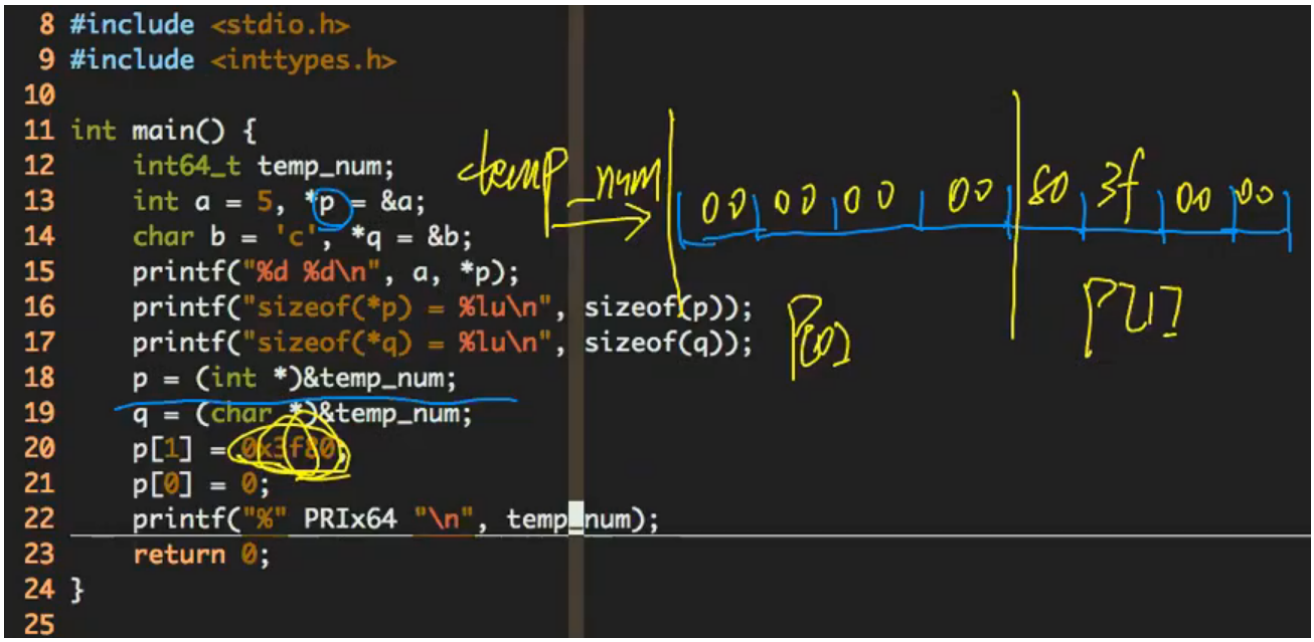
p->filed = (*p).filed = a.filed

函数指针

int (*add)(int, int);//变量

typedef int (*add)(int,int);//类型

指针的解析




```c
#include <stdio.h>
#include <inttypes.h>

int main() {
    int64_t temp_num;
    int a = 5, *p = &a;
    char b = 'c', *q = &b;
    printf("%d %d\n", a, *p);
    printf("sizeof(*p) = %lu\n", sizeof(p));
    printf("sizeof(*q) = %lu\n", sizeof(q));
    p = (int *)&temp_num;
    q = (char *)&temp_num;
    p[1] = 0x3f80;
    p[0] = 0;
    printf("%" PRIx64 "\n", temp_num);
    return 0;
}
```

```c
#include <stdio.h>
#include <inttypes.h>

int main() {
    int64_t temp_num;
    int a = 5, *p = &a;
    char b = 'c', *q = &b;
    printf("%d %d\n", a, *p);
    printf("sizeof(*p) = %lu\n", sizeof(p));
    printf("sizeof(*q) = %lu\n", sizeof(q));
    p = (int *)&temp_num;
    q = (char *)&temp_num;
    p[1] = 0x3f80;
    p[0] = 0;
    printf("%" PRIx64 "\n", temp_num);
    double *p_ = (double *)&temp_num;
    *p_ = 1.0;
    printf("%" PRIx64 "\n", temp_num);
    return 0;
}
```

c函数指针语法检查

c语法在进行语法检查的时候是看函数指针的类型的参数列表,和你调用的类型对不对的上,对于你之前是那个函数他不管

main函数

是有参数的

`int main(int argc, char *argv[], char *env[])`

argc标记的是有几个argv

env标记结束的方式是最后一个位置指向一个空地址

取得环境变量的作用适应不同的系统或者环境

test测试框架

test.h

```
/*************************************************************************
    > File Name: test.h
    > Author: ldc
    > Mail:
    > Created Time: 2018年10月05日 星期五 15时17分05秒
 ************************************************************************/

#ifndef _TEST_H
#define _TEST_H
#include <stdlib.h>

//测试结构体
struct TestFuncData{
    int total, expect;
    //总量和成功量
};

//声明一个函数指针传递的参数是,链表节点
typedef void (*test_func_t)(struct TestFuncData *);

//链表节点
struct FuncData {
    const char *a_str, *b_str;//函数名
    test_func_t func;//函数指针
    struct FuncData *next;//链表指针
};
//增加链表节点的函数,参数有函数名,和需要测试的函数指针
void addFuncData(const char *a, const char *b, test_func_t func);
//声明一个函数
int RUN_ALL_TEST();


//声明一个函数
//__attribute__让下面那个函数比主函数先运行
//生成一个增加函数
```

```
//函数内部调用增加链表节点函数
//1讨论为什么要在函数内部调用那个函数,
//因为测试是多种函数一起编译,这样就可以重复的调用addFuncData
//最后一行生成一个函数头,注意函数的参数是测试结构体
//注意这个是多行宏定义
#define TEST(a,b) \
    void a##_haizeix_##b(struct TestFuncData *); \
    __attribute__((constructor)) \
    void ADDFUNC_##a##_haizeix_##b() { \
        addFuncData(#a, #b, a##_haizeix_##b); \
    } \
    void a##_haizeix_##b(struct TestFuncData *__data)


//判断该是否相等,ret是局部变量
//注意最外面有()
#define EXPECT_EQ(a, b) ({ \
    int ret; \
    printf("%s == %s %s\n", #a, #b, (ret = (a == b)) ? "True" : "False"); \
    __data->total += 1; \
    __data->expect += ret; \
})

#define EXPECT_GT(a, b) ({ \
    int ret; \
    printf("%s > %s %s\n", #a, #b, (ret = (a > b)) ? "True" : "False"); \
    __data->total += 1; \
    __data->expect += ret; \
})

#define EXPECT_LT(a, b) ({ \
    int ret; \
    printf("%s < %s %s\n", #a, #b, (ret = (a < b)) ? "True" : "False"); \
    __data->total += 1; \
    __data->expect += ret; \
})

#define EXPECT_NE(a, b) ({ \
    int ret; \
    printf("%s != %s %s\n", #a, #b, (ret = (a != b)) ? "True" : "False"); \
    __data->total += 1; \
    __data->expect += ret; \
})

#define EXPECT_GE(a, b) ({ \
    int ret; \
    printf("%s >= %s %s\n", #a, #b, (ret = (a >= b)) ? "True" : "False"); \
    __data->total += 1; \
    __data->expect += ret; \
})

#define EXPECT_LE(a, b) ({ \
    int ret; \
```

```
    printf("%s <= %s %s\n", #a, #b, (ret = (a <= b)) ? "True" : "False"); \
    __data->total += 1; \
    __data->expect += ret; \
})

#endif
```

test.c

```
/*************************************************************************
    > File Name: test.c
    > Author: ldc
    > Mail:
    > Created Time: 2018年10月05日 星期五 15时16分21秒
 ************************************************************************/
#include <stdio.h>
#include "test.h"
//链表头部
static struct FuncData *FuncData_head = NULL;
//增加链表节点函数,头插法
void addFuncData(const char *a, const char *b, test_func_t func) {
    struct FuncData *p = (struct FuncData *)malloc(sizeof(struct FuncData));
    p->a_str = a;
    p->b_str = b;
    p->func = func;
    p->next = FuncData_head;
    FuncData_head = p;
    return ;
}

int RUN_ALL_TEST() {
    struct FuncData *current_node, *next_node;
    if (FuncData_head == NULL) return 0;
    current_node = FuncData_head->next;
    FuncData_head->next = NULL;
    //链表逆序
    while (current_node != NULL) {
        next_node = current_node->next;
        current_node->next = FuncData_head;
        FuncData_head = current_node;
        current_node = next_node;
    }
    //三种不同测试结果的输出样式
    char color[3][100] = {
        "[ \033[1;32m%.2lf%%\033[0m ] total : %d, expect : %d\n",
        "[ \033[0;31m%.2lf%%\033[0m ] total : %d, expect : %d\n",
        "[ \033[1;31m%.2lf%%\033[0m ] total : %d, expect : %d\n"};
    //遍历链表,并且处理每个节点信息
    for (struct FuncData *p = FuncData_head; p; p = p->next) {
        printf("[%s, %s]\n", p->a_str, p->b_str);
        struct TestFuncData data = {0, 0};
```

```
        p->func(&data);
        double rate = 1.0 * data.expect / data.total * 100;
        int index = 0;
        if (rate < 100) index = 1;
        if (rate < 50) index = 2;
        printf(color[index], rate, data.total, data.expect);
    }
    return 0;
}
```

main.c

```
/*************************************************************************
    > File Name: text.cpp
    > Author: ldc
    > Mail: litesla
    > Created Time: 2018年10月04日 星期四 17时52分37秒
 ************************************************************************/

#include <stdio.h>
#include "test.h"

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

int is_prime(int x) {
    if (x <= 1) return 0;
    for (int i = 2; i * i <= x; i++) {
        if (x % i == 0) return 0;
    }
    return 1;
}

TEST(test, is_prime_func) {
    EXPECT(is_prime(2), 0);
    EXPECT(is_prime(-2), 0);
    EXPECT(is_prime(15), 0);
    EXPECT(is_prime(9973), 1);
}

TEST(test, add_func) {
    EXPECT(add(1, 2), 3);
    EXPECT(add(3, 4), 7);
    EXPECT(add(2, 2), 4);
}

int main() {
    return RUN_ALL_TEST();
}
```