河海大学

第7章 指针

7.1 指针基本概念

7.2 变量与指针

7.3 数组与指针

7.4 指针与函数

7.5 指针与字符串

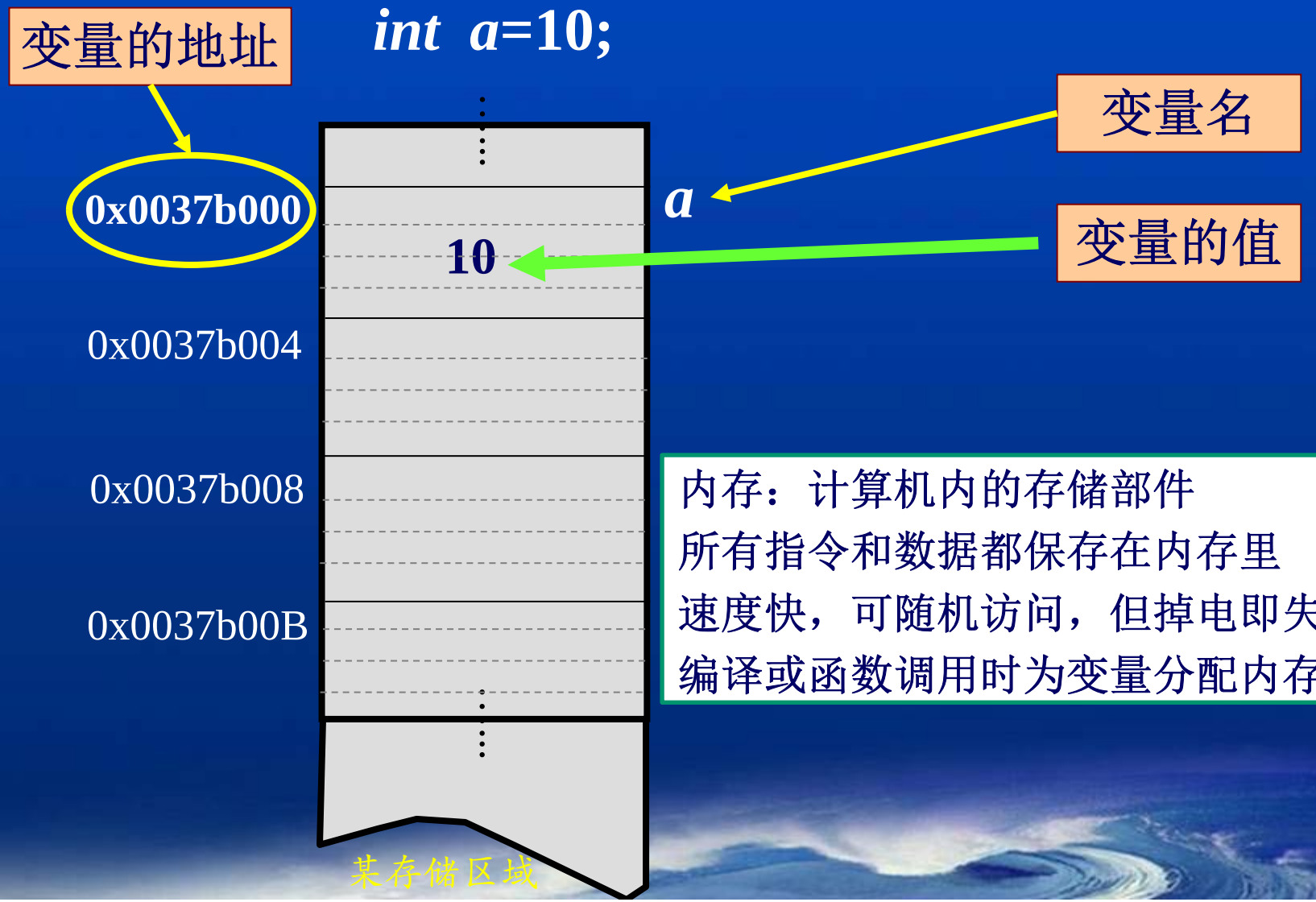
7.7 引用

7.6 动态内存分配

河海大学

7.1 指针基本概念

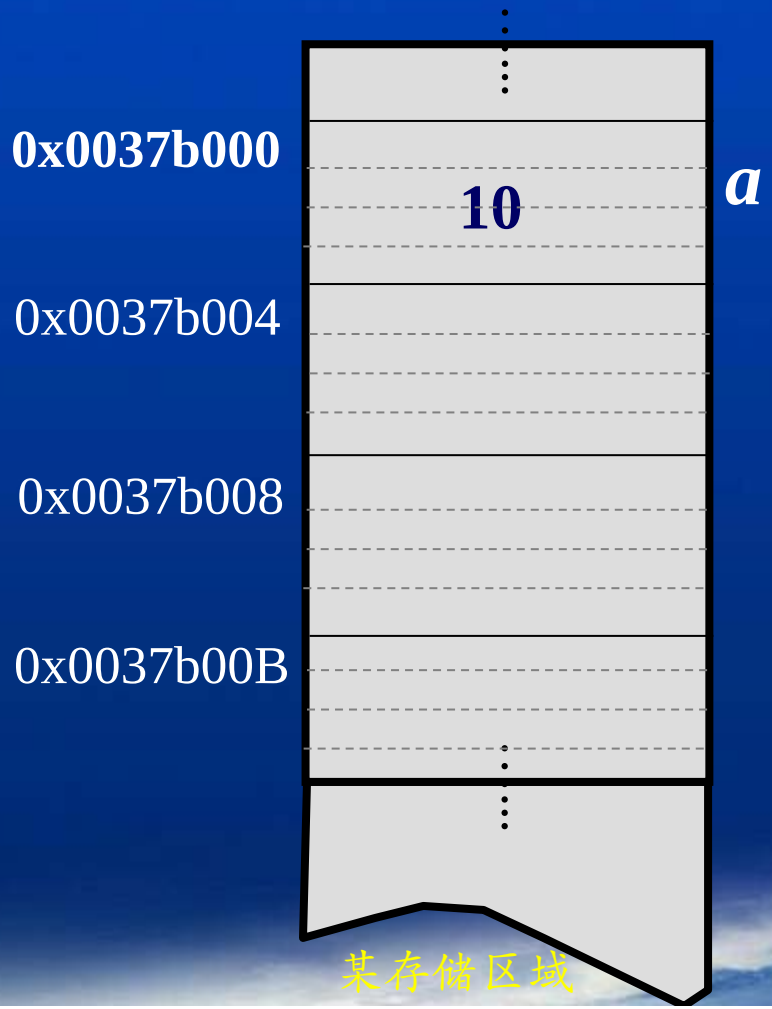
变量 (Variables)与变量的地址 (Address)



内存：计算机内的存储部件
所有指令和数据都保存在内存里
速度快，可随机访问，但掉电即失
编译或函数调用时为变量分配内存单元

变量 (Variables)与变量的地址 (Address)

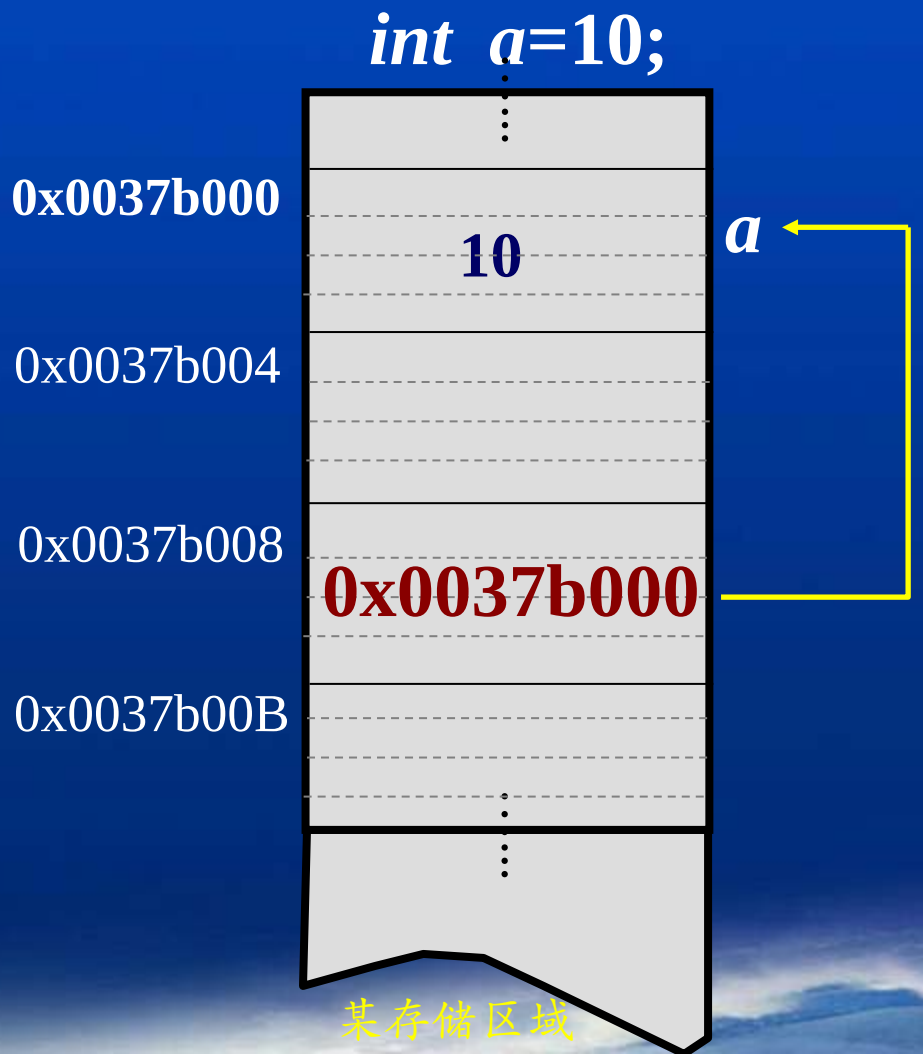
int a=10;



计算机内部对变量的操作都是通过地址进行的，而不是通过变量名，变量名在编译的时候会被替换成变量的地址。

```
scanf("%d", &a);
```

变量 (Variables)与变量的地址 (Address)



间接访问：通过存放变量地址的变量去访问变量

为什么可以直接存取变量，还要间接存取变量呢？

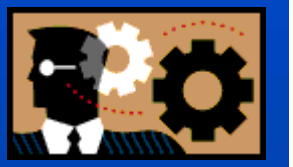
（1）方便传递参数；

如果有一个数组需要通过函数进行处理，我们是将整个数组的元素复制一份传递给函数，还是直接将数组的地址传递给函数呢？

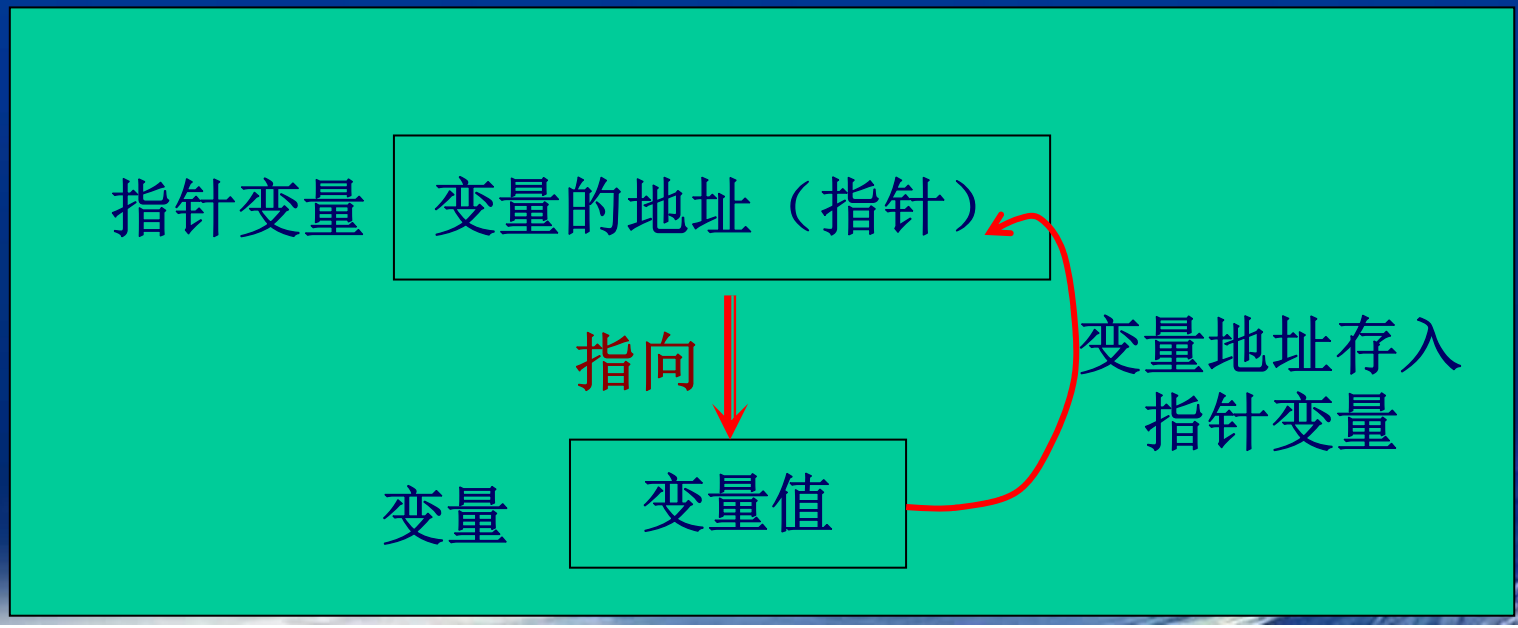
（2）动态分配内存；

编写程序时，很多情况下，并不知道运行时需要多少内存，因为不知道有多少数据需要处理，因此在运行时，通过动态分配内存解决，此时，需要保存动态分配的内存的地址，并通过该地址对分配的空间进行存取。

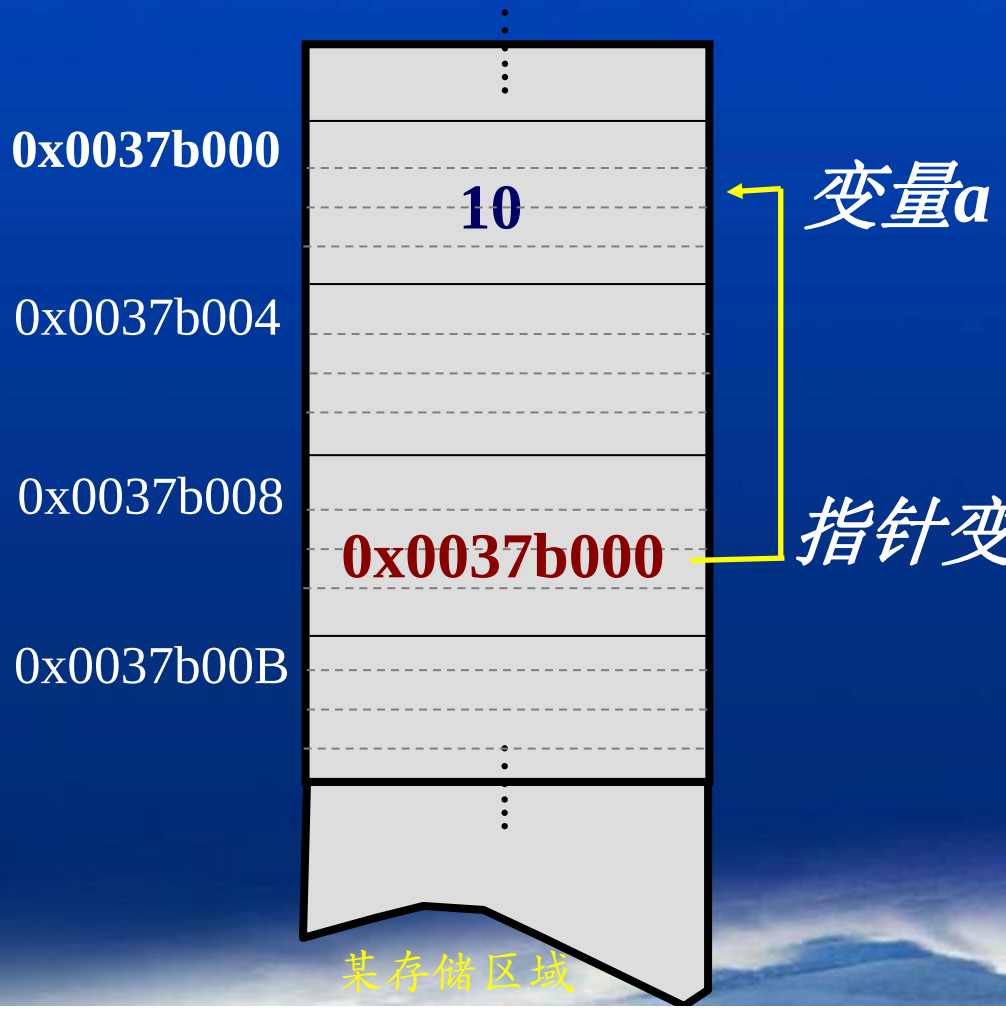
指针（Pointer）的概念



- 什么类型的变量可以存放变量的地址？
- 指针类型——存放地址型数据的特殊数据类型
- 指针变量——专门用于存放地址型数据的变量
- 变量的指针←→变量的地址



int a=10;



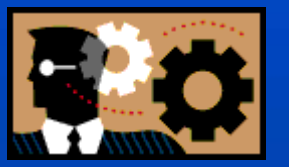
变量a

指针变量p

0x0037b000是变量a的地址或指针。

指针变量p指向变量a。

指针（Pointer）的概念



- 什么类型的变量可以存放变量的地址？
- 指针类型——存放地址型数据的特殊数据类型
- 指针变量——专门用于存放地址型数据的变量
- 变量的指针←→变量的地址

河海大学

7.2 变量与指针

定义指针变量

C规定所有变量在使用前必须先定义，即指定其类型，并分配空间。在编译时按变量类型分配存储空间。对指针变量必须将它定义为指针类型。先看一个具体例子：

```
int i,j;                          //定义整型变量i,j
int *pointer_1, *pointer_2 ; //定义指向整型变量的指针变量
```

定义指针变量的一般形式为

基类型 指针变量名；*

下面都是合法的定义：

float* pointer_3; // pointer_3是指向单精度型数据的指针变量

char* pointer_4; // pointer_4是指向字符型数据的指针变量

请注意： 同时定义多个指针变量时，每个指针变量前都需要带上*。

float *p1, *p2, *p3; //p1,p2,p3都是float类型的指针变量

float *p1, *p2, p3; //p1,p2是float类型指针变量，但是p3是float类型变量。

当定义 一个指针变量 时，尽量将*靠近基类型书写，即**float* p1;**

引用指针变量

有两个与指针变量有关的运算符：

(1) & 取地址运算符。

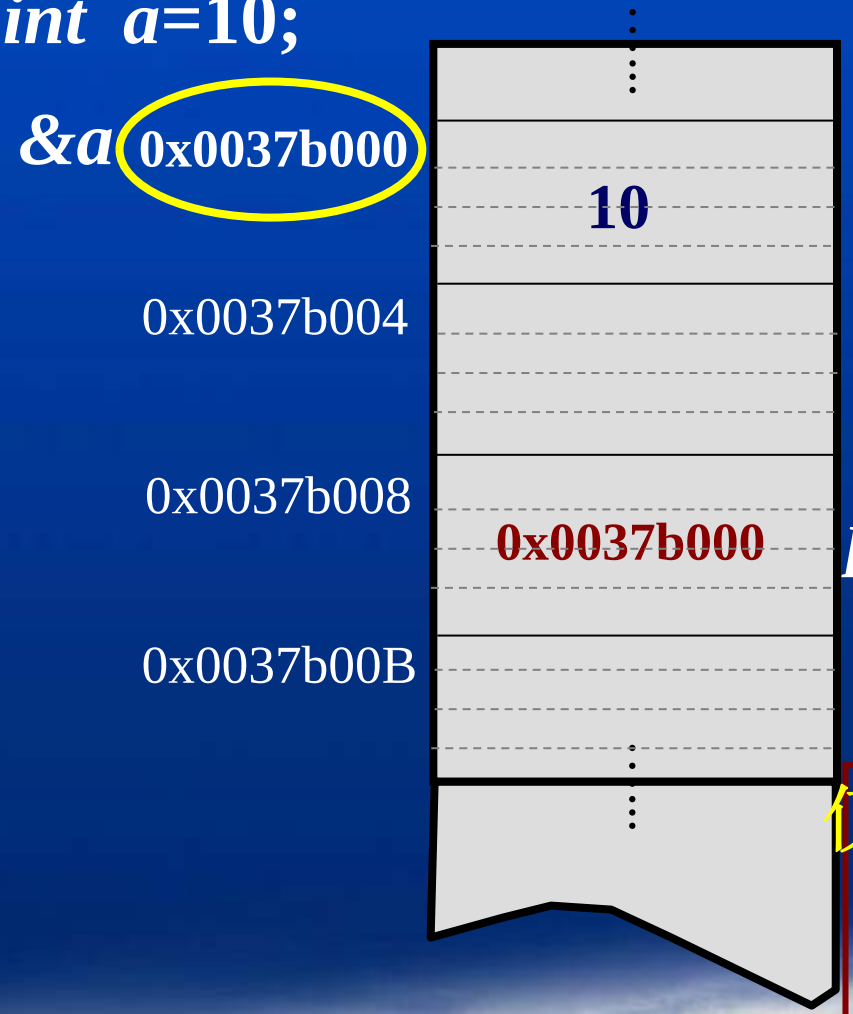
(2) ***指针运算符（或称间接访问运算符）。**

例如： **&a为变量a的地址，*p为指针变量p所指向的存储单元。**

```
int i, j;
int *pi = 0, *pj = 0;
pi = &i;
pj = &j;
i = 100;
printf( “%d”, *pi); //100
*pj = 200;
printf(“%d”, j);  //200
```

指针赋值

int a=10;




```
int  *pa;
pa = &a;
```

仅仅是定义了可以指向*int*型数据的指针变量，但并未指向*a*

使其指向a需对指针变量初始化

例 通过指针访问整型变量。

```
#include <stdio.h>
int main( )
{int a,b;                          //定义整型变量a,b
int *pointer_1,*pointer_2; //定义指针变量*pointer_1,*pointer_2
a=100;b=10;                        //对a,b赋值
pointer_1=&a;                      //把变量 a 的地址赋给pointer_1
pointer_2=&b;                      //把变量 a 的地址赋给pointer_2
printf(“%d %d”,a,b);               //输出a和b的值
printf(“%d %d”,*pointer_1,*pointer_2);  //输出*pointer_1和*pointer_2的值
printf(“%d %d”,pointer_1, pointer_2); //输出地址
return 0;
}
```

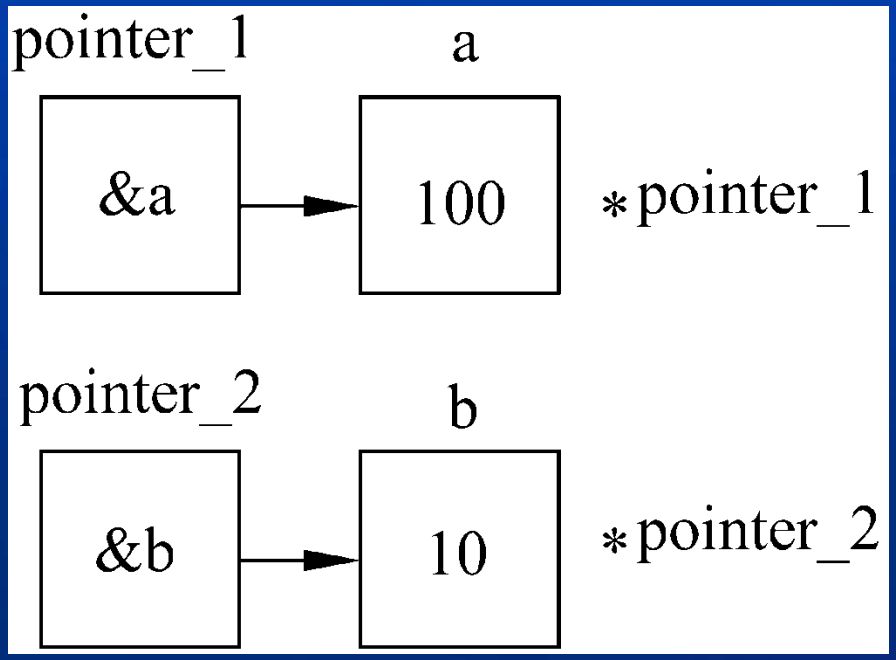
运行结果为

100 10 **(a和b的值)**

100 10 **(*pointer_1和*pointer_2的值)**

0x28fed4 0x28fed0 **(pointer_1和pointer_2的值)**

请对照分析。



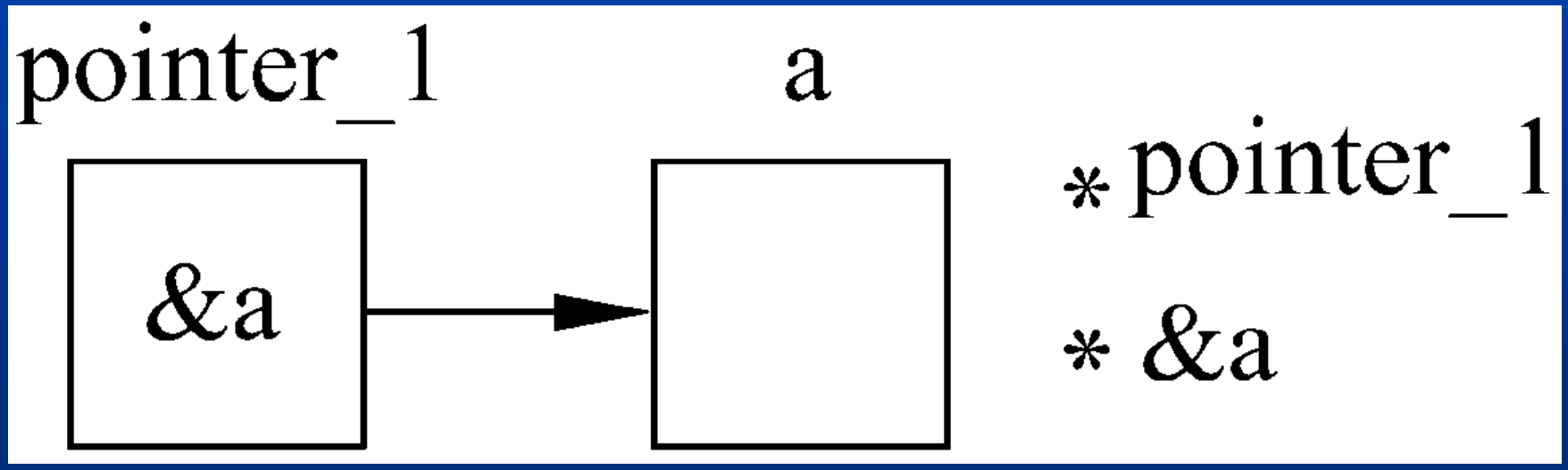
下面对“&”和“*”运算符再做些说明：

(1) 如果已执行了“pointer_1=&a;”语句，请问&*pointer_1的含义是什么？

“&”和“*”两个运算符的优先级别相同，但按自右至左方向结合，因此先进行*pointer_1的运算，它就是变量a，再执行&运算。因此，&*pointer_1与&a相同，即变量a的地址。

如果有pointer_2 =&*pointer_1；它的作用是将&a(a的地址)赋给pointer_2，如果pointer_2原来指向b，经过重新赋值后它已不再指向b了，而也指向了 a 。图 (a)是原来的情况，图 (b)是执行上述赋值语句后的情况。

(2) *&a的含义是什么？先进行&a的运算，得a的地址，再进行*运算，即&a所指向的变量，*&a和*pointer_1的作用是一样的（假设已执行了“pointer_1=&a;”），它们等价于变量a。即*&a与a等价，见图。



指针是有类型的，不同类型的指针不能互相赋值。

```
int* pi;
int i;
float* pf;
float f;
pi = &f; //X 指向float类型的指针不能赋值给int指针变量
pf = &i; //X 指向float类型的指针不能赋值给int指针变量
void* pv = NULL; //无类型指针
pv = pi; //可以把其他类型指针赋值给void*
pi = pv;
//void*赋值给具体类型指针变量时，需要强制类型转换
pv = pf;
pf = pv;
```

分析以下运行结果。

```
int* pi;
int i = 100;
float* pf;
float f = 3.14;
pi = &i;
cout << *pi <<endl;  //100
pi = (int*)&f;  //语法正确
cout << *pi << endl;       //1078523331
char c = ‘a’;
pi = (int*)&c;
cout << *pi;
```

```
100
1078523331
1224065889
```

定义指针类型别名

```
typedef int* PINT;
//定义了一个指向int类型变量的指针类型PINT,就是给int*起了别名.

int main(){
        PINT pi; //和int* pi;效果相同
        int i;
        pi = &i;

        return 0;
}
```

* 是靠近变量还是靠近类型名?

修饰符 * 靠近数据类型，例如：**int* a;** 从语义上讲此写法比较直观，即**a**是**int** 指针类型变量。
弊端是容易引起误解，例如：**int* x, y;** 此处**y**容易被误解为指针变量，其实**y**就是一个**int**类型变量。

若在一行定义多个指针变量，则可以将*靠近变量。

```
char *a, *b, *c;
```

若一行定义一个变量，可以将*靠近类型，如

```
float*  pf;
```

int* pi;和**int *pi;**定义的**pi**是相同的。

常量指针和指针常量

常量指针就是指向常量的指针。

指针常量是指针的值是常量。

常量指针：指向常量的指针

```
#include <stdoi.h>


int main(){
        const int *p; //指向常量的指针p
        const int ci = 100;
        int i;
        p = &i; //p是可以被赋值的
        *p = 200; //错误
}
```

```
[Error] assignment of read-only location '* p'
```

指针常量

```
1  #include <stdio.h>
2
3  int main( ){
4      int* const p;
5      int i;
6      p = &i;
7      printf("%d", *p);
8
9      return 0;
10 }
11
```

const修饰p，因此在定义时就需要对p初始化，以后不能再赋值。

```
[Error] assignment of read-only variable 'p'
```

指针常量

```
#include <stdio.h>


int main(){
        int i;
        int * const p = &i; //p是常量
        *p = 200; //*p可以赋值，即可以修改p指向的内
                  //容的值，但是不能修改p的值
}
```

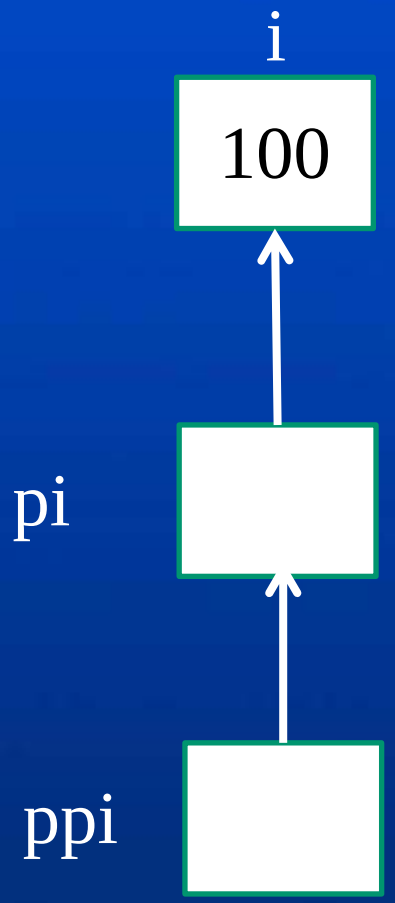
指向常量的指针常量

```
#include <stdio.h>

int main(){
    int i,  j;
    int const * const  p1 = &i;//指向常量的指针常量
    //第一个const修饰*p，第二个const修饰p1
    //const int * const p1 = &i;  也可以

    *p1 = 100; //错误，站在p1的角度，它指向的内容为常量
    i = 100;   //通过i改变自己.
    p = &j; //错误,因为p1被定义为常量,不能被修改.
}
```

二级指针

指向指针变量的指针。

```
int *pi;
int **ppi;

pi = &i;       //pi指向i
ppi = &pi;  //ppi指向pi
*pi = 100;
printf("%d",i); //100
printf("%d",**ppi);//100
```



函数指针

函数指针是指向函数的指针。存放函数的入口地址.

每个函数的代码块都占用一段连续的内存空间.

```
#include <stdio.h>

void f(){
        cout << "ffff" << endl;
}
void g(){
        cout << "gggg" << endl;
}
int main(){
        void (*pf)(); //定义函数指针变量
        pf = f; //直接用函数名赋值
        pf(); //ffff
        (*pf)(); //ffff
        pf = g;
        pf(); //gggg
        (*pf)(); //gggg
        return 0;
}
```

定义函数指针变量时，就是把函数原型中函数名替换成 (*指针名)

函数指针是具有类型的：函数返回值类型，以及形参类型列表确定函数指针类型

```
[Error] invalid conversion from 'int (*)()' to 'void (*)()' [-fpermissive]
```

```
1  #include <iostream>
2  using namespace std;
3
4  void f(){
5      cout << "ffff" << endl;
6  }
7  int g(){
8      cout << "gggg" << endl;
9  }
10 int main(){
11     void (*pf)();
12     pf = f;
13     pf(); //ffff
14     pf = g;
15 }
```

```
1  #include <iostream>
2  using namespace std;
3
4  void f(){
5      cout << "ffff" << endl;
6  }
7  int g(){
8      cout << "gggg" << endl;
9  }
10 int main(){
11     void (*pf)();
12     int (*pg)();
13     pf = f;
14     pf(); //ffff
15     pg = g;
16     pg();
17 }
```

```
#include <stdio.h>
using namespace std;
int findmax(int a, int b){
        if (a>b)
                return a;
        else
                return b;
}

int findmin(int a, int b){
        if (a<b)
                return a;
        else
                return b;
}
```

```
int main(){

        int (*pf)(int, int);
        int a= 100;
        int b = 200;
        pf = findmax;
        printf("%d",pf(a, b));
        pf = findmin;
        printf("%d",pf(a, b));
        return 0;
}
```

使用typedef定义函数指针类型别名

```
#include <stdio.h>


int findmax(int a, int b){
        if (a>b)
                return a;
        else
                return b;
}
```

```
typedef int (*PF)(int, int);

int main(){

        int (*pf)(int, int);
        PF pf1;
        int a= 100;
        int b = 200;
        pf = findmax;
        printf("%d",pf(a, b));
        pf1 = pf;
        printf("%d",pf1(a, b));

        return 0;
}
```

NULL指针和void*类型指针

NULL指针：指针变量中的地址为0，表示不指向任何内存单元，此时该指针称为NULL指针。

void*类型指针：任何类型指针变量都可以赋值给void*类型指针变量。 void*类型指针赋值给其他类型指针变量时必须进行强制类型转换。

河海大学

欢 迎 交 流