C语言

里昂学长

哈喽各位亲爱的宝子，我是里昂！很高兴认识你。

本C语言课程为速成课程，主要为了让跨考或者C语言基础一般的同学能快速入门C语言，便于学习数据结构。

课程使用指南

1. 本课程包含知识点讲解以及代码演示，请在学习每个知识点后手敲一遍代码，课程代码文件已放入交流群。
2. 本课程主要用于速成和新手学习，有一些表述为了便于理解，不太严谨，还希望小伙伴们见谅~
2. 具体课程使用请根据自己复习进度及学习习惯安排(●'◡'●)

初识C语言
- Devc++
- 内存和变量
- 定义变量并赋值
- 输入输出
- 常见的运算符
- 条件判断
- 循环
- 变量和函数

进阶C语言
- 数组
- 指针
- 结构体
- 链表
- 引用
- 递归

初识C语言

中国人讲中文，英国人讲英文，计算机人讲计算机文。为了和计算打交道，我们必须学一门计算机能听懂的语言。

Devc++

我们写C语言代码时可能经常一不小心写错了还没发现，检查半天还是没法执行！

很多同学一顿C语言输出后，直接实现从C语言入门到放弃！

即使代码全写对了，还需要编译等工作才能执行

而IDE(集成开发环境)就是帮助我们检查代码正确与否和进行编译的工具

Devc++是IDE的一种，相对于VScode等工具，显然Devc++已经是个大老土了！它唯一的优势就是简单！

安装简单！打开简单！界面简单！对于新手来说，简单就是最好的。因此，本课程使用该IDE~

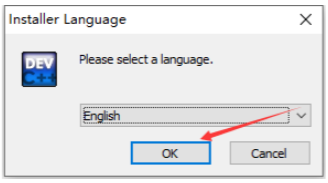
下载方式

1. 视频简介或者评论区都有下载链接
2. 百度搜索devc++ 安装即可（小心广告！）
3. 评论区加群，下载群文件即可

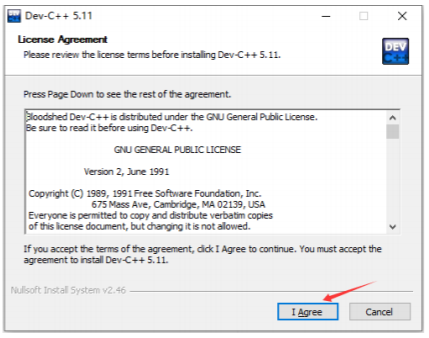
安装步骤

STEP1 双击安装包。
注意，万一电脑管家、安全卫士等软件跳出来提示“是否允许安装”，点击允许安装。

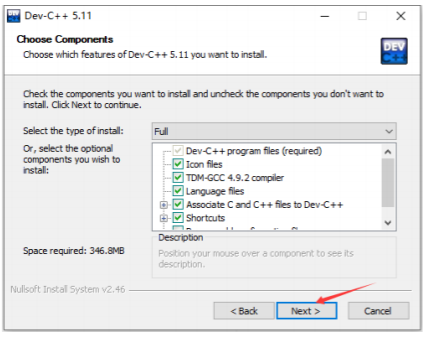
STEP2 点击OK。



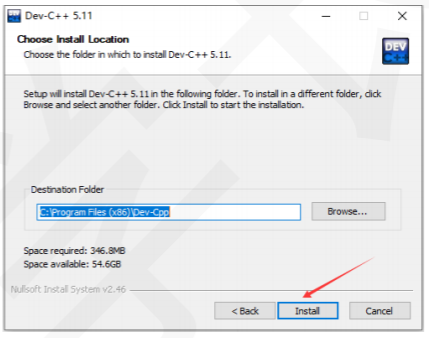
STEP3 点击I Agree。



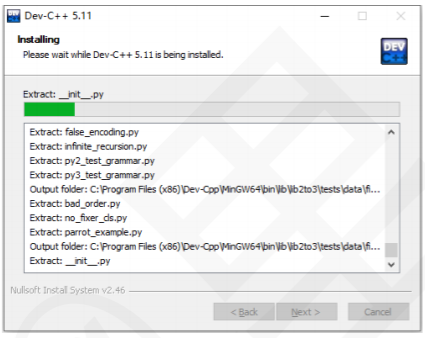
STEP4 点击next。



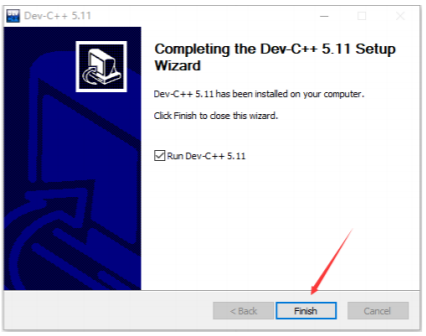
STEP5 点击Install。



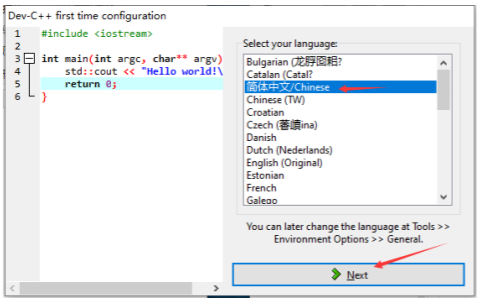
STEP6 等待安装。



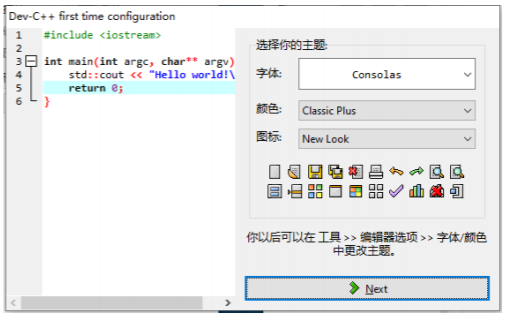
STEP7 点击Finish。



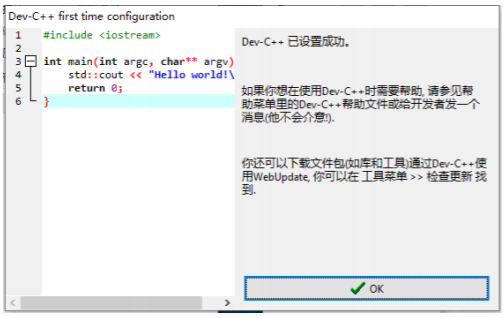
STEP8 点击next。



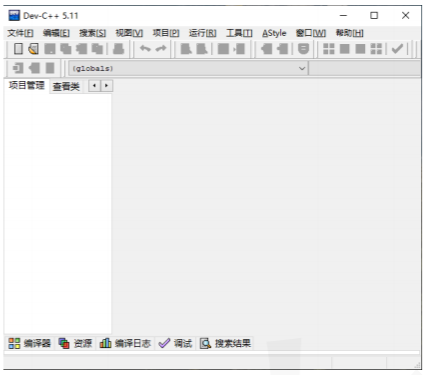
STEP9 点击next。



STEP10 点击OK。



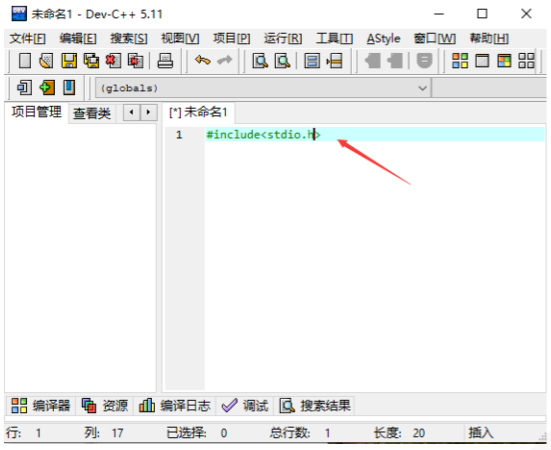
安装成功!



第一个程序

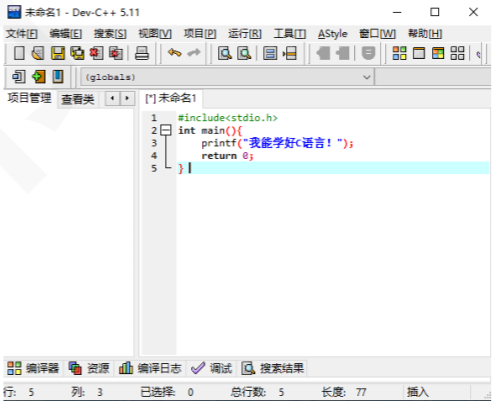
STEP1 我们点击 左上角 文件--》新建--》源代码便可以创立一个源代码文件。

STEP2 接着就可以在源代码文件中写代码啦！



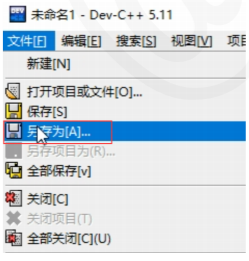
STEP3 小伙伴们可以把以下代码手敲一遍，或者直接复制也成！

```
#include<stdio.h>
int main(){
	printf("我能学好C语言！");
	return 0;
}
```

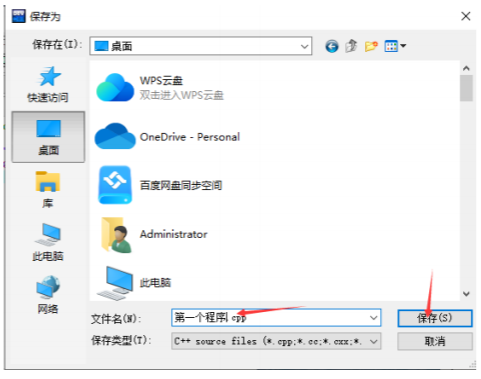


输入后效果如上图所示

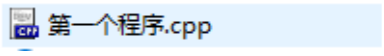
STEP4 点击左上角 文件--》另存为



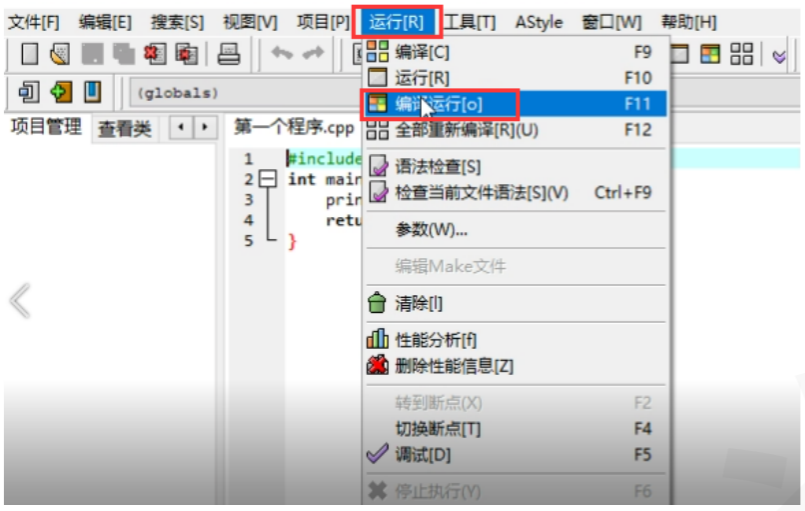
STEP5 修改文件名并保存。



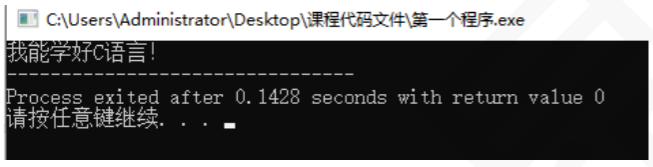
STEP6 双击打开源代码文件。



STEP7 点击 运行--》编译运行。



运行结果如下！恭喜你，你的第一个程序跑通了！



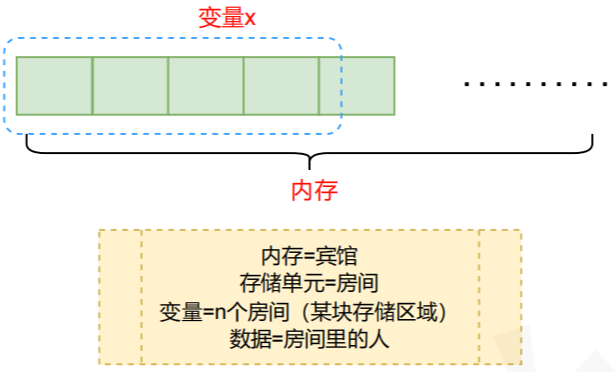
内存和变量

在计算机中有大量的数据，例如你的微信聊天记录，又或是要上交的期末大作业都是数据。这些数据都被保存在内存中。

内存由一个个小房间（即存储单元）构成，每个小房间都可以存储有限的数据。

而**变量代表了内存中的某一块存储区域，我们可以在这个变量中保存某个数（某些数据）。**

例如，变量x代表了内存的某块存储区域，这块存储区域由4个小房间共同构成。



举例，变量x就等于n个房间，n个房间里一共有100个人，所以变量x=100

因此，变量值就是指的一个变化的值！

所以，内存中的某个存储区域（变量）里的值是可以改变的，因此这个存储区域我们用变量表示。

变量不光可以用x表示，也可以用字母、数字及下划线表示！这里强调一下表示变量的规范：

1. 不是阿猫阿狗都可以做变量名，变量名只能由字母、数字及下划线构成

2. 变量名不能以数字开头

3. 变量名中字母的大小写也表示不同的变量

举例：

变量x（ ） 变量y（ ） 变量9e（ ） 变量liangxuezhang（ ） 变量923_e132（ ） 变量X（ ）

什么是变量类型?

现在有两个数100和10000000000000000000000000000000
很明显，前者是个三位数，而后者却是个长长三十几位的数!
倘若一个小房间只能存3位数，那前者用一个小房间就够了，而后者要用足足十几个小房间!

因此，不同的数值在内存中占用的存储空间大小是不一样的，有的占用空间很小，有的很大。

这种对房间（存储空间）大小需求的不同，就产生了不同的变量类型（be like 我们说的标准间、套房……住不同的客户）

计算机通过观察到不同的变量类型得知需要为不同的客户分配多大的存储空间。

那计算机是如何观察到变量类型的呢？引入一个新的概念——标记符

标记符：用于定义不同的变量，标记符固定不变，计算机看到不同的标记符，产生不同的理解。

常见的标记符包括int、char、long、short……

比如，

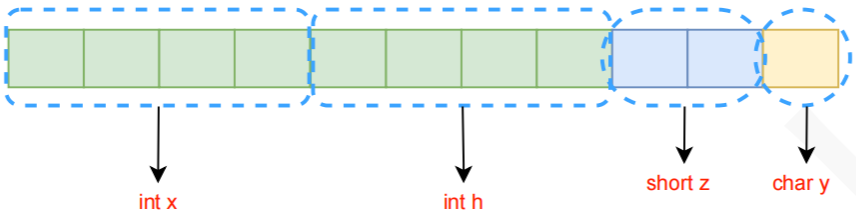
int x，标记符为int，表示定义了一个int类型的变量，变量名为x，在内存中需要占用4个房间

int h，标记符为int，表示定义了一个int类型的变量，变量名为h，在内存中需要占用4个房间

short z，标记符为short，表示定义了一个short类型的变量，变量名为z，在内存中占用2个房间

char y，标记符为char，表示定义了一个char类型的变量，变量名为y，在内存中需要占用1个房间

总结，变量类型就是来表示不同变量（或者说同行旅客）的特点，就和“情侣”“家庭”一样，是一类人（一类数据）的代称！



变量类型的运用规范

不同类型的变量除了**占用的存储空间不同，存的内容类型也不同！**

有的变量类型只允许存整数，有的不止允许存整数，还允许存小数，有的主要以存字符为主。

c语言的基本数据类型（变量类型）如下所示，不用记，了解即可，我们常用的就是int，char，short。

	数据类型	长度（字节）	取值范围
整型	int	4	-2147483648~2147483647
	short	2	-32768~32767
	long	4	-2147483648~2147483647
	long long	8	-2^{63}~$2^{63}-1$
	unsigned int	4	0~4294967295
字符型	char	1	-128~127
浮点数型	float	4	$-3.4*10^{38}$ ~ $3.4*10^{38}$
	double	8	$-1.8*10^{1024}$ ~ $1.8*10^{1024}$

变量如何赋值?

变量=whatever，一个可变化的量

那么变量赋值就是来规定此时变量的值为多少?

我们在计算机中用 = 表示赋值，举例

x = 1000，表示把1000赋值给变量x，

h=888，表示把888赋值给变量h，

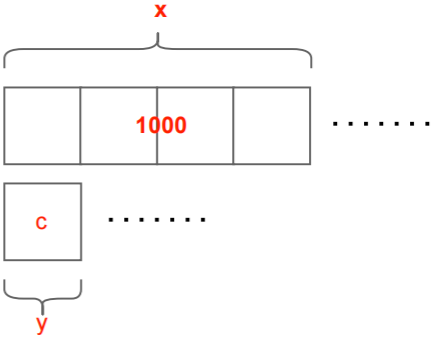
y= ‘c’，表示把字符c赋值给变量y，注意，字符一定要有"

结合刚才介绍的变量类型一起来看，

int x = 1000，表示定义了一个int类型的变量，变量名为x，并且把1000赋值给变量x

char y = ‘ c ’，表示定义了一个char类型的变量，变量名为y，并把字符c赋值给变量y

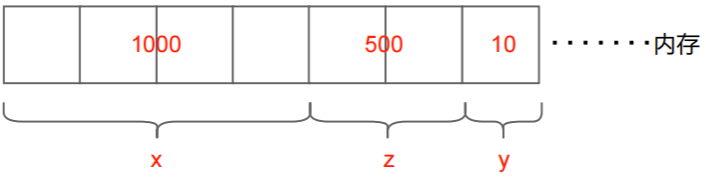
……



以下面的代码，实操一下

```
int main(){
    int x=1000; //定义了一个int类型的变量x并赋值1000
    short z=500; //定义了一个short类型的变量z并赋值500
    char y=10; //定义了一个char类型的变量y并赋值10
    return 0;
}
```

内存和变量



在这个例子中，还有一些代码规范可以注意：

1. 一般情况下，一行只需写一个语句，以 英文分号 “;” 结尾！注意，一定是英文分号！
2. 每一行语句分号后跟着的双斜杠 “//” 也叫注释符，是给人看的，计算机看不懂，会直接略过//及//后面的内容！
3. 每一行语句前的空格【可有可无】，仅是美观使用

```
int main(){
int x=1000;//定义了一个int类型的变量x并赋值1000
short z=500;//定义了一个short类型的变量z并赋值500
char y=10;//定义了一个char类型的变量y并赋值10
return 0;
}
```

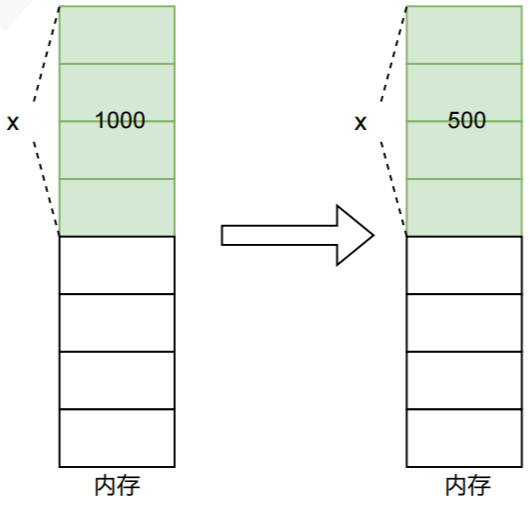
```
int main(){
    int x=1000;//定义了一个int类型的变量x并赋值1000
short z=500;//定义了一个short类型的变量z并赋值500
    char y=10;//定义了一个char类型的变量y并赋值10
return 0;
}
```

（两种书写方式，无论空格与否均可）

变量赋值后如何修改变量呢?

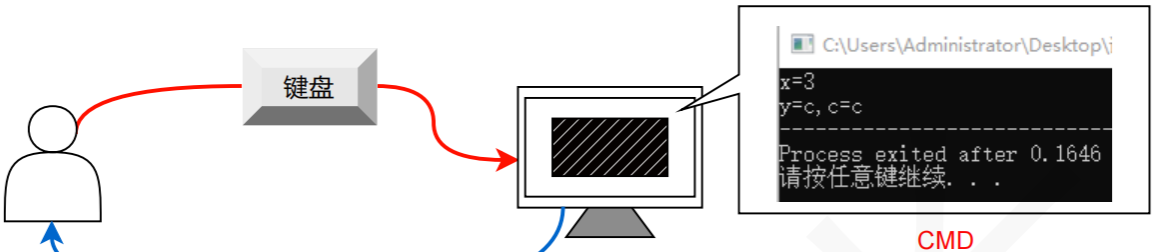
变量是一个可以变化的量，赋值后依然可以变化，在代码中多增加一条语句就可以搞定！

```
int main(){
    int x=1000;//定义了一个int类型的变量x并赋值1000
    x=500; //修改x的值，赋值500
    return 0;
}
```



输入输出

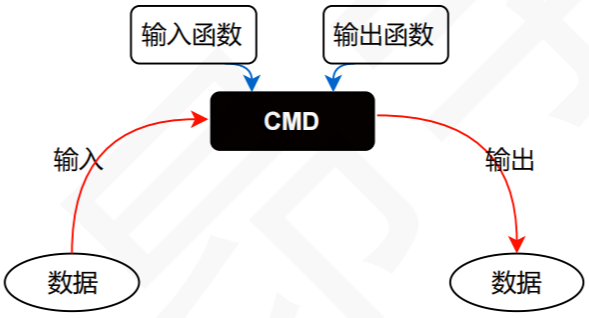
首先，理解什么叫输入输出。
输入输出，就是我们与计算机实现交流的过程
输出，通俗来讲，就是把数据（比如数字、字符等）显示在你的电脑屏幕上
输入，就是获取你在键盘上输入的数据



CMD

当然，在电脑中输入输出的实现必须借助工具，就类似于我们要使用电脑和别人沟通的时候，要先装qq或者微信一类的软件
我们在C语言的输入输出时，也同样需要一个工具，就是我们的CMD（就是上图中的小黑框）

小黑框可以显示我们想要输出的结果，也可以向电脑输入我们用键盘敲的数据



而小黑框的背后，就是我们的输入函数和输出函数（函数就是来实现功能的）
借助输入函数和输出函数，数据在电脑中的输入和输出得以实现并在CMD中显示

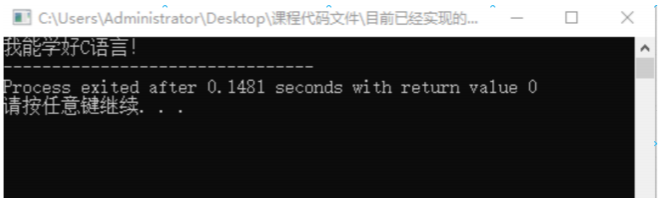
输出函数

最常用的标准输出函数是printf，结合以下例子，来掌握printf函数的基本用法

例1 printf("我要学好C语言!")

```
#include<stdio.h>
int main(){
    printf("我能学好C语言！");
    return 0;
}
```

输出代码



CMD显示结果

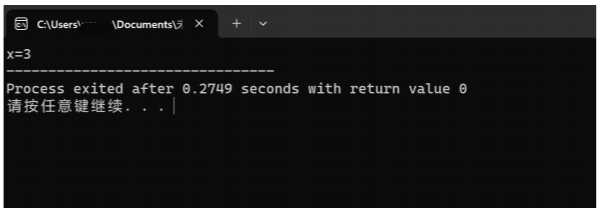
总结：printf("")会直接输出""中的内容

练习：printf("hello world") 输出的结果是：

例2 printf(“x=%d”,x);

```
#include<stdio.h>
int main(){
	int x=3;
	printf("x=%d",x);
	return 0;
}
```

输出代码



CMD显示结果

为什么输出（显示）的结果是3?
介绍一个新的概念——占位符（%格式字符）

int x=3;
printf(“x=%d”,x);

占位符，顾名思义，就是占着位置的符号，常见的占位符包括%d、%c、%f
在""中，除占位符外，其他内容均原封不动输出（即 x= 原封不动输出）
而占位符将会被""外的变量x的值所替代（即%d被变量x的值替代后输出）

int x=3;
printf(“x=%d”,x);

总结：占位符不直接输出，而是被随后跟着的变量的值所取代后输出
并且，不同类型的变量，使用于不同的占位符
一般整型用%d，字符型用%c，浮点型用%f

练习：

打印整数

int i = 10;

printf("i = %d", i);

输出:

打印字符:

char c = 'A';

printf("c = %c", c);

输出:

打印浮点数:

float x = 3.14;

printf("x = %f", x);

输出:

例3 printf("y=%c,c=%c",y,y);

```
#include<stdio.h>
int main(){
	char y='c';
	printf("y=%c,c=%c",y,y);
	return 0;
}
```

输入代码

CMD显示结果

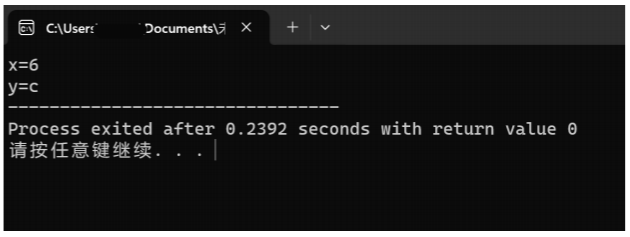
char y='c';
printf("y=%c,c=%c",y,y);

总结：占位符可以有多个，后面的变量值依次替换前面的占位符即可。

例4 printf("x=%d\ny=%c",x,y);

```
#include<stdio.h>
int main(){
	int x=6;
	char y='c';
	printf("x=%d\ny=%c",x,y);
	return 0;
}
```

输入代码



CMD显示结果

```
int x=6;
char y='c';
printf("x=%d\ny=%c",x,y);
```

换行符

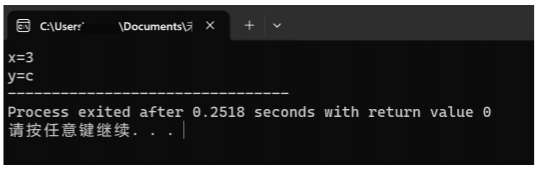
总结：换行符\n 可以使得语句换行

输入函数

最常用的标准输入函数是scanf()，结合以下例子，来掌握scanf()函数的基本用法

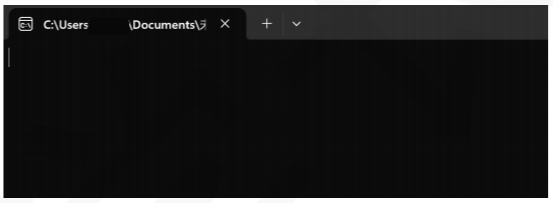
```
#include<stdio.h>
int main(){
    int x=3;
    char y='c';
    printf("x=%d\ny=%c",x,y);
    return 0;
}
```

输出代码



CMD显示结果

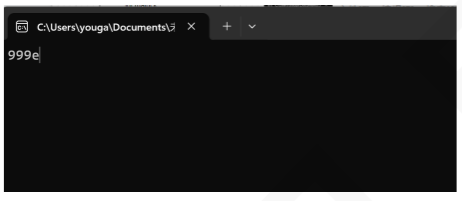
```
#include<stdio.h>
int main(){
    int x=3;
    char y='c';
    scanf("%d%c",&x,&y);
    printf("x=%d\ny=%c",x,y);
    return 0;
}
```



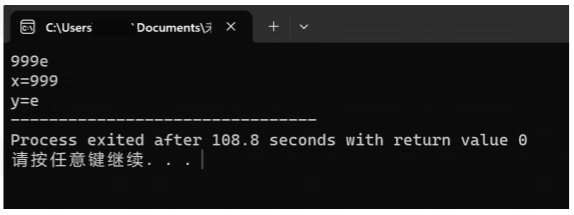
CMD显示画面

接下来，我们在代码中新增一个**输入语句**

此时，我们在CMD中随机输入数据999e



CMD显示结果



CMD显示结果

最终显示结果x=999
y=e

新增了一条输入语句后，最终输出的结果完全不相同，为什么?

在没有输入语句前，
输出的结果由""内的内容和替代占位符的变量值所决定

```
int x=3;
char y='c';
printf("x=%d\ny=%c",x,y);
```

而输入语句的加入，改变了原来的代码逻辑

```
int x=3;
char y='c';
scanf("%d%c",&x,&y);
printf("x=%d\ny=%c",x,y);
```

地址符

地址符的作用在于，使得我们输入的数据找到它在内存中该存放的位置
比如，我们输入999e，数据999会根据&x找到自己的位置，
数据e会根据&y找到自己的位置

[理解]
还记得内存=酒店，存储空间=房间，变量=n个房间（某块存储区域），数据=人吗?
&地址符使得酒店中的每一个房间有了自己的名字。
当旅客（输入的数据）看到地址符的时候，就会知道：我应该入住哪一个房间（我们应该入住哪几个房间）
因此，当人进入到自己的房间后，这几个房间（这块存储区域）就会被入住的人（也就是输入的数据）占领，
这个时候，变量的值就随着新入住的人数发生了变化！x不再等于赋值的3，而是新输入的999！

条件判断

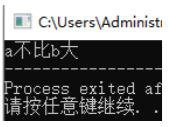
if else

```
#include<stdio.h>
int main(){
    int a=5;
    int b=8;
    if(a>b){
        printf("a=%d更大",a);
    }
    else {
        printf("a不比b大");
    }
    return 0;
}
```

括号里面是条件判断语句
满足a>=b时，我们称判断结果为true
执行if()后面的执行体

{}之间的内容称为执行体

不满足a>=b时，我们称判断结果为false
执行else后面跟着的执行体



该代码运行结果如上

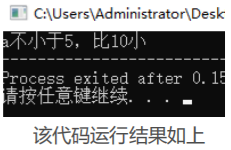
if else +else if

```
#include<stdio.h>
int main(){
    int a=5;
    int b=5;
    if(a>b){
        printf("a=%d更大",a);
    }
    else if(a==b){
        printf("一样大");
    }
    else{
        printf("b=%d更大",b);
    }
    return 0;
}
```

该代码运行结果如上

else if可以在其中反复使用

```
#include<stdio.h>
int main(){
    int a=5;
    if(a<0){
        printf("a比0小");
    }
    else if(a<5){
        printf("a不小于0，比5小");
    }
    else if(a<10){
        printf("a不小于5，比10小");
    }
    else if(a<15){
        printf("a不小于10，比15小");
    }
    else{
        printf("a不小于15");
    }
    return 0;
}
```



该代码运行结果如上

常见的运算符

赋值符号'='以及判断大小符号'<'其实都是运算符，常见的运算符如下所示。

算术运算符

下表显示了 C 语言支持的所有算术运算符。假设变量 **A** 的值为 10，变量 **B** 的值为 20，则：

运算符	描述	示例
+	把两个操作数相加	A + B 将得到 30
-	从第一个操作数中减去第二个操作数	A - B 将得到 -10
*	把两个操作数相乘	A * B 将得到 200
/	分子除以分母	B / A 将得到 2
%	取模运算符，整除后的余数	B % A 将得到 0
++	自增运算符，整数值增加 1	A++ 将得到 11
--	自减运算符，整数值减少 1	A-- 将得到 9

关系运算符

下表显示了 C 语言支持的所有关系运算符。假设变量 A 的值为 10，变量 B 的值为 20，则：

运算符	描述	示例
==	检查两个操作数的值是否相等，如果相等则条件为真。	(A == B) 为假。
!=	检查两个操作数的值是否相等，如果不相等则条件为真。	(A != B) 为真。
>	检查左操作数的值是否大于右操作数的值，如果是则条件为真。	(A > B) 为假。
<	检查左操作数的值是否小于右操作数的值，如果是则条件为真。	(A < B) 为真。
>=	检查左操作数的值是否大于或等于右操作数的值，如果是则条件为真。	(A >= B) 为假。
<=	检查左操作数的值是否小于或等于右操作数的值，如果是则条件为真。	(A <= B) 为真。

逻辑运算符

下表显示了 C 语言支持的所有关系逻辑运算符。假设变量 A 的值为 1，变量 B 的值为 0，则：

运算符	描述	示例
&&	称为逻辑与运算符。如果两个操作数都非零，则条件为真。	(A && B) 为假。
||	称为逻辑或运算符。如果两个操作数中有任意一个非零，则条件为真。	(A || B) 为真。
!	称为逻辑非-------运算符。用来逆转操作数的逻辑状态。如果条件为真则逻辑非运算符将使其为假。	!(A && B) 为真。

赋值运算符

下表列出了 C 语言支持的赋值运算符：

运算符	描述	示例
=	简单的赋值运算符，把右边操作数的值赋给左边操作数	C = A + B 将把 A + B 的值赋给 C
+=	加且赋值运算符，把右边操作数加上左边操作数的结果赋值给左边操作数	C += A 相当于 C = C + A
-=	减且赋值运算符，把左边操作数减去右边操作数的结果赋值给左边操作数	C -= A 相当于 C = C - A
*=	乘且赋值运算符，把右边操作数乘以左边操作数的结果赋值给左边操作数	C *= A 相当于 C = C * A
/=	除且赋值运算符，把左边操作数除以右边操作数的结果赋值给左边操作数	C /= A 相当于 C = C / A
.....	.....	.....

另外，还有位运算符，一般不怎么用，这里不再赘述，感兴趣的同学可以自己康康。

循环

当有需要重复运行的语句时，使用循环能让我们少写很多重复的代码。

如果，我们要输出十句“我要学好C语言！”
则要写printf('我要学好C语言！')这一语句10次
但一旦我们引入了循环，则只需要一个语句就可以搞定！

while循环

下面我们来看个while循环的例子，c语言代码如下。

```
#include<stdio.h>
int main(){
    int x=3;
    while(x<10){
        printf("%d\n",x);
        x++;
    }
    return 0;
}
```

while后面的**括号里内容**为条件判断语句
x<10条件成立时，判断结果为true，
执行后面的执行体

执行体

条件不成立时，判断结果为false，
跳出while循环，
执行后面的语句

该代码运行结果如上

do-while循环

do-while循环原理与while循环类似，区别是先执行，再判断，用的比较少下面我们来看个例子，c语言代码如下。

```
#include<stdio.h>
int main(){
    int x=3;
    do{
        printf("%d\n",x);
        x++;
    }while(x<10);
return 0;
}
```

该代码运行结果如上

continue

当我们想要跳过多轮循环中的某一轮时，就可以使用continue,在while,for,do-while中都可以使用

```
#include<stdio.h>
int main(){
    int x=3;
    while(x<10){
        if(x==5){
            x++;
        continue;
        }
        printf("%d\n",x);
        x++;
    }
    return 0;
}
```

该代码运行结果如上

break

当我们想要满足某个条件时就直接结束循环，就可以使用break,在while,for,do-while中都可以使用，示例如下。

```
#include<stdio.h>
int main(){
    int x=3;
    while(x<10){
        if(x==5){
            break;
        }
        printf("%d\n",x);
        x++;
    }
    return 0;
}
```

该代码运行结果如上

for循环

现在我们现在想依次输出10个元素1~10。。

```
#include<stdio.h>
int main(){
    int i=1;
    while(i<=10){
        printf("%d\n",i);
        i++;
    }
    return 0;
}
```

上述的语句可以替换成for循环执行

```
#include<stdio.h>
int main(){
    int i=1;
    for(i;i<=10;i++){
        printf("%d\n",i);
    }
    return 0;
}
```

for循环和while循环的区别就在于for后面的（）中分成了三个部分

```
for( i=0;i<10;i++)
```

给i赋值　条件判断　每轮循环执行完修改i的值

```
#include<stdio.h>
int main(){
    int i=3;
    for(i=1;i<=10;i++){
        printf("%d\n",i);
    }
    return 0;
}
```

所以也可以写成这样。

```
#include<stdio.h>
int main(){
    for(int i=1;i<=10;i++){
        printf("%d\n",i);
    }
    return 0;
}
```

或者这样。

该代码运行结果如上

注意，**变量是有作用范围的**，在函数里定义的变量只在本函数里有效，出了本函数就没人认识了
而在for()里定义的变量只在本for循环里有效，出了for循环就没人认识啦

变量和函数

我们在学初中数学的时候其实就已经接触了变量和函数这俩概念。

例如 **y=x**， 这是个正比例函数，其中y是因变量，x是自变量。

函数的好处在于，把y和x的关系封装了起来，我们只要给x特定的值，输入到函数中，就可以输出对应的y值，
例如，x=3时，y=3；x=100时，y=100。

变量x —输入→ **函数f(x)** —输出→ **变量y**

计算机沿用了这一概念，当我们想要完成某个任务时，就需要去写一个函数
在这个函数里写许多条执行语句，执行语句可以理解为命令。
类似于我们军训时想要完成点名这个任务，教官会喊出“立正、稍息、向右转、报数！

```
点名（）{
    立正;
    稍息;
    向右转;
    报数!
}
```

"点名"是这个函数的函数名，两个大括号之间的部分被称为函数体，函数体由一条条执行语句构成。
把一系列操作封装在一个函数中的好处是：
我们省去大量重复的代码，当其他函数需要实现类似的功能时，没必要从头再写一遍，直接调用其他实现了该功能的函数即可。
比如我们想让计算机帮我们完成一个二次函数的运算，f(x)=x^2+5x+1，其中，x作为输入，将x带入f(x)，即可得到输出结果.

```
int f(int x){
    int y=0;
    y=x^2+5x+1;
    return y;
}
```

注：这里只是示例，严格来说，写成x*x+5*x+1

当我们想在main函数完成多个f(x)这样的计算时，就没必要每次都从头到尾把这个函数写一遍，而是直接调用就行。

```
int main( ){
    int a,b,c;
    a=5^2+5*5+1;
    b=6^2+5*6+1;
    c=7^3+5*7+1;
    return 0;
}
```

→

```
int main( ){
    int a,b,c;
    a=f(5);
    b=f(6);
    c=f(7);
    return 0;
}

int f(int x){
    int y=0;
    y=x^2+5x+1;
    return y;
}
```

接下来我们康康一个函数长什么样子。

函数名，用来区分不同的函数，
函数名最好能表达出这个函数的功能。
例如，add，一看就知道是加法，
sub，一看就知道是减法。

输入变量的类型　输入变量名

```
int f(int x){
    int y=0;
    y=x^2+5x+1;
    return y;
}
```

我们最后的计算结果，要返回给调用f(x)的函数
return 就是返回的意思，这里的y就是返回的结果
需要在函数名前标识返回变量的类型

因此，我们就知道了函数格式三板斧，函数名，输入（参数），输出。

对于有的问题，仅有一个输入还不够，比如要计算x_1和x_2的和，这个时候就需要两个输入。

```
int add(int x1,int x2){
        int y=x1+x2;
        return y;
}
```

其中，add是函数名，输入的变量是x_1,x_2，变量y保存了x_1+x_2的结果，return 会将计算好的结果返回给调用add的函数。

```
void  sayhello(){
        printf("hello!");
}
```

函数也不一定非要有输入，也不一定非要返回结果，
这个时候我们空着函数名后面的括号，返回类型写成void（空类型）即可。

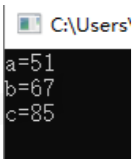
实参和形参

为了后续便于理解"指针"，"引用"等概念（引用严格来说属于C++，但是许多教材都直接混用，因此我们也要习惯C语言代码中直接出现引用的用法），我们得理解函数调用时参数传递的过程，示例如下。

```
#include<stdio.h>
int f(int x){
        int y=0;
        y=x*x+5*x+1;
        return y;
}
int main( ){
        int a,b,c;
        a=f(5);
        b=f(6);
        c=f(7);
        printf("a=%d\n",a);
        printf("b=%d\n",b);
        printf("c=%d\n",c);
        return 0;
}
```

f()函数

main()函数



该代码运行结果如上

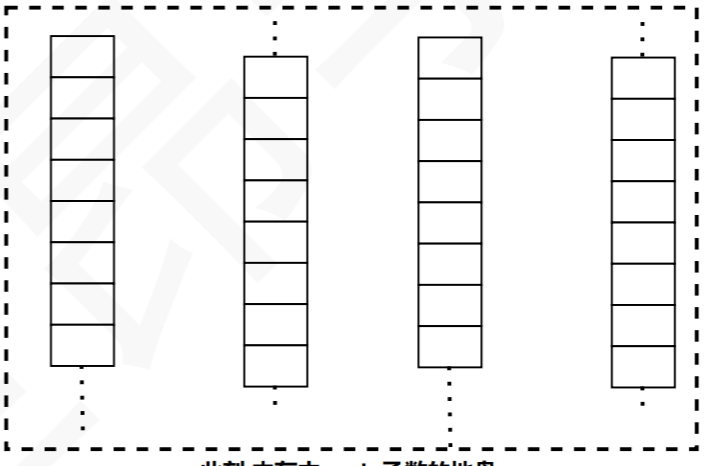
整个代码的运行过程如下所示：

main函数是我们整个程序的入口函数，
就像我们去公园、进校园都必须从入口通过，
所以main函数是第一个运行的函数。

首先执行main函数部分

```
#include<stdio.h>
int main( ){
        int a,b,c;
        a=f(5);
        b=f(6);
        c=f(7);
        printf("a=%d\n",a);
        printf("b=%d\n",b);
        printf("c=%d\n",c);
        return 0;
}
```

当我们开始执行main（）时，会在内存中建立属于它的存储区域（地盘）

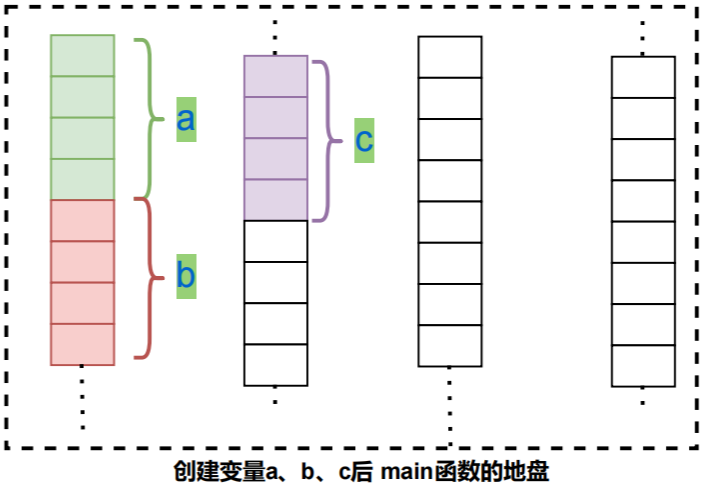


此刻 内存中 main函数的地盘

接着我们创建了3个变量a、b、c

这里我们没有赋初值，所以其存储的数是随机的，
有可能这块区域之前代表的是其他变量，里面还存在其他数，
最好还是赋初值。

```
#include<stdio.h>
int main( ){
        int a,b,c;
        a=f(5);
        b=f(6);
        c=f(7);
        printf("a=%d\n",a);
        printf("b=%d\n",b);
        printf("c=%d\n",c);
        return 0;
}
```



创建变量a、b、c后 main函数的地盘

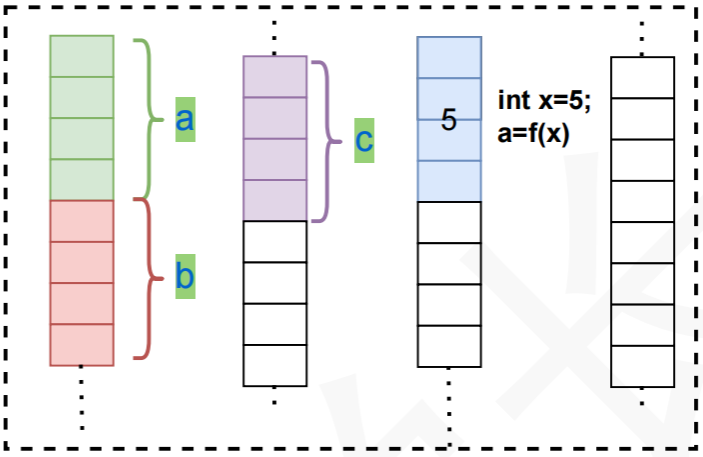
接着执行语句a=(f5)

```
#include<stdio.h>
int main( ){
        int a,b,c;
        a=f(5);
        b=f(6);
        c=f(7);
        printf("a=%d\n",a);
        printf("b=%d\n",b);
        printf("c=%d\n",c);
        return 0;
}
```

这条语句会分好几步执行，首先会执行后面的f(5)。
我们已经知道，调用函数时候需要给函数输入，另一个说法就叫做参数。

a=f(5)

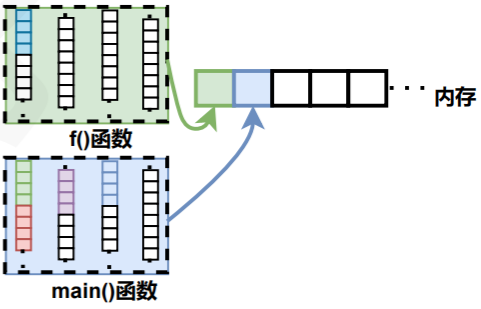
这里的5**我们称之为实参，也就是实际要传入的参数，**



此时 main()函数地盘中的情况

```
#include<stdio.h>
int f(int x){
        int y=0;
        y=x*x+5*x+1;
        return y;
}
int main( ){
        int a,b,c;
        a=f(5);
        b=f(6);
        c=f(7);
        printf("a=%d\n",a);
        printf("b=%d\n",b);
        printf("c=%d\n",c);
        return 0;
}
```

int x=5;
a=f(x)



a=f()的运算需要借助f()函数完成，

f(int x)未被调用时，还未建立它的地盘，

只有被调用后，才会建立其地盘。

因此未被调用时

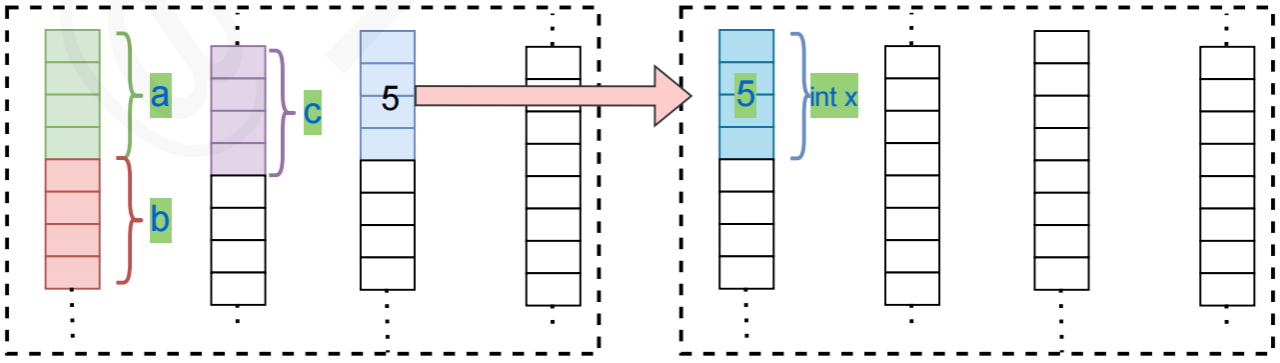
int x只是一个占位符，没有实际在内存中被创建

因此，int x也叫做形参，就是形式上的参数

执行f(5)，意味着main函数调用f()函数，

因此内存给f()函数分配了属于自己的地盘，

接着它会在自己的地盘中创建变量x，并复制main函数中的实参到x中



此时 main()函数地盘中的情况

此时 内存中f()函数的地盘

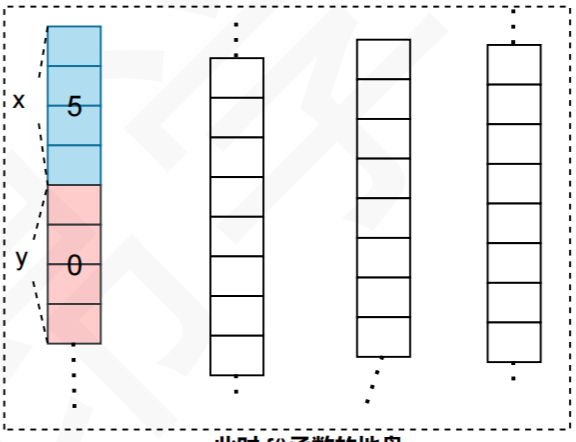
tip：这一步请务必理解透彻！为计算a=f(5)而调用函数f(int x)，
main()函数中的创建的变量5 和 调用的f()函数的变量5 同时存在

接着我们会一条条执行f(5)中的语句

```
#include<stdio.h>
int f(int x){
    int y=0;
    y=x*x+5*x+1;
    return y;
}
int main( ){
    int a,b,c;
    a=f(5);
    [illegible]
}
```

step1 创建变量y，赋值为0

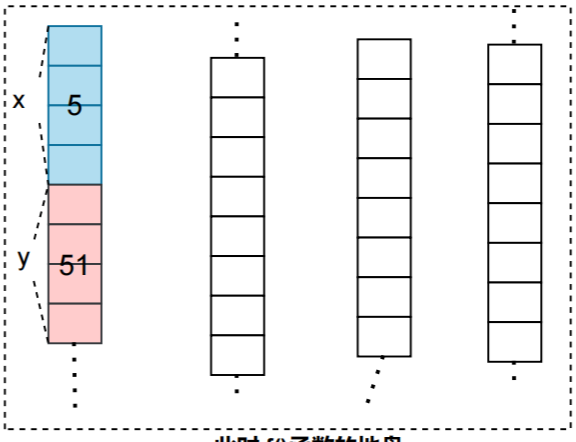
```
#include<stdio.h>
int f(int x){
    int y=0;
    y=x*x+5*x+1;
    return y;
}
int main( ){
    int a,b,c;
    a=f(5);
    [illegible]
}
```



此时 f()函数的地盘

step2 计算x*x+5*x+1，并将结果赋值给y

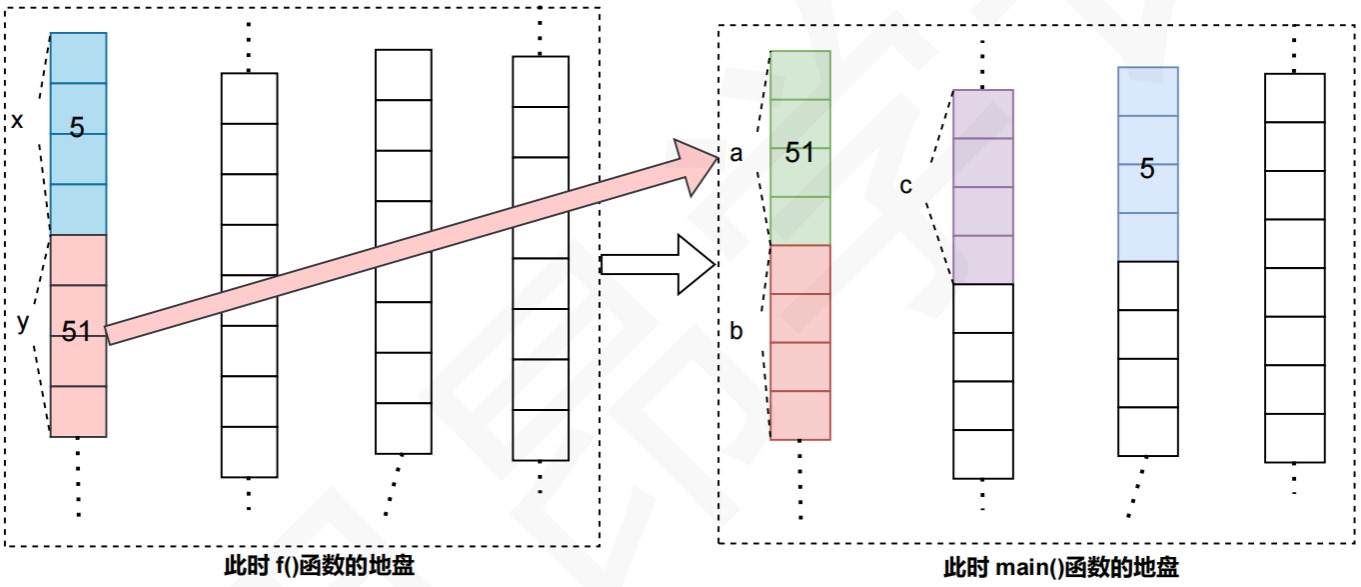
```
#include<stdio.h>
int f(int x){
    int y=0;
    y=x*x+5*x+1;
    return y;
}
int main( ){
    int a,b,c;
    a=f(5);
    [illegible]
}
```



此时 f()函数的地盘

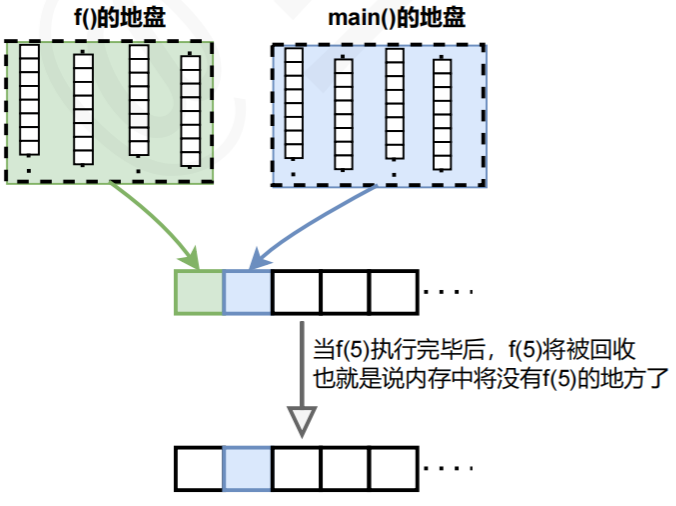
step3 将y的值返回给调用f()的main函数，赋值给a。

```
#include<stdio.h>
int f(int x){
    int y=0;
    y=x*x+5*x+1;
    return y;
}
int main( ){
    int a,b,c;
    a=f(5);
```



此时 f()函数的地盘　　此时 main()函数的地盘

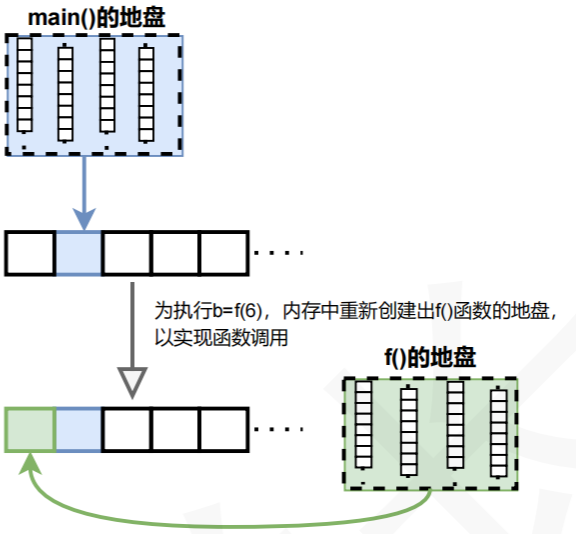
step4 回收f()的地盘，结束。

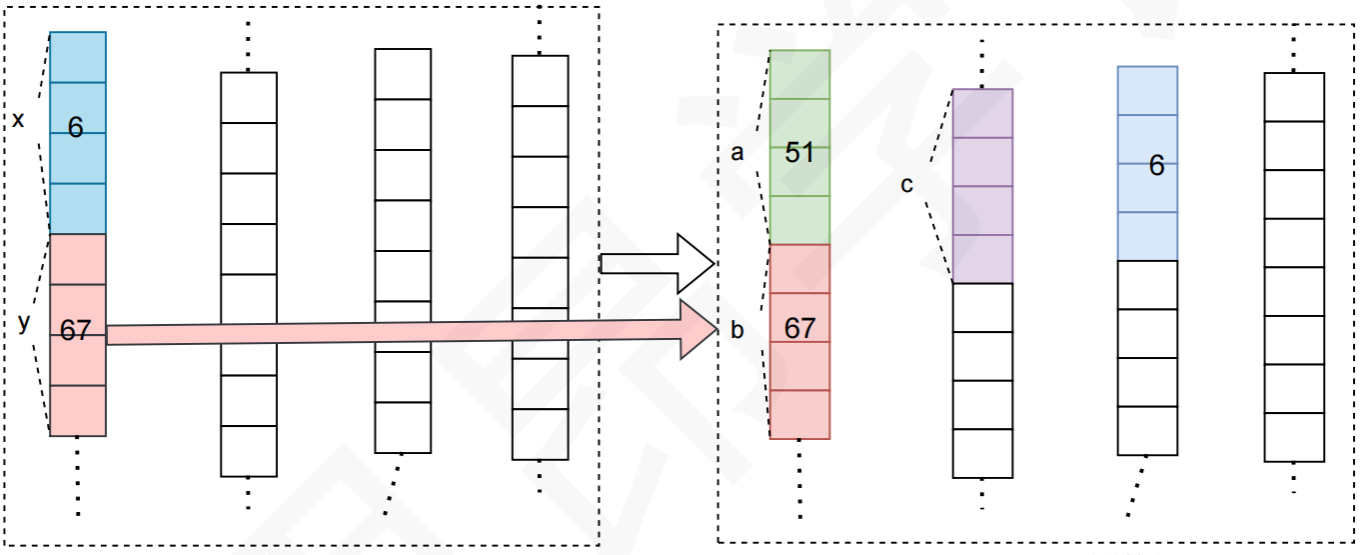


随后执行b=f(6)

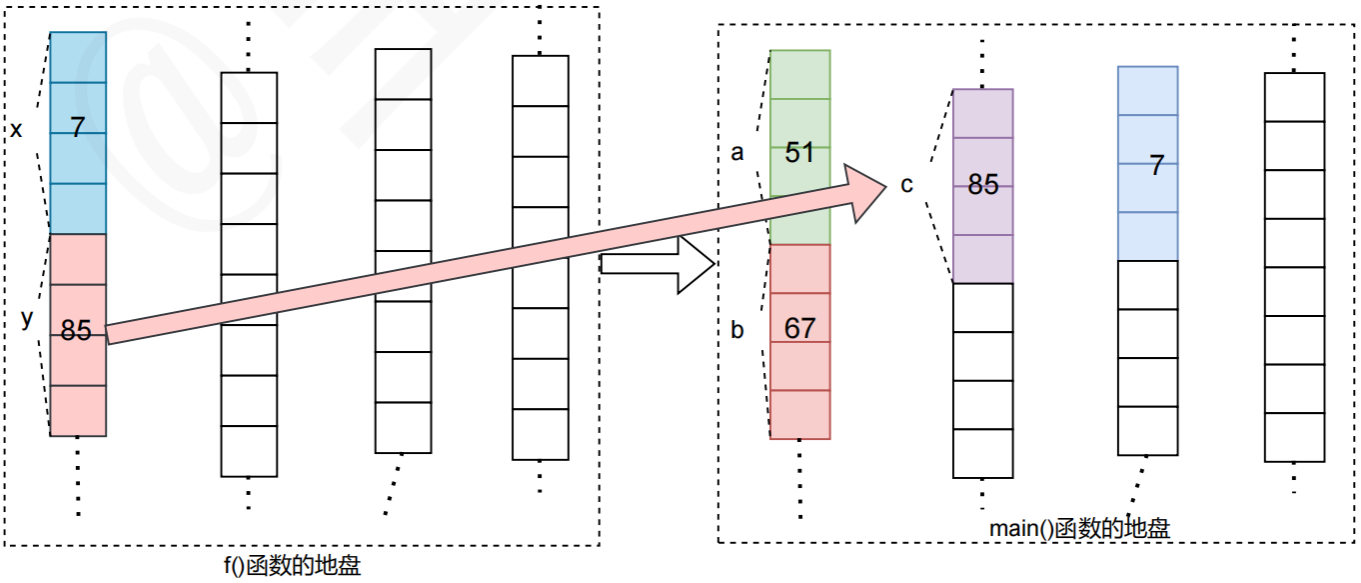
接着我们执行b=f(6)，这条语句的执行过程与a=f(5)类似，首先也会在内存中创建f(6)的地盘，在f(6)的地盘中创建变量x，赋值为6，执行结束后，会将y的值赋值给main函数中的b。

```
#include<stdio.h>
int f(int x){
    int y=0;
    y=x*x+5*x+1;
    return y;
}
int main( ){
    int a,b,c;
    a=f(5);
    b=f(6);
    c=f(7);
    printf("a=%d\n",a);
    printf("b=%d\n",b);
    printf("c=%d\n",c);
    return 0;
}
```





接着执行c=f(7);的原理类似，不再赘述。



注：上述过程中的所有图画并不严谨，只是为了方便小白理解函数调用以及参数传递的过程。