指针：存放地址的变量。

数组名是指针常量：可以改变指针指向的地址存放的值，但不能修改该指针。

例：程序func\_array\_pointer2

int i;

int j;

int a[5];

int\* p = a;

for (i = 0; i != 5; ++i)

{

a[i] = i + 1;

}

for (j = 0; j != 5; ++j)

{

// 三种方式访问数组得到的结果完全相同

printf("%d\n", a[j]);

printf("%d\n", \*(p + j));

printf("%d\n", \*(a + j));

}

\*a = 10;

\*(a + 1) = 11;

for (j = 0; j != 5; ++j)

{

printf("%d\n", a[j]);

printf("%d\n", \*(p + j));

printf("%d\n", \*(a + j));

}

// a++; // error，数组名是指针常量

二维数组与指针

例：程序func\_array\_pointer3

int i;

int j;

int a[3][4];

for (i = 0; i != 3; ++i)

{

for (j = 0; j != 4; ++j)

{

a[i][j] = i + j;

}

}

printf("%x\n", a); // 输出地址为：0x18fc40（系统随机分配）

printf("%x\n", a[0]); // 输出地址为：0x18fc40

printf("%d\n", a[0][0]);

a与a[0]指向的地址相同，a指向第0行首地址，a[0]指向第0行第0列元素地址。

在C语言中，形参为数组名或指针是相同的。

例：程序func\_array\_pointer3

void func(int a[10]);

int main(int argc, char\* argv[])

{

int i;

int b[5];

for (i = 0; i != 5; ++i)

{

b[5] = i;

}

func(b);

return 0;

}

void func(int a[10])

{

printf("%d\n", a[7]);

}

上述程序在编译过程中没有问题，因为：在C/C++中，将形参数组名作为指针变量来处理，它只会检查实参是不是指针、指针的类型和数组元素的类型是否匹配，而不会检查数组的长度。但在运行过程中，会报出内存错误。

指向数组的指针：

指针本身可以指向一个数组，而指向数组的指针则保存的是一个数组的地址。

int (\*p)[4]; // 指向数组的指针，p指向一个包含4个整型数据的一维数组

int \*p[4]; // 指针数组，每个数组元素都是一个指针变量

例：程序func\_array\_pointer4

#include <stdio.h>

void average(float\* p, int n);

void search(float (\*p)[4], int n); // 指向数组的指针

void search1(float\*\* p, int n);

int main(int argc, char\* argv[])

{

int i;

float\* p[3]; // 指针数组

float score[3][4] = {{65, 67, 70, 60}, {80, 87, 90, 81}, {90, 99, 100, 98}};

for (i = 0; i != 3; ++i)

{

p[i] = score[i];

}

average(\*score, 12);

search(score, 2);

// search1(score, 2); // 不能这样赋值，在C语言中编译不会出问题，但无法运行，不能将一个二维数组赋给一个二重指针，但可以将指针数组赋给二重指针,原因是编译器无法判断二维数组是几行几列。

search1(p, 2);

return 0;

}

void average(float\* p, int n)

{

float\* p\_end;

float sum = 0;

float aver;

p\_end = p + n - 1;

for (; p <= p\_end; ++p)

{

sum = sum + (\*p);

}

aver = sum / n;

printf("average = %5.2f\n", aver);

}

void search(float (\*p)[4], int n)

{

int i;

for (i = 0; i < 4; ++i)

{

printf("%5.2f ", \*(\*(p + n) + i));

}

printf("\n");

}

void search1(float\*\* p, int n)

{

int i;

for (i = 0; i < 4; ++i)

{

printf("%5.2f ", \*(\*(p + n) + i));

}

printf("\n");

}

输出的结果：search与search1的结果完全相同

二重指针

// 二重指针

// 二重指针

例：程序func\_array\_pointer5

#include <stdio.h>

void func(float\*\* p);

void func1(float\*\* p);

void func2(float\*\* p);

int main(int argc, char\* argv[])

{

int i;

int j;

float\* p[3];

float score[3][4] = {{65, 67, 70, 60}, {80, 87, 90, 81}, {90, 99, 100, 98}};

for (i = 0; i != 3; ++i)

{

p[i] = score[i];

}

func(p);

// func1(p);

func2(p);

printf("%4.2f\n", score[0][0]);

printf("\n");

printf("%4.2f\n", p[0][0]);

return 0;

}

void func(float\*\* p)

{

\*\*p = 102;

}

void func1(float\*\* p)

{

\*p = NULL;

}

void func2(float\*\* p)

{

p = NULL;

}

输出的结果为：

102.00

102.00

说明，在func函数中，修改二重指针的指向的对象的值影响了main函数中实参的值，并且在func2中直接修改二重指针，没有任何影响。这和在数组形参直接定义成一个int型变量，而后在函数中修改这个变量而不会影响实参的值的道理完全相同。

如果调用func1，使\*p = NULL，则会有影响：

// 二重指针

#include <stdio.h>

void func(float\*\* p);

void func1(float\*\* p);

void func2(float\*\* p);

int main(int argc, char\* argv[])

{

int i;

int j;

float\* p[3];

float score[3][4] = {{65, 67, 70, 60}, {80, 87, 90, 81}, {90, 99, 100, 98}};

for (i = 0; i != 3; ++i)

{

p[i] = score[i];

}

func(p);

func1(p);

// func2(p);

printf("%4.2f\n", score[0][0]);

printf("\n");

printf("%4.2f\n", p[1][0]);

printf("%4.2f\n", p[0][0]);

return 0;

}

void func(float\*\* p)

{

\*\*p = 102;

}

void func1(float\*\* p)

{

\*p = NULL;

}

void func2(float\*\* p)

{

p = NULL;

}

输出结果：

102.00

80.00

然后程序崩溃，原因是：在func1函数中，执行了\*p = NULL操作，导致main函数的实参p[0]指向了NULL地址，所以无法输出p[0][0]。