程序设计
Programming

Lecture 10: 指针进阶

内容概要

- 指针数组、二级指针和二维数组（重要，熟练掌握）
- 指针作为函数返回值（重要，掌握正确的使用方式）
- 函数指针（了解基本用法）
- 命令行参数（程序运行参数，非常有用！）

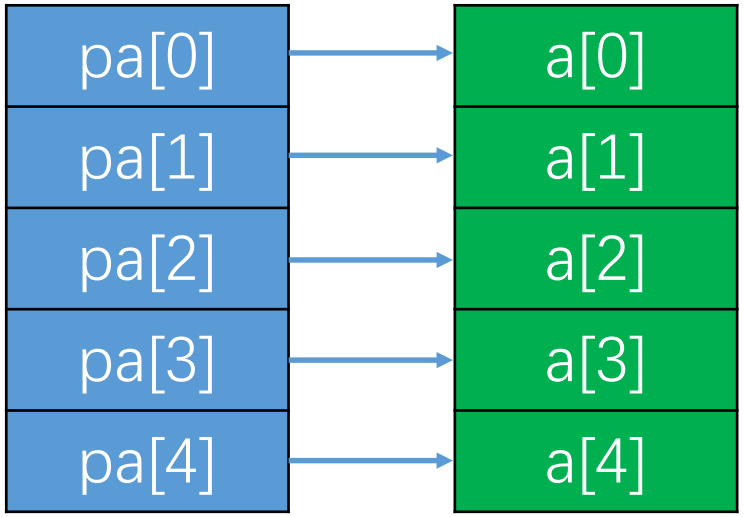
1、指针数组

指针数组

- 类型名 *数组名 [数组长度]
  - ✓数组中的每一个元素都是一个指针，可指向一个“类型名”变量
- char *color[5]
  - ✓定义了5个字符型指针组成的指针数组
  - ✓color[i]可以指向一个字符型变量，即可以用来存储一个字符串（的首地址）
- int *a[10]
  - ✓定义了10个整型指针组成的指针数组
  - ✓a[i]可以指向一个整型变量

整型指针数组

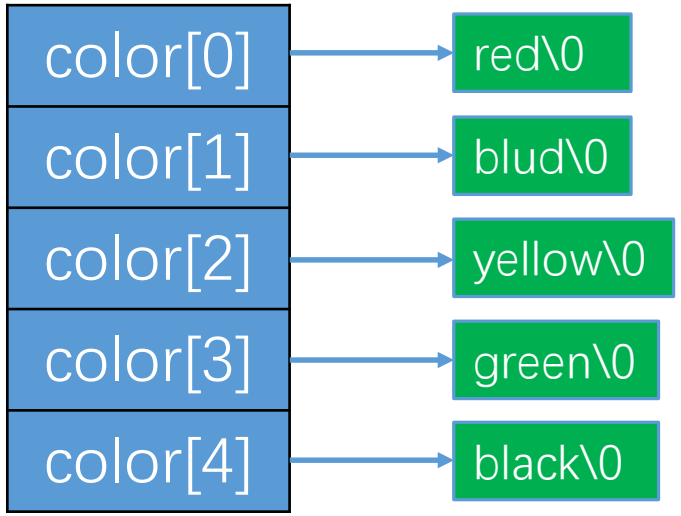
```c
#include <stdio.h>

int main()
{
    int a[5] = {1, 2, 3, 4, 5};
    int *pa[5];
    for(int i = 0; i < 5; i++)
    {
        pa[i] = &a[i];
    }
    return 0;
}

```



字符型指针数组

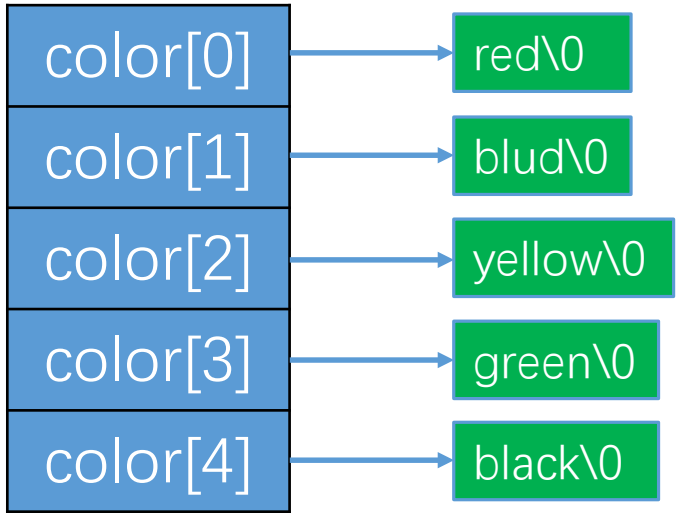
```
char *color[5] = {"red", "blue", "yellow", "green", "black"};
```

输出以字母'b'开头的颜色



字符型指针数组

char *color[5] = {"red", "blue", "yellow", "green", "black"};



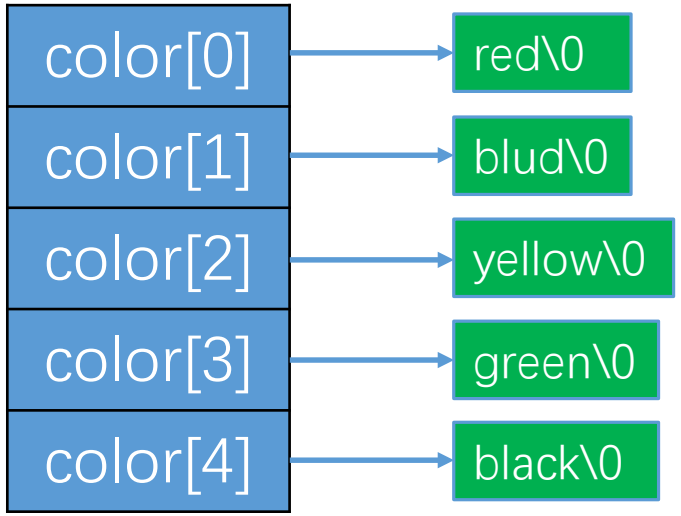
输出以字母'b'开头的颜色

```c
#include <stdio.h>

int main()
{
    char *color[5] = {"red", "blue", "yellow", "green", "black"};
    for(int i = 0; i < 5; i++)
    {
        if(color[i][0] == 'b')
            printf("%s\n", color[i]);
    }
    return 0;
}
```

字符型指针数组

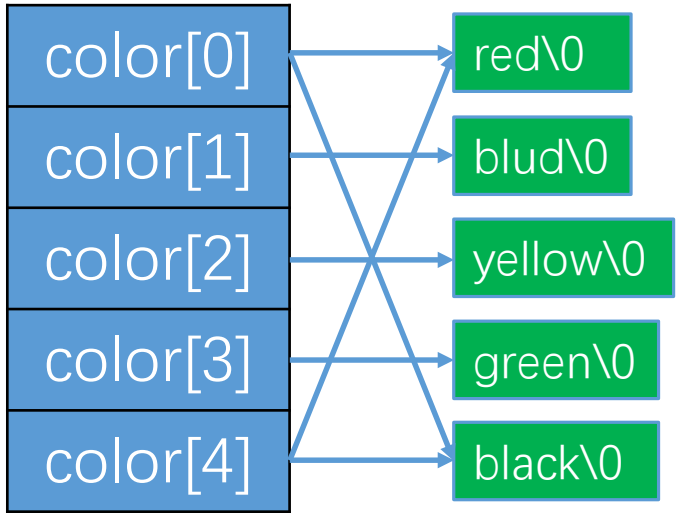
```
char *color[5] = {"red", "blue", "yellow", "green", "black"};
```

交换字符串color[0]和color[4]



字符型指针数组

char *color[5] = {"red", "blue", "yellow", "green", "black"};



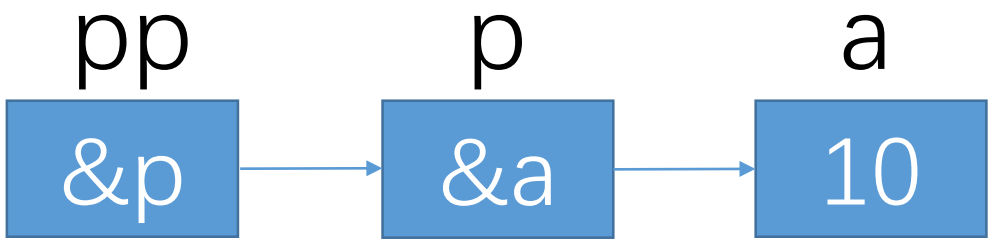
交换字符串color[0]和color[4]

```c
#include <stdio.h>

int main()
{
    char *color[5] = {"red", "blue", "yellow", "green", "black"};
    char *str = color[0];
    color[0] = color[4];
    color[4] = str;
    for(int i = 0; i < 5; i++)
        printf("%s\n", color[i]);
    return 0;
}
```

2、二级指针

二级指针：指向指针的指针

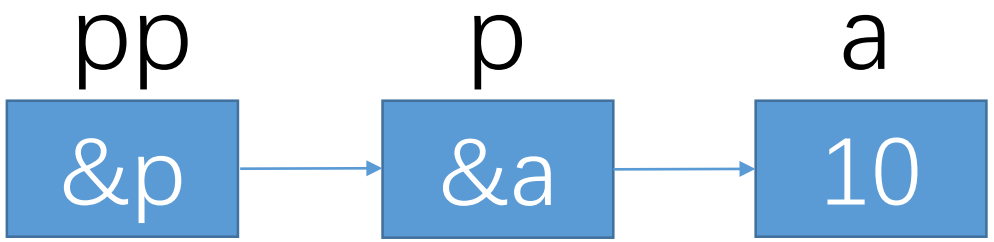
- 类型名 ** 变量名
  - ✓即指向“类型名”指针的指针



- 假设p是一个整型指针
  - ✓int a = 10; int *p = &a;
  - ✓int **pp = &p，即pp是指向整型指针的指针

二级指针：指向指针的指针

- 类型名 ** 变量名
  - ✓即指向“类型名”指针的指针



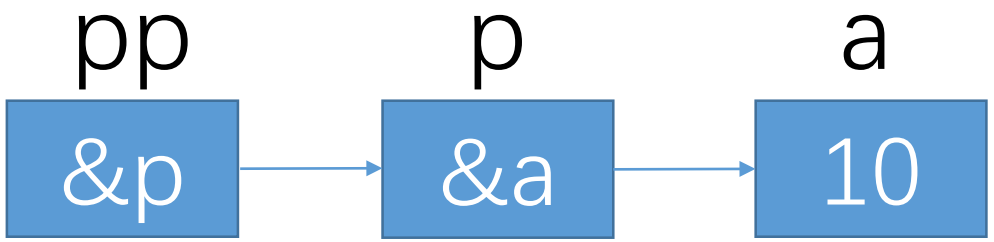
- 假设p是一个整型指针
  - ✓int a = 10; int *p = &a;
  - ✓int **pp = &p，即pp是指向整型指针的指针
  - ✓int *pp = &p; ?

会有warning!

保证同级指针相互赋值!

二级指针：指向指针的指针

- 类型名 ** 变量名
  - ✓即指向“类型名”指针的指针

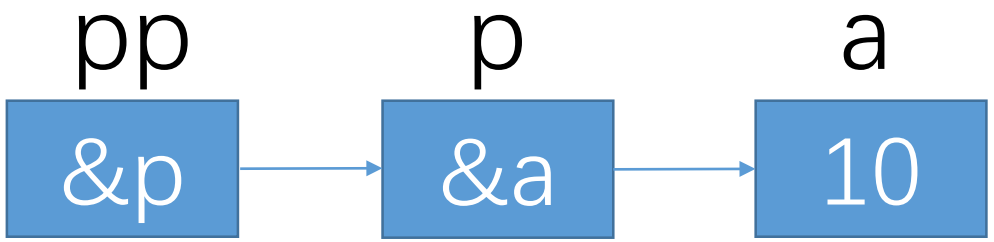


- 假设p是一个整型指针
  - ✓int a = 10; int *p = &a;
  - ✓int **pp = &p，即pp是指向整型指针的指针

- pp 等价于？
- *pp 等价于？
- **pp 等价于？

二级指针：指向指针的指针

- 类型名 ** 变量名
  - ✓即指向“类型名”指针的指针

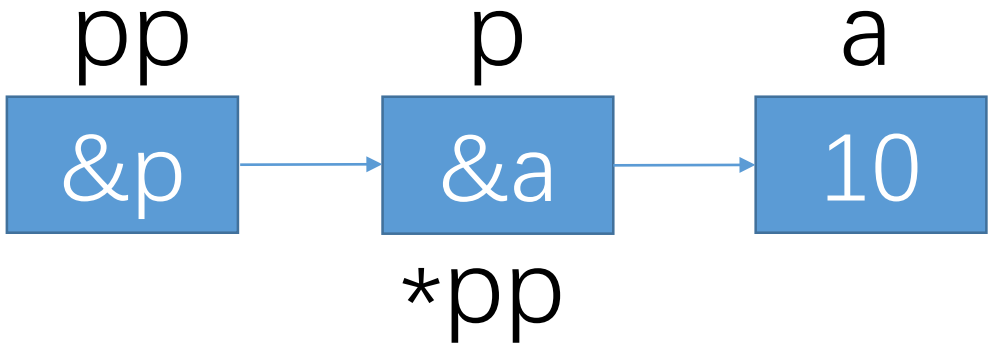


- 假设p是一个整型指针
  - ✓int a = 10; int *p = &a;
  - ✓int **pp = &p，即pp是指向整型指针的指针

- pp 等价于 &p
- *pp 等价于？
- **pp 等价于？

二级指针：指向指针的指针

- 类型名 ** 变量名
  - ✓即指向“类型名”指针的指针

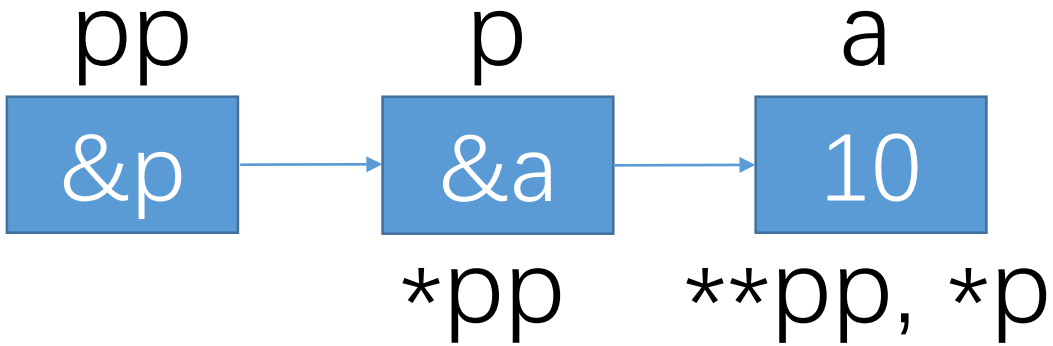


- 假设p是一个整型指针
  - ✓int a = 10; int *p = &a;
  - ✓int **pp = &p，即pp是指向整型指针的指针

- pp 等价于 &p
- *pp 等价于 p 或者 &a
- **pp 等价于？

二级指针：指向指针的指针

- 类型名 ** 变量名
  - ✓即指向“类型名”指针的指针

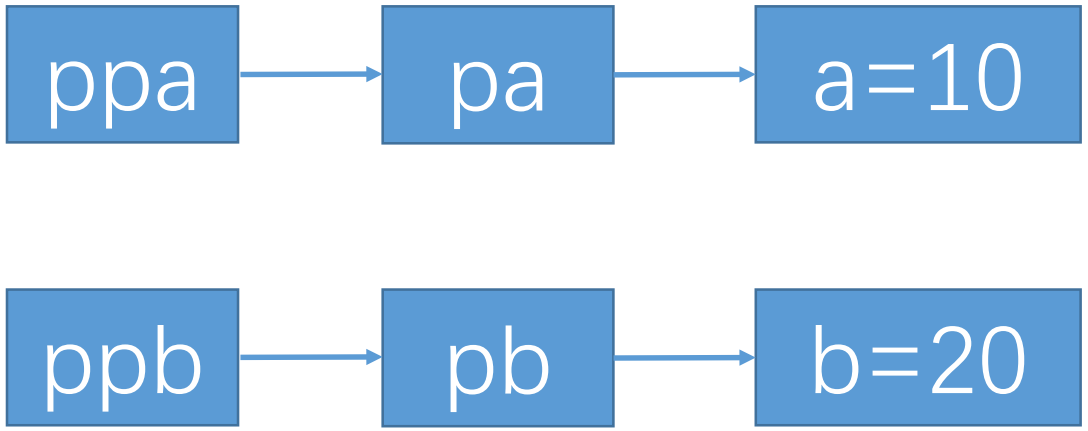


- 假设p是一个整型指针
  - ✓int a = 10; int *p = &a;
  - ✓int **pp = &p，即pp是指向整型指针的指针

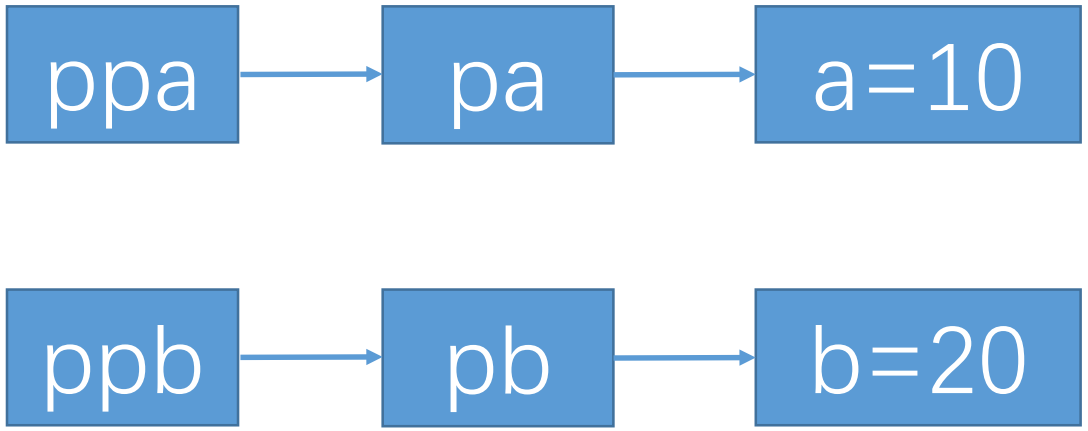
- pp 等价于 &p
- *pp 等价于 p 或者 &a
- **pp 等价于 *p 或者 a

二级指针的使用

```
int a = 10, b = 20, t;
int *pa = &a, *pb = &b, *pt;
int **ppa = &pa, **ppb = &pb, **ppt;
```



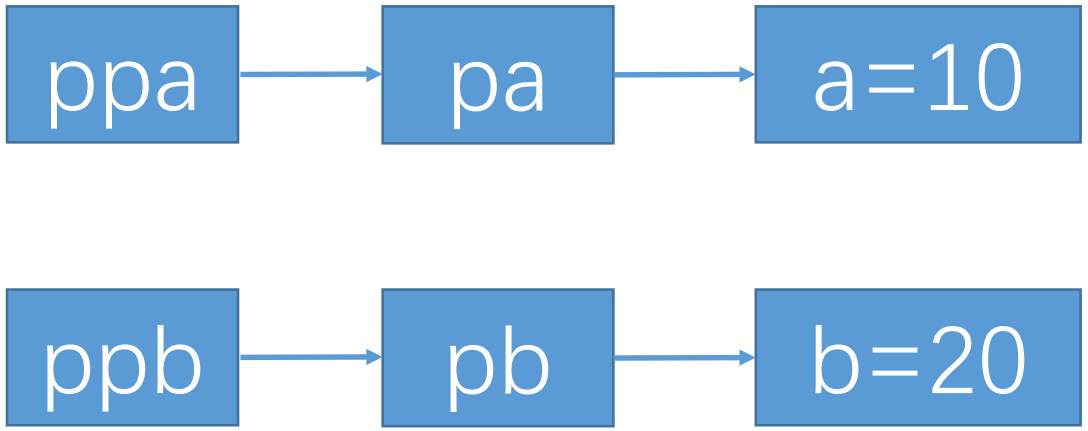
二级指针的使用




```
int a = 10, b = 20, t;
int *pa = &a, *pb = &b, *pt;
int **ppa = &pa, **ppb = &pb, **ppt;
printf("%d %d %d %d %d %d\n", a,
                              *pa,
                              **ppa,
                              b,
                              *pb,
                              **ppb);
```

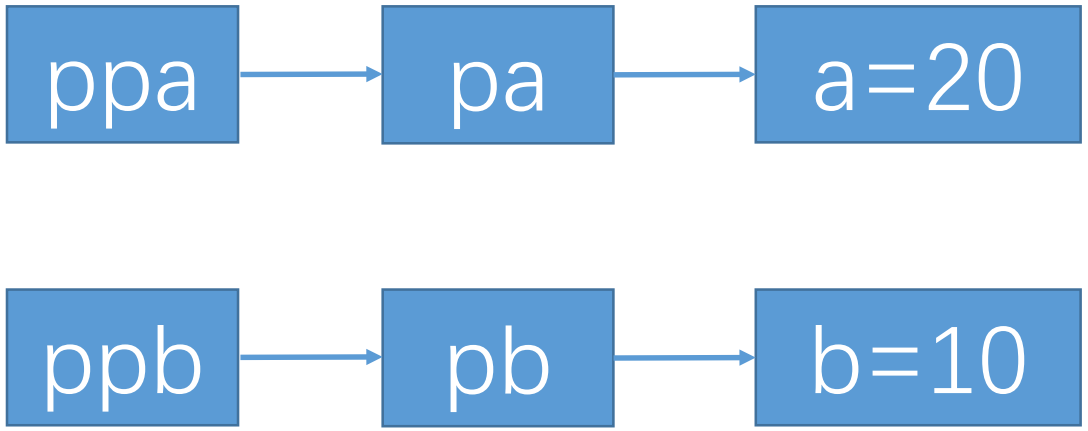
```
10 10 10 20 20 20
```

二级指针的使用




```
int a = 10, b = 20, t;
int *pa = &a, *pb = &b, *pt;
int **ppa = &pa, **ppb = &pb, **ppt;

t = b;
b = a;
a = t;
printf("%d %d %d %d %d %d\n", a,
                              *pa,
                              **ppa,
                              b,
                              *pb,
                              **ppb);
```

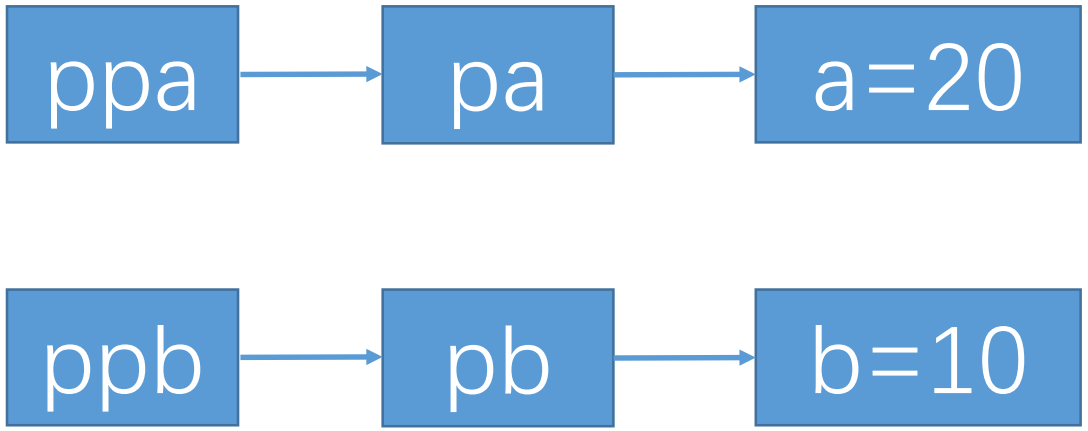
二级指针的使用




```
int a = 10, b = 20, t;
int *pa = &a, *pb = &b, *pt;
int **ppa = &pa, **ppb = &pb, **ppt;

t = b;
b = a;
a = t;
printf("%d %d %d %d %d %d\n", a,
                              *pa,
                              **ppa,
                              b,
                              *pb,
                              **ppb);
```

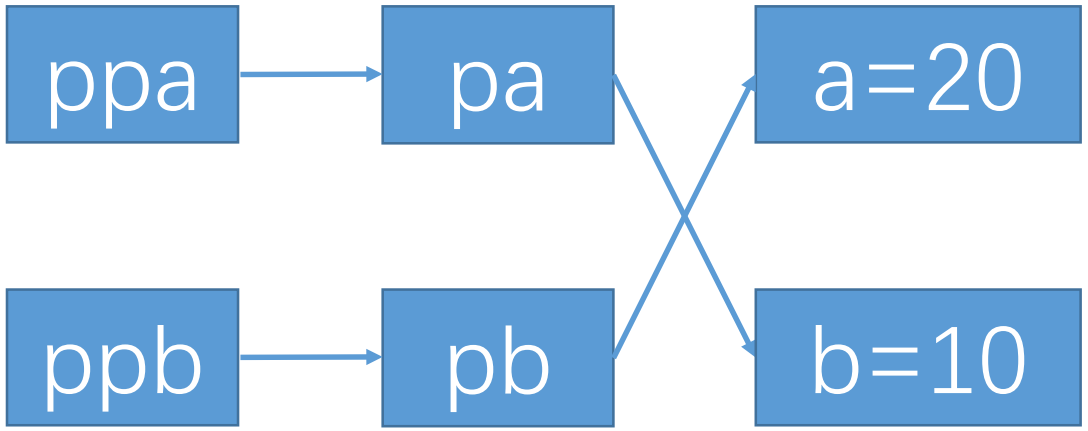
20 20 20 10 10 10

二级指针的使用




```
int a = 10, b = 20, t;
int *pa = &a, *pb = &b, *pt;
int **ppa = &pa, **ppb = &pb, **ppt;

pt = pb;
pb = pa;
pa = pt;
printf("%d %d %d %d %d %d\n", a,
                              *pa,
                              **ppa,
                              b,
                              *pb,
                              **ppb);
```

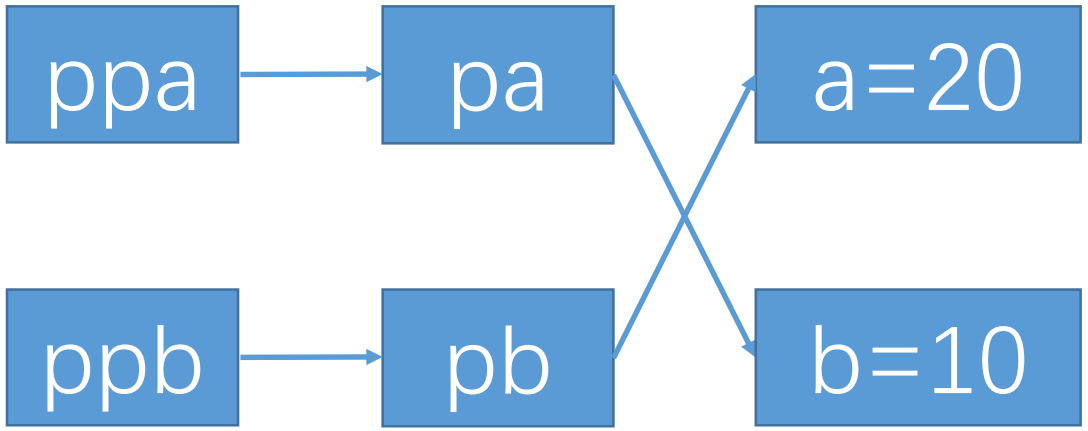
二级指针的使用




```
20 10 10 10 20 20
```

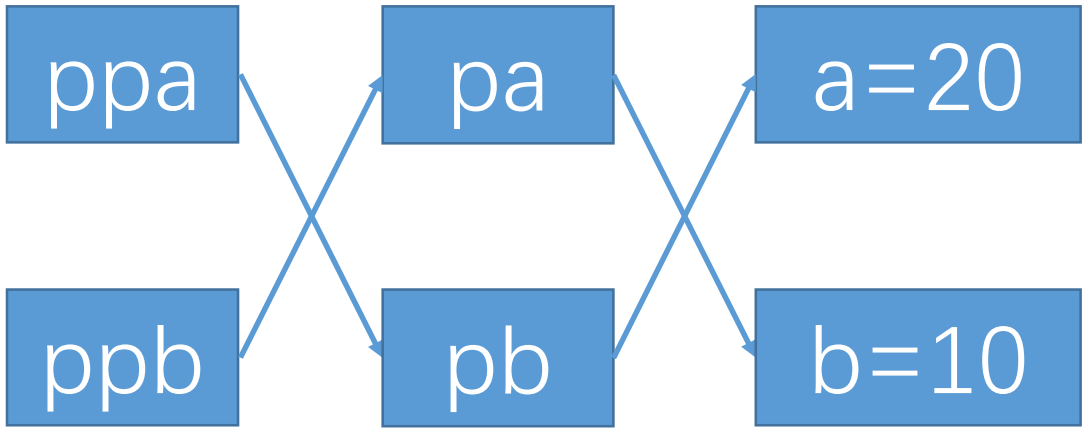
```c
int a = 10, b = 20, t;
int *pa = &a, *pb = &b, *pt;
int **ppa = &pa, **ppb = &pb, **ppt;

pt = pb;
pb = pa;
pa = pt;
printf("%d %d %d %d %d %d\n", a,
                              *pa,
                              **ppa,
                              b,
                              *pb,
                              **ppb);
```

二级指针的使用




```
int a = 10, b = 20, t;
int *pa = &a, *pb = &b, *pt;
int **ppa = &pa, **ppb = &pb, **ppt;

ppt = ppb;
ppb = ppa;
ppa = ppt;
printf("%d %d %d %d %d %d\n", a,
                              *pa,
                              **ppa,
                              b,
                              *pb,
                              **ppb);
```

二级指针的使用




```
int a = 10, b = 20, t;
int *pa = &a, *pb = &b, *pt;
int **ppa = &pa, **ppb = &pb, **ppt;

ppt = ppb;
ppb = ppa;
ppa = ppt;
printf("%d %d %d %d %d %d\n", a,
                              *pa,
                              **ppa,
                              b,
                              *pb,
                              **ppb);
```

```
20 10 20 10 20 10
```

3、二级指针与二维数组、指针数组

二级指针与二维数组

- 一维数组名是一个指针（常量）；同理，二维数组名是一个二级指针（常量）

二级指针与二维数组

- 一维数组名是一个指针（常量）；同理，二维数组名是一个二级指针（常量）

- 回想：int a[10]
  - ✓那么a是一个指针常量，其值为数组第一个元素的地址，即&a[0]

二级指针与二维数组

- 一维数组名是一个指针（常量）；同理，二维数组名是一个二级指针（常量）

- 回想：int a[10]
  - ✓那么a是一个指针常量，其值为数组第一个元素的地址，即&a[0]

- 假定有二维数组 int a[3][2]
  - ✓a是一个二级指针常量，其值为a的第一个一维数组的开始地址，即&a[0]
  - ✓a[0]是一个一级指针常量，其值为a的第一个一维数组的第一个元素的地址，即&a[0][0]
  - ✓所以，a 等价于 &a[0] 或 “&&a[0][0]”

二级指针与二维数组

- 一维数组名是一个指针（常量）；同理，二维数组名是一个二级指针（常量）

- 回想：int a[10]
  - ✓那么a是一个指针常量，其值为数组第一个元素的地址，即&a[0]

- 假定有二维数组 int a[3][2]
  - ✓a是一个二级指针常量，其
  - ✓a[0]是一个一级指针常量，
    &a[0][0]
  - ✓所以，a 等价于 &a[0] 或 “&&a[0][0]”

a和a[0], &a[0][0]的值相同，但不等价!

二级指针与二维数组

- 假定有二维数组 int a[3][2]
  - ✓a是一个二级指针常量，其值为a的第一个一维数组的开始地址，即&a[0]
  - ✓a[0]是一个一级指针常量，其值为a的第一个一维数组的第一个元素的地址，即&a[0][0]
  - ✓所以，a == &a[0] == “&&a[0][0]”

- a+1

二级指针与二维数组

- 假定有二维数组 int a[3][2]
  - ✓a是一个二级指针常量，其值为a的第一个一维数组的开始地址，即&a[0]
  - ✓a[0]是一个一级指针常量，其值为a的第一个一维数组的第一个元素的地址，即&a[0][0]
  - ✓所以，a == &a[0] == “&&a[0][0]”

- a+1是a的第二个一维数组的开始地址，即&a[1];

二级指针与二维数组

- 假定有二维数组 int a[3][2]
  - ✓a是一个二级指针常量，其值为a的第一个一维数组的开始地址，即&a[0]
  - ✓a[0]是一个一级指针常量，其值为a的第一个一维数组的第一个元素的地址，即&a[0][0]
  - ✓所以，a == &a[0] == “&&a[0][0]”

- a+1是a的第二个一维数组的开始地址，即&a[1];
  - ✓a+i ==

二级指针与二维数组

- 假定有二维数组 int a[3][2]
  - ✓a是一个二级指针常量，其值为a的第一个一维数组的开始地址，即&a[0]
  - ✓a[0]是一个一级指针常量，其值为a的第一个一维数组的第一个元素的地址，即&a[0][0]
  - ✓所以，a == &a[0] == “&&a[0][0]”

- a+1是a的第二个一维数组的开始地址，即&a[1];
  - ✓a+i == &a[i];

二级指针与二维数组

- 假定有二维数组 int a[3][2]
  - ✓a是一个二级指针常量，其值为a的第一个一维数组的开始地址，即&a[0]
  - ✓a[0]是一个一级指针常量，其值为a的第一个一维数组的第一个元素的地址，即&a[0][0]
  - ✓所以，a == &a[0] == “&&a[0][0]”

- a+1是a的第二个一维数组的开始地址，即&a[1]；
  - ✓a+i == &a[i]； *(a+i) ==

二级指针与二维数组

- 假定有二维数组 int a[3][2]
  - ✓a是一个二级指针常量，其值为a的第一个一维数组的开始地址，即&a[0]
  - ✓a[0]是一个一级指针常量，其值为a的第一个一维数组的第一个元素的地址，即&a[0][0]
  - ✓所以，a == &a[0] == “&&a[0][0]”

- a+1是a的第二个一维数组的开始地址，即&a[1];
  - ✓a+i == &a[i]; *(a+i) == a[i];

二级指针与二维数组

- 假定有二维数组 int a[3][2]
  - ✓a是一个二级指针常量，其值为a的第一个一维数组的开始地址，即&a[0]
  - ✓a[0]是一个一级指针常量，其值为a的第一个一维数组的第一个元素的地址，即&a[0][0]
  - ✓所以，a == &a[0] == “&&a[0][0]”

- a+1是a的第二个一维数组的开始地址，即&a[1];
  - ✓a+i == &a[i]; *(a+i) == a[i]; **(a+i) ==

二级指针与二维数组

- 假定有二维数组 int a[3][2]
  - ✓a是一个二级指针常量，其值为a的第一个一维数组的开始地址，即&a[0]
  - ✓a[0]是一个一级指针常量，其值为a的第一个一维数组的第一个元素的地址，即&a[0][0]
  - ✓所以，a == &a[0] == “&&a[0][0]”

- a+1是a的第二个一维数组的开始地址，即&a[1];
  - ✓a+i == &a[i]; *(a+i) == a[i]; **(a+i) == a[i][0]

二级指针与二维数组

- 假定有二维数组 int a[3][2]
  - ✓a是一个二级指针常量，其值为a的第一个一维数组的开始地址，即&a[0]
  - ✓a[0]是一个一级指针常量，其值为a的第一个一维数组的第一个元素的地址，即&a[0][0]
  - ✓所以，a == &a[0] == “&&a[0][0]”

- a+1是a的第二个一维数组的开始地址，即&a[1];
  - ✓a+i == &a[i]; *(a+i) == a[i]; **(a+i) == a[i][0]

- *(a+i)+j ==

二级指针与二维数组

- 假定有二维数组 int a[3][2]
  - ✓a是一个二级指针常量，其值为a的第一个一维数组的开始地址，即&a[0]
  - ✓a[0]是一个一级指针常量，其值为a的第一个一维数组的第一个元素的地址，即&a[0][0]
  - ✓所以，a == &a[0] == “&&a[0][0]”

- a+1是a的第二个一维数组的开始地址，即&a[1];
  - ✓a+i == &a[i]; *(a+i) == a[i]; **(a+i) == a[i][0]

- *(a+i)+j == &a[i][j];

二级指针与二维数组

- 假定有二维数组 int a[3][2]
  - ✓a是一个二级指针常量，其值为a的第一个一维数组的开始地址，即&a[0]
  - ✓a[0]是一个一级指针常量，其值为a的第一个一维数组的第一个元素的地址，即&a[0][0]
  - ✓所以，a == &a[0] == “&&a[0][0]”

- a+1是a的第二个一维数组的开始地址，即&a[1];
  - ✓a+i == &a[i]; *(a+i) == a[i]; **(a+i) == a[i][0]

- *(a+i)+j == &a[i][j]; *(*(a+i)+j) ==

二级指针与二维数组

- 假定有二维数组 int a[3][2]
  - ✓a是一个二级指针常量，其值为a的第一个一维数组的开始地址，即&a[0]
  - ✓a[0]是一个一级指针常量，其值为a的第一个一维数组的第一个元素的地址，即&a[0][0]
  - ✓所以，a == &a[0] == “&&a[0][0]”

- a+1是a的第二个一维数组的开始地址，即&a[1];
  - ✓a+i == &a[i]; *(a+i) == a[i]; **(a+i) == a[i][0]

- *(a+i)+j == &a[i][j]; *(*(a+i)+j) == a[i][j]

二级指针与二维数组

- 使用二级指针变量对二维数组a进行访问
  - ✓int *pa = *a;
    int **ppa = &pa;

```c
#include <stdio.h>

int main()
{
    int a[3][2] = {2, 3, 4, 5, 6, 7};
    int *pa = a[0];
    int **ppa = &pa;
    for(int i = 0; i < 6; i++)
    {
        printf("%d ", **ppa);
        (*ppa)++;
    }
    return 0;
}
```

二级指针与二维数组

- 使用二级指针变量对二维数组a进行访问
  - ✓int *pa = *a;
    int **ppa = &pa;

- 不能直接用int **ppa = a;
  - ✓因为a本质上只是一个地址！(&a[0][0])
  - ✓如果直接赋值，*ppa的值即为a[0][0]

```c
#include <stdio.h>

int main()
{
    int a[3][2] = {2, 3, 4, 5, 6, 7};
    int *pa = a[0];
    int **ppa = &pa;
    for(int i = 0; i < 6; i++)
    {
        printf("%d ", **ppa);
        (*ppa)++;
    }
    return 0;
}
```

二级指针与二维数组

- 使用二级指针变量对二维数组a进行访问
  - ✓int *pa = *a;
    int **ppa = &pa;
- 不能直接用int **ppa = a;
  - ✓因为a本质上只是一个地址！(&a[0][0])
  - ✓如果直接赋值，*ppa的值即为a[0][0]
- ppa++或者ppa+1没有意义
  - ✓会指向指针pa的下一个指针

```c
#include <stdio.h>

int main()
{
    int a[3][2] = {2, 3, 4, 5, 6, 7};
    int *pa = a[0];
    int **ppa = &pa;
    for(int i = 0; i < 6; i++)
    {
        printf("%d ", **ppa);
        (*ppa)++;
    }
    return 0;
}
```

二级指针与指针数组

- 指针数组名也是一个二级指针

- 假定有字符指针数组 char *str[3]
  - ✓str等价于&str[0]，str[0]是一个指针，所以str是一个二级指针

```
char *str[3] = {"one", "two", "three"};
```

二级指针与指针数组

- 指针数组名也是一个二级指针

- 假定有字符指针数组 char *str[3]
  - ✓str等价于&str[0]，str[0]是一个指针，所以str是一个二级指针

```
char *str[3] = {"one", "two", "three"};
```

- *str == str[0]　**str == ?

二级指针与指针数组

- 指针数组名也是一个二级指针

- 假定有字符指针数组 char *str[3]
  - ✓str等价于&str[0]，str[0]是一个指针，所以str是一个二级指针

```
char *str[3] = {"one", "two", "three"};
```

- *str == str[0]　**str == str[0][0]（字符'o'）

二级指针与指针数组

- 指针数组名也是一个二级指针

- 假定有字符指针数组 char *str[3]
  - ✓str等价于&str[0]，str[0]是一个指针，所以str是一个二级指针

```
char *str[3] = {"one", "two", "three"};
```

- *str == str[0]　**str == str[0][0]（字符'o'）
- *(str+i) == ?

二级指针与指针数组

- 指针数组名也是一个二级指针

- 假定有字符指针数组 char *str[3]
  - ✓str等价于&str[0]，str[0]是一个指针，所以str是一个二级指针

```
char *str[3] = {"one", "two", "three"};
```

- *str == str[0]　**str == str[0][0]（字符'o'）
- *(str+i) == str[i]

二级指针与指针数组

- 指针数组名也是一个二级指针

- 假定有字符指针数组 char *str[3]
  - ✓str等价于&str[0]，str[0]是一个指针，所以str是一个二级指针

```
char *str[3] = {"one", "two", "three"};
```

- *str == str[0]　**str == str[0][0]（字符'o'）
- *(str+i) == str[i]　*(*(str+i)+j) == ?

二级指针与指针数组

- 指针数组名也是一个二级指针

- 假定有字符指针数组 char *str[3]
  - ✓str等价于&str[0]，str[0]是一个指针，所以str是一个二级指针

```
char *str[3] = {"one", "two", "three"};
```

- *str == str[0]　**str == str[0][0]（字符'o'）
- *(str+i) == str[i]　*(*(str+i)+j) == str[i][j]

二级指针与指针数组

- 指针数组名也是一个二级指针

- 假定有字符指针数组 char *str[3]
  - ✓str等价于&str[0]，str[0]是一个指针，所以str是一个二级指针

```
char *str[3] = {"one", "two", "three"};
```

- *str == str[0]　**str == str[0][0]（字符'o'）
- *(str+i) == str[i]　*(*(str+i)+j) == str[i][j]

- 可以用二级指针变量对str进行循环访问
  - ✓char **ppstr = str;

二级指针与指针数组

- 指针数组名也是一个二级指针
- 假定有字符指针数组 char *str[3]
  - ✓str等价于&str[0]，str[0]是一个指针，所以str是一个二级指针
- *str == str[0]　**str == str[0][0]（字符
- *(str+i) == str[i]　*(*(str+i)+j) == str[i][
- 可以用二级指针变量对str进行循环访
  - ✓char **ppstr = str;

```c
#include <stdio.h>

int main()
{
    char *str[3] = {"one", "two", "three"};
    char **ppstr = str;
    for(int i = 0; i < 3; i++)
    {
        printf("%s\n", *(ppstr+i));
    }
}
```

```c
#include <stdio.h>

int main()
{
    char *str[3] = {"one", "two", "three"};
    char **ppstr = str;
    for(int i = 0; i < 3; i++)
    {
        printf("%s\n", *ppstr);
        ppstr++;
    }
}
```

二维字符数组和字符指针数组

- 字符串数组可以用二维字符数组或者字符指针数组存储
  - ✓char str[3][6] = {"one", "two", "three"};
  - ✓char *str[3] = {"one", "two", "three"};

二维字符数组和字符指针数组

- 字符串数组可以用二维字符数组或者字符指针数组存储
  - ✓char str[3][6] = {"one", "two", "three"};
  - ✓char *str[3] = {"one", "two", "three"};

- 使用二维数组更加直观，但会造成内存浪费

二维字符数组和字符指针数组

- 字符串数组可以用二维字符数组或者字符指针数组存储
  - ✓char str[3][6] = {"one", "two", "three"};
  - ✓char *str[3] = {"one", "two", "three"};

- 使用二维数组更加直观，但会造成内存浪费

- 字符指针数组结合动态内存分配进行初始化
  - ✓str[0] = (char *) malloc (sizeof (char) * (str_length+1))

二维字符数组和字符指针数组

- 字符串数组可以用二维字符数组或者字符指针数组存储
  - ✓char str[3][6] = {"one", "two", "three"};
  - ✓char *str[3] = {"one", "two", "three"};

- 使用二维数组更加直观，但会造成内存浪费

- 字符指针数组结合动态内存分配进行初始化
  - ✓str[0] = (char *) malloc (sizeof (char) * (str_length+1))

必须给字符指针分配指向的内存地址才可以赋值！！！

4、命令行参数

命令行参数

- 让可执行程序（如.exe文件）接收参数，然后执行

命令行参数

- 让可执行程序（如.exe文件）接收参数，然后执行

- 格式：可执行程序　参数1　参数2　参数3　。。。
  - ✓例如Linux命令：cd　/home/xuesong
  - ✓cd就是可执行程序，/home/xuesong就是参数

命令行参数

- 让可执行程序（如.exe文件）接收参数，然后执行

- 格式：可执行程序　参数1　参数2　参数3　。。。
  - ✓例如Linux命令：cd　/home/xuesong
  - ✓cd就是可执行程序，/home/xuesong就是参数

- 实现方式：int main(int argc, char *argv[])
  - ✓argc: 包括可执行程序在内的参数个数
  - ✓argv: 实际存放参数的字符串数组（字符指针数组）

运行时输入 vs 命令行参数输入

```c
int main()
{
    int a[5];
    for (int i = 0; i < 5; i++)
        scanf("%d", &a[i]);

    for (int i = 0; i < 5; i++)
        printf("%d ", a[i]);
    printf("\n");

    return 0;
}
```

```c
int main(int argc, char *argv[])
{
    //int a[5];
    //for (int i = 0; i < 5; i++)
    //    scanf("%d", &a[i]);

    for (int i = 1; i < argc; i++)
        printf("%s ", argv[i]);
    printf("\n");

    return 0;
}
```

运行时输入 vs 命令行参数输入

主函数的参数

argv[0]是程序，运行参数从argv[1]开始

argv[i]是字符指针（指向字符串）！

```
int main(int argc, char *argv[])
{
    //int a[5];
    //for (int i = 0; i < 5; i++)
    //    scanf("%d", &a[i]);

    for (int i = 1; i < argc; i++)
        printf("%s ", argv[i]);
    printf("\n");

    return 0;
}
```

行参数输入

```c
int main()
{
    int a[5];
    for (int i = 0; i < 5; i++)
        scanf("%d", &a[i]);

    for (int i = 0; i < 5; i++)
        printf("%d ", a[i]);
    printf("\n");

    return 0;
}
```

```
gcc print.c -o print
./print
```

```c
int main()
{
    int a[5];
    for (int i = 0; i < 5; i++)
        scanf("%d", &a[i]);

    for (int i = 0; i < 5; i++)
        printf("%d ", a[i]);
    printf("\n");

    return 0;
}
```

```c
int main(int argc, char *argv[])
{
    //int a[5];
    //for (int i = 0; i < 5; i++)
    //    scanf("%d", &a[i]);

    for (int i = 1; i < argc; i++)
        printf("%s ", argv[i]);
    printf("\n");

    return 0;
}
```

```
gcc print.c -o print
./print
```

```
gcc print.c -o print
./print 1 2 3 4 5
```

```c
int main()
{
    int a[5];
    for (int i = 0; i < 5; i++)
        scanf("%d", &a[i]);

    for (int i = 0; i < 5; i++)
        printf("%d ", a[i]);
    printf("\n");

    return 0;
}
```

```
gcc print.c -o print
./print
```

```c
int main(int argc, char *argv[])
{
    for (int i = 1; i < argc; i++)
        printf("%s ", argv[i]);
    printf("\n");

    return 0;
}
```

编译这一步可以由Code Runner代劳

```
gcc print.c -o print
./print 1 2 3 4 5
```

将字符串转换为整型

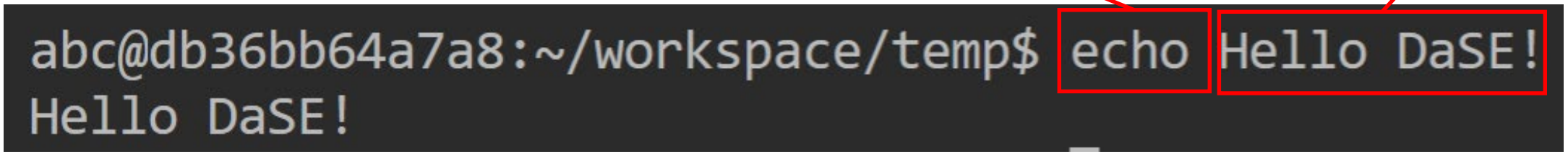
scanf("%d", &a);

sscanf(str, "%d", &a);

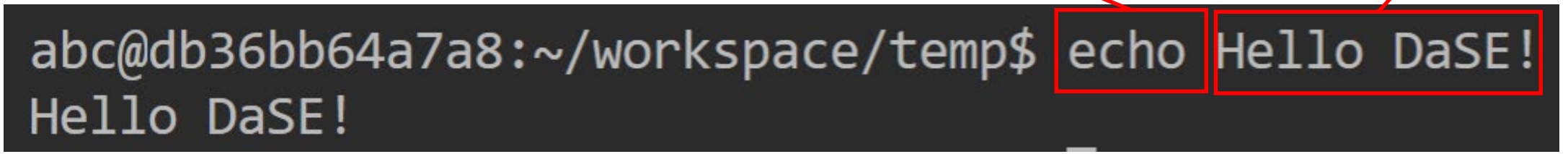
str是一个字符串
(相当于从字符串读入！)

```c
int main(int argc, char *argv[])
{
    //int a[5];
    //for (int i = 0; i < 5; i++)
    //    scanf("%d", &a[i]);

    for (int i = 1; i < argc; i++)
    {
        int a;
        sscanf(argv[i], "%d", &a);
        printf("%d ", a);
    }
    printf("\n");

    return 0;
}
```

实现Linux命令echo



实现Linux命令echo




```c
int main(int argc, char *argv[])
{
    int i = 1;
    while(argv[i] != NULL)
    {
        printf("%s ", argv[i]);
        i++;
    }
    printf("\n");

    return 0;
}
```

实现Linux命令echo

可执行程序

执行参数

```
abc@db36bb64a7a8:~/workspace/temp$ echo Hello DaSE!
Hello DaSE!
```

```c
int main(int argc, char *argv[])
{
    int i = 1;
    while(argv[i] != NULL)
    {
        printf("%s ", argv[i]);
    }
}
```

```
abc@db36bb64a7a8:~/workspace/temp$ gcc -o echo2 test.c
abc@db36bb64a7a8:~/workspace/temp$ ./echo2 Hello DaSE!
Hello DaSE!
```

5、指针返回值

指针作为函数返回值

- 函数的返回值类型可以是指针，如 int *func(int a, int b)

指针作为函数返回值

- 函数的返回值类型可以是指针，如 int *func(int a, int b)
- 使得函数可以返回字符串、数组、结构等构造数据类型!

```c
char *subString(char *str, char ch)
{
    while(*str != '0')
    {
        if(*str == ch)
            return str;
        str++;
    }
    return NULL;
}

int main()
{
    char *str = "Hello DaSE!";
    char ch = 'l';
    printf("%s\n", subString(str, ch));
    return 0;
}
```

```c
int *selectionSort(int a[], int n)
{
    for(int i = 0; i < n-1; i++)
    {
        int min = i;
        for(int j = i+1; j < n; j++)
        {
            if(a[j] < a[min])
                min = j;
        }
        int tmp = a[i];
        a[i] = a[min];
        a[min] = tmp;
    }
    return a;
}
```

```c
typedef struct student
{
    int no;
    char name[20];
}student;

student *newStudent(int sno, char sname[])
{
    student *stu = (student *)malloc(sizeof(student));
    stu->no = sno;
    strcpy(stu->name, sname);
    return stu;
}

int main()
{
    student *stu = newStudent(20220001, "Xiao Ming");
    printf("%d %s\n", stu->no, stu->name);
    return 0;
}
```

6、函数指针

函数指针

- C语言的函数名代表函数的入口地址，所以指针变量可以指向函数（入口地址），即形成所谓的“函数指针”
  - ✓类型名 (*指针变量名) （函数参数类型列表）
  - ✓int (* addptr) (int , int);

函数指针

- C语言的函数名代表函数的入口地址，所以指针变量可以指向函数（入口地址），即形成所谓的“函数指针”
  - ✓类型名 (*指针变量名) （函数参数类型列表）
  - ✓int (* addptr) (int , int);

- 函数指针赋值
  - ✓addptr = add; //add为已定义的、返回值为int、有两个int形参的函数

函数指针

```
3 int add(int a, int b)
4 {
5     return a + b;
6 }
```

- C语言的函数名代表函数的入口地址
  数（入口地址），即形成所谓的“函
  - ✓类型名 (*指针变量名) （函数参数类型
  - ✓int (* addptr) (int , int);

- 函数指针赋值
  - ✓addptr = add; //add为已定义的、返回值为int、有两个int形参的函数

函数指针

- C语言的函数名代表函数的入口地址
  数（入口地址），即形成所谓的“函
  - ✓类型名 (*指针变量名) （函数参数类型
  - ✓int (* addptr) (int , int);

```
 3  int add(int a, int b)
 4  {
 5      return a + b;
 6  }
 7
 8  int main()
 9  {
10      int (*addr)(int, int);
11      addr = add;
```

- 函数指针赋值
  - ✓addptr = add; //add为已定义的、返回值为int、有两个int形参的函数

函数指针

- C语言的函数名代表函数的入口地址
  - ✓类型名 (*指针变量名) （函数参数类型
  - ✓int (* addptr) (int , int);

```
 3  int add(int a, int b)
 4  {
 5      return a + b;
 6  }
 7
 8  int main()
 9  {
10      int (*addr)(int, int);
11      addr = add;
12      printf("%d\n", (*addr)(3, 5));
13      return 0;
14  }
```

- 函数指针赋值
  - ✓addptr = add; //add为已定义的、返回值为int、有两个int形参的函数

- 通过函数指针可以调用函数
  - ✓(*addptr) (3, 5) 等价于 add(3, 5)

函数指针作为函数参数

- C语言函数可以作为其他函数的参数，所以函数指针也可以作为函数参数

```c
#include <stdio.h>

int print(int n)
{
    printf("%d", n);
}

void visit(int n, int f(int))
{
    for(int i = 0; i < n; i++)
        f(i);
}

int main()
{
    visit(10, print);
    return 0;
}
```

```c
#include <stdio.h>

int print(int n)
{
    printf("%d", n);
}

void visit(int n, int f(int))
{
    for(int i = 0; i < n; i++)
        f(i);
}

int main()
{
    int (*func)(int);
    func = print;
    visit(10, (*func));
    return 0;
}
```

```c
int print(int n)
{
    printf("%d", n);
}

void visit(int n, int (*f)(int))
{
    for(int i = 0; i < n; i++)
        (*f)(i);
}

int main()
{
    int (*func)(int);
    func = print;
    visit(10, func);
    return 0;
}
```

小节

- 指针数组、二级指针和二维数组（重要，熟练掌握）
- 指针作为函数返回值（重要，掌握正确的使用方式）
- 函数指针（了解基本用法）
- 命令行参数（程序运行参数，非常有用！）

Next：文件