

宝贵建议请发送至：wangzhenyang@itcast.cn



黑马程序员

itheima.com

-编程，始于黑马

Android 课程同步笔记

Beta 0.01 版

By 阳哥

Android-JNI-01C 语言入门

1. JNI 简介 (★)

1.1 什么是 JNI

Java Native Interface(JNI), 它允许 Java 代码和其他语言写的代码进行交互。JNI 一开始是为了本地已编译语言, 尤其是 C 和 C++而设计的, 但是它并不妨碍你使用其他语言, 只要调用约定受支持就可以了。

1.12 为什么用 JNI

- ◆ 1 JNI 扩展了 Java 虚拟机的能力, 因为 Java 不能直接和硬件交互, 不能开发驱动
- ◆ 2 Java 代码效率一般要低于 C 代码, 而 Native code 效率高, 因此在数学运算, 实时渲染的游戏上以及音视频处理上都需要用 Java 调用 C 语言
- ◆ 3 复用 C/C++ 代码, C 语言经过几十年的发展, 已经形成了强大的类库(比如文件压缩, 人脸识别 opencv, 7zip, ffmpeg 等), 这些类库我们没必要用 java 语言重新实现一遍, 通过 JNI 直接调用这些类库即可
- ◆ 4 特殊的业务场景, 比如电视、车载系统、微波炉等跟硬件直接相关的开发

2. C 语言入门 (★★★)

2.1 C 语言开发工具

C 语言的开发工具比较多, 最常用是微软的 Visual Studio 系列。我们教学用的是一款轻量级开发工具 [Dev-Cpp.exe](#), 其 gcc 编译器是 C99 标准。该软件的安装比较简单, 直接下一步, 下一步即可。

安装好的图标



如图, 双击打开该软件, 然后创建一个新源文件(默认是 CPP 文件, 在保存的时候

文件名称改为 hello.c)。

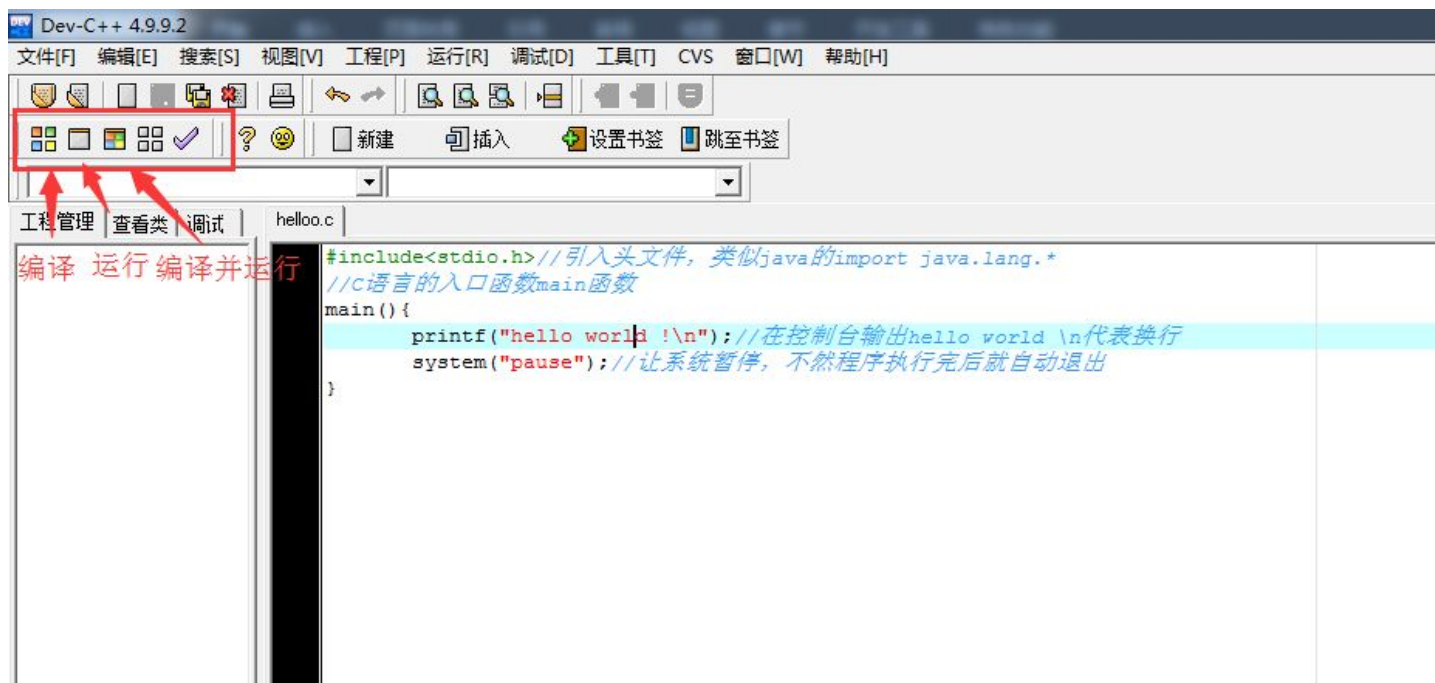
1

编写 hello world 程序

```
#include<stdio.h>//引入头文件，类似java的import java.lang.*
//C语言的入口函数main函数
main(){
    printf("hello world !\n");//在控制台输出hello world \n代表换行
    system("pause");//让系统暂停，不然程序执行完后就自动退出
}
```

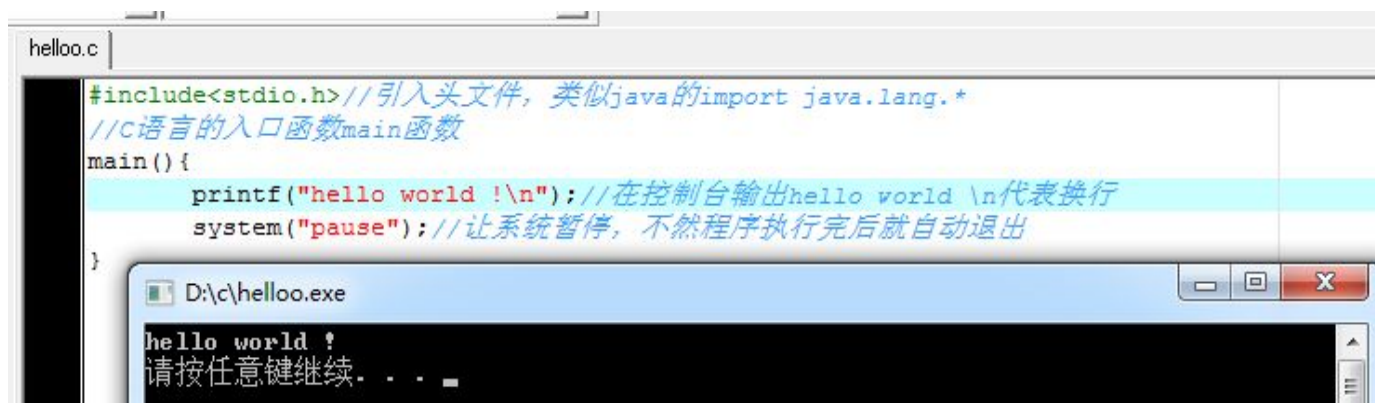
2

编译源程序，源文件要想运行必须先编译成 hello.exe 二进制文件，然后才能运行。



3

运行程序



2.2 C 语言的基本数据类型

java 语言的 8 大基本类型：

- ◆ boolean 1byte 8 位
- ◆ byte 1byte 8 位
- ◆ short 2byte 16 位
- ◆ char 2byte 16 位
- ◆ int 4byte 32 位
- ◆ float 4byte 32 位
- ◆ long 8byte 64 位
- ◆ double 8byte 64 位

C 语言的基本数据类型：

- ◆ 在 C 语言里面没有 boolean 类型，0 假 非 0 真
- ◆ 在 C 语言里面没有 byte 类型 可以用 char 表示 byte 类型。
- ◆ char 1byte 8 位 和 java 不同
- ◆ short 2byte 16 位 还可以表示 java 里面的 char
- ◆ int 4byte 32 位 和 java 一致
- ◆ float 4byte 32 位 和 java 一致
- ◆ long 4byte 32 位和 java 不同
- ◆ double 8byte 64 位和 java 一致

Tips：int、long 等整型可以用 signed 和 unsigned 关键字修饰，而 float、double 等浮点类型则不可以。

signed 有符号的，是默认的，本身不会修改类型的长度 unsigned 无符号，第一位不是符号位，所有的数都是正数。

2.2.1 案例：通过 C 代码查看 C 语言的常用数据类型长度

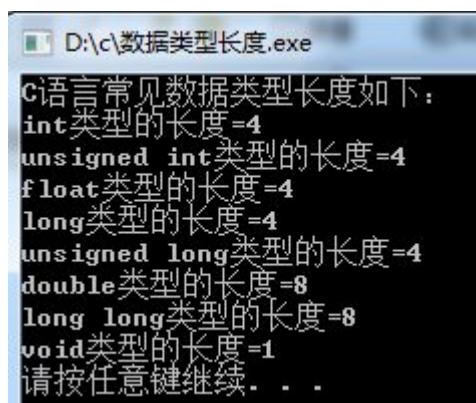
在 C 语言中查看数据类型主要靠 `sizeof(type)` 函数实现的。

```
/*
  查看 C 语言数据类型的长度
*/
main(){

    printf("C 语言常见数据类型长度如下: \n");
    printf("int 类型的长度=%d\n", sizeof(int));
    printf("unsigned int 类型的长度=%d\n", sizeof(unsigned int));
    printf("float 类型的长度=%d\n", sizeof(float));
    printf("long 类型的长度=%d\n", sizeof(long));
    printf("unsigned long 类型的长度=%d\n", sizeof(unsigned long));
    printf("double 类型的长度=%d\n", sizeof(double));
    printf("long long 类型的长度=%d\n", sizeof(long long));
    printf("void 类型的长度=%d\n", sizeof(void));

    system("pause");
}
```

执行上面代码，运行结果如下：



```
D:\c\数据类型长度.exe
C语言常见数据类型长度如下:
int类型的长度=4
unsigned int类型的长度=4
float类型的长度=4
long类型的长度=4
unsigned long类型的长度=4
double类型的长度=8
long long类型的长度=8
void类型的长度=1
请按任意键继续...
```

Tips：在上述代码中 `\n` 代表着换行，启动一个控制台程序后，必须将这个程序关闭才能再次运行另外一个控制台程序。

`long long` 类型是长长整型，64 位，这个类型一般很少用。

2.3 C 语言的输出/输入

2.3.1 C 语言的输出

C 语言格式化输出都需要用到占位符，常用的占位符如下表格所示。

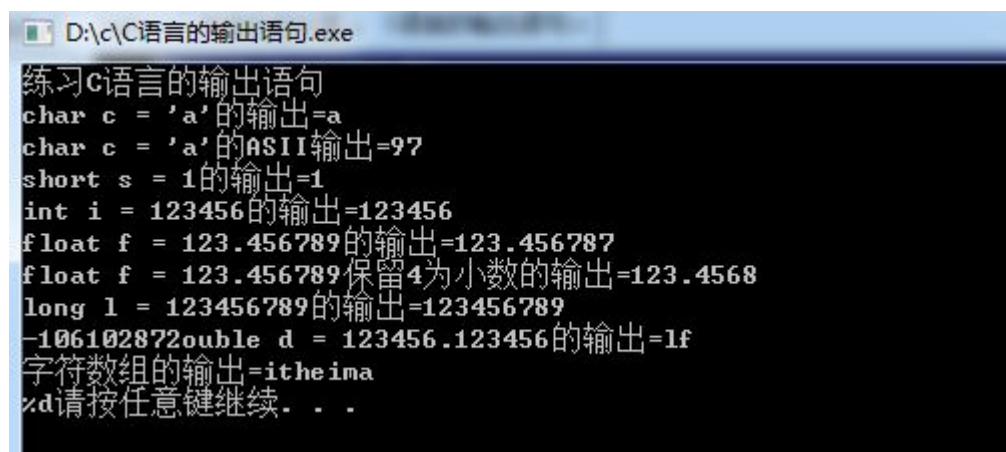
- ◆ %d - int 整数
- ◆ %ld - long int 长正数
- ◆ %c - char 表示字符
- ◆ char 类型如果以%d 输出会打印当前字符的 ASII 值
- ◆ %f - float 浮点类型
- ◆ %u - 无符号数
- ◆ %hd - 短整型 short
- ◆ %lf - double 双精度浮点类型
- ◆ %x - 十六进制输出 int 或者 long int 或者 short int
- ◆ %o - 八进制输出(是字母 o 而不是数字 0)
- ◆ %s - 字符串

2.3.2 案例-练习 C 语言的常用输出语句

```
#include<stdio.h>
main(){
    printf("练习 C 语言的输出语句\n");
    char c = 'a';
    short s = 1;
    int i = 123456;
    float f = 123.456789;
    long l = 123456789;
    double d = 123456.123456;
    char arr[] = {'i','t','h','e','i','m','a'};
    printf("char c = 'a' 的输出=%c\n",c);
    printf("char c = 'a' 的 ASII 输出=%d\n",c);
}
```

```
printf("short s = 1 的输出=%hd\n",s);
printf("int i = 123456 的输出=%d\n",i);
printf("float f = 123.456789 的输出=%f\n",f);
printf("float f = 123.456789 保留 4 为小数的输出=%.4f\n",f);
printf("long l = 123456789 的输出=%ld\n",l);
printf("%double d = 123456.123456 的输出=lf\n",d);
printf("字符数组的输出=%s\n",arr);
//输出占位符本身
printf("%%d");
system("pause");
}
```

运行结果如下：



Tips：如果在输出的语句中包含中占位符（或者说是特殊字符），那么需要在占位符本身前面再加个%。比如

`printf("%%d");`输出的结果就是%`d`。

2.3.3 C 语言的输入

C 语言的键盘输入主要是通过 `scanf("%c" ,&c)`函数实现的。

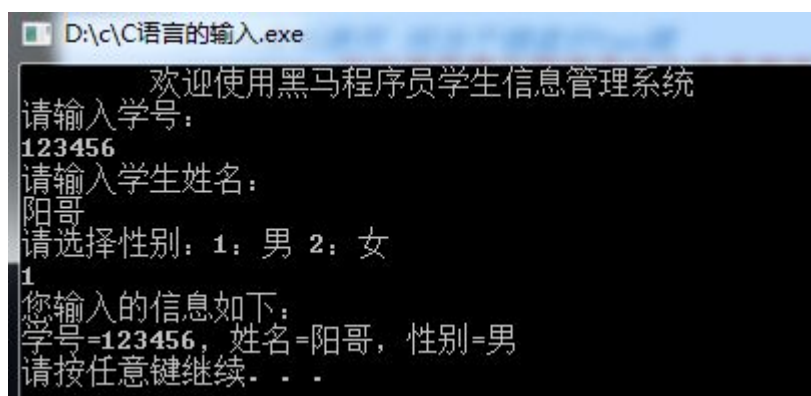
需求：从键盘中分别输入整数，字符串，模拟用户信息的录入并打印在控制台。

```
#include<stdio.h>
main(){
    // \t 是制表符 相当于键盘的 Tab 键
    printf("\t 欢迎使用黑马程序员学生信息管理系统\n");
    printf("请输入学号: \n");
    int num;//学生学号
```



```
scanf("%d",&num);
printf("请输入学生姓名: \n");
char name[]={};
scanf("%s",name);
printf("请选择性别: 1: 男 2: 女\n");
int sex=1;//性别默认 1
scanf("%d",&sex);
printf("您输入的信息如下: \n");
char* c_sex = sex==2?"女":"男";
printf("学号=%d, 姓名=%s, 性别=%s\n",num,name,c_sex);
system("pause");
}
```

程序运行结果如下：



Tips : scanf("%d",&num);中%d 是要输入的数据类型，&num 是取变量 num 的地址，其实输入的原理就是将 num 在内存的地址处存放输入的数据，这样 num 变量的值就有了。在 C 语言中没有 String 类型，因此只能用 char* 指针代替。

2.4 C 语言的指针

指针是 C 语言的难点，也是精华所在！

指针是一个特殊的变量，它里面存储的数值被解释成为内存里的一个地址。

要搞清一个指针需要搞清指针的四方面的内容：指针的类型、指针所指向的类型、指针的值或者叫指针所指向的内存区、指针本身所占据的内存区。

◆ 指针的类型 :把指针声明语句里的指针名字去掉 ,剩下的部分就是这个指针的类型 ,比如 int* p;语句中 int* 就

是指针的类型。

◆ 指针所指向的类型：把指针声明语句中的指针名字和名字左边的指针声明符*去掉，剩下的就是指针所指向的类型（在指针的算术运算中，指针所指向的类型有很大的作用）比如 `int* p;` 语句中，`int` 就是指针指向的类型。

◆ 指针所指向的内存区：从指针的值所代表的那个内存地址开始，长度为 `sizeof(指针所指向的类型)` 的一片内存区。（一个指针指向了某块内存区域，就相当于说该指针的值是这块内存区域的首地址）

◆ 指针本身所占据的内存区：用函数 `sizeof(指针的类型)` 可以测出指针本身所占据的内存区（在 32 位平台里，指针本身占据了 4 个字节的长度）

2.4.1 指针用法入门

```
#include<stdio.h>
main(){
    int i =1234;
    int* p =&i;
    //通过指针输出变量 i 的值
    printf("%d\n",*p);
    //通过指针输出字符串
    char* str="哈哈，我是字符串";
    printf("%s\n",str);
    system("pause");
}
```

2.4.2 认识多种指针

◆ `int *p;`（普通指针） //首先从 `p` 处开始,先与*结合,所以说明 `p` 是一个指针,然后再与 `int` 结合,说明指针所指向的内容的类型为 `int` 型。所以 `p` 是一个返回整型数据的指针。

◆ `int p[3];`（数组不是指针） //首先从 `P` 处开始,先与[]结合,说明 `p` 是一个数组,然后与 `int` 结合,说明数组里的元素是整型的,所以 `p` 是一个由整型数据组成的数组。

◆ `int *p[3];`（多个指针组成的数组） //首先从 `P` 处开始,先与[]结合,因为其优先级比*高,所以 `P` 是一个数组,然后再

与*结合,说明数组里的元素是指针类型,然后再与 int 结合,说明指针所指向的内容的类型是整型的,所以是一个由返回整型数据的指针所组成的数组

◆ `int (*p)[3];` (指向数组的指针) //首先从 p 处开始,先与*结合,说明 p 是一个指针然后再与[]结合(与"()")这步可以忽略,只是为了改变优先级),说明指针所指向的内容是一个数组,然后再与 int 结合,说明数组里的元素是整型的。所以 p 是一个指向由整型数据组成的数组的指针

◆ `int **p;` (二级指针,指向指针的指针) //首先从 p 开始,先与*结合,说明 p 是一个指针,然后再与*结合,说明指针所指向的元素是指针,然后再与 int 结合,说明该指针所指向的元素是整型数据。所以 p 是一个返回指向整型数据的指针的指针。

◆ `int p(int);` (返回值为 int 的函数,不是指针) //从 p 处起,先与()结合,说明 p 是一个函数,然后进入()里分析,说明该函数有一个整型变量的参数然后再与外面的 int 结合,说明函数的返回值是一个整型数据。所以 p 是一个有整型参数且返回类型为整型的函数

◆ `int (*p)(int);` (指向函数的指针) //从 p 处开始,先与指针结合,说明 p 是一个指针,然后与()结合,说明指针指向的是一个函数,然后再与()里的 int 结合,说明函数有一个 int 型的参数,再与最外层的 int 结合,说明函数的返回类型是整型,所以 p 是一个指向有一个整型参数且返回类型为整型的函数的指针

◆ `int *(*p(int))[3];` (大脑分析不出来了,有点儿变态了,看说明吧) //从 p 开始,先与()结合,说明 p 是一个函数,然后进入()里面,与 int 结合,说明函数有一个整型变量参数,然后再与外面的*结合,说明函数返回的是一个指针,然后到最外面一层,先与[]结合,说明返回的指针指向的是一个数组,然后再与*结合,说明数组里的元素是指针,然后再与 int 结合,说明指针指向的内容是整型数据。所以 p 是一个参数为一个整数且返回一个指向由整型指针变量组成的数组的指针变量的函数

2.4.3 外挂的原理 (拓展趣味知识)

很多游戏的外挂,其实就是通过找到变量(比如记录游戏剩余时间的变量)的地址,然后修改该地址的值来实现

的。CheatEngine561.exe 就是一款可以查找其他应用地址，并能修改地址值的应用。

CheatEngine561.exe 软件的安装很简单，一直点击下一步即可。安装好以后打开程序主界面，如下图所示：



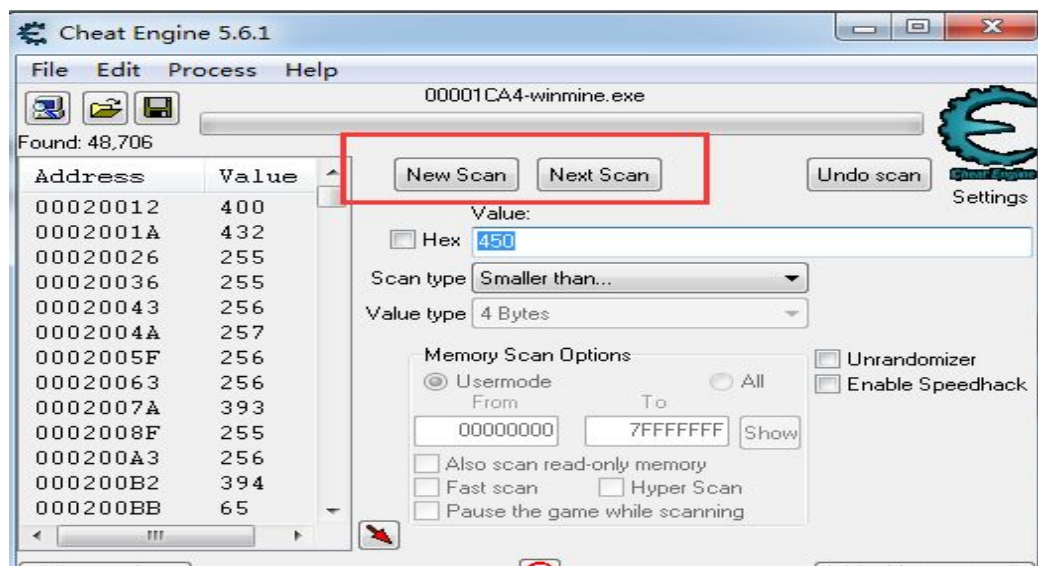
我们用一个老版本的扫雷为例，演示如何修改扫雷的时间。打开超链接从百度网盘下载：[CheatEngine561.exe](#) 和 [winmine.exe](#)

- 1 在菜单项打开 Process 选项，选择扫雷进程
- 2 在 Value 输入框中输入一个跟当前计时相近的值，如果是大于真实计时值则在 scan type 下来框中选择 Bigger than, 然后点击 First Scan 按钮。



3

扫描以后会在左侧列表框中列出所有符合条件的变量，但是太多还无法确定扫雷记录时间的变量时哪个，因此需要在 Value 框中继续反复多次输入数字，然后选择 Scan type。第二次扫描要点击 Next Scan 按钮，这样才会对左侧的值进行筛选。



4

重复第 3 步骤，指导找到扫雷记录时间的变量，双击变量值，这是该变量值就会在软件的最下侧显示



5

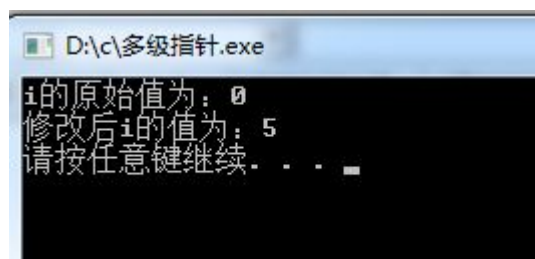
找到扫雷记录时间的地址以后，我们可以慢慢的把扫雷完成，然后修改时间为 3 秒，大功告成。

2.4.4 案例-使用多级指针

```
#include<stdio.h>
main(){
    int i=0;
    int *p = &i; //一级指针 存放变量的地址
```

```
int **q = &p; //二级指针 存放一级指针的地址
int ***r = &q; //三级指针 存放二级指针的地址
int ****s = &r; //四级指针 存放三级指针的地址
printf("i 的原始值为: %d\n", i);
****s = 5;
printf("修改后 i 的值为: %d\n", i);
system("pause");
}
```

运行结果如图：



显然我们通过四级指针成功修改了变量 i 的值。

2.4.5 指针常见的错误

◆ 野指针错误

```
main(){
    int *p; //定义了一个未指向任何地址的指针，该指针随机指向内存中的一个地址
    *p = 1; //给改地址赋值 这是非法的
    printf("%d\n", *p);
    system("pause");
}
```

使用指针一定要先赋值，再使用。

◆ 指针类型不正确异常

```
int i = 123456;
short *p = &i;
printf("%hd\n", *p);
system("pause");
```

上面代码会有警告：[Warning] initialization from incompatible pointer type 而且运行结果值为负数，因为已经

溢出了。

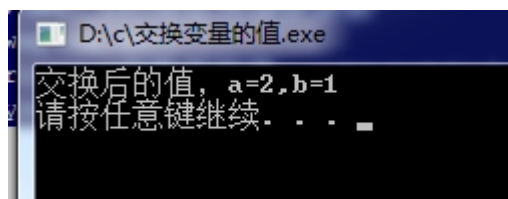
```
int i = 123456;
char *p = &i;
printf("%s\n", *p);
system("pause");
```

上面代码运行时，程序异常终止。

2.4.6 案例-通过函数交换两个变量的值

```
#include<stdio.h>
swap(int *p,int *q){
    //将 a、b 两个变量的地址的值交换
    int tmp = *p;
    *p = *q;
    *q= tmp;
}
main(){
    int a = 1;
    int b = 2;
    //将 a、b 的地址传递给函数
    swap(&a,&b);
    printf("交换后的值，a=%d,b=%d\n",a,b);
    system("pause");
}
```

运行的结果为：



显然 a、b 的值成功交换了。

2.5 C 语言的数组

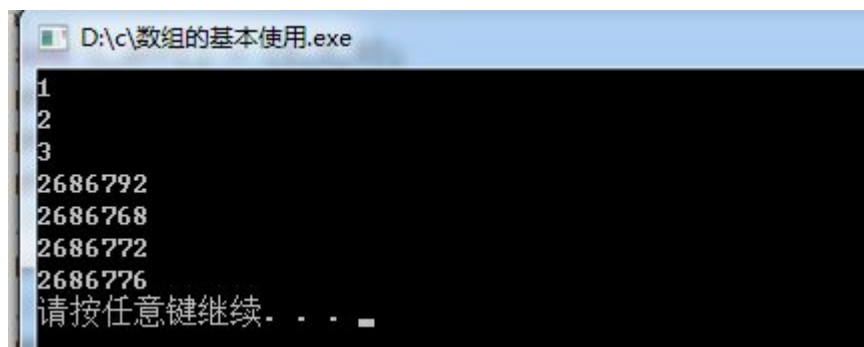
- ◆ 1 数组中的所有元素存在一块连续的内存空间中

◆ 2 数组名就是第一个元素的地址

2.5.1 案例-数组的基本使用

```
#include<stdio.h>
main(){
    int i_arr[] = {1,2,3};
    printf("%d\n",i_arr[0]);
    printf("%d\n",i_arr[1]);
    printf("%d\n",i_arr[2]);
    //C 语言不对脚标越界进行检查
    printf("%d\n",i_arr[4]);
    //打印数组的成员的地址
    printf("%d\n",&i_arr[0]);
    printf("%d\n",&i_arr[1]);
    printf("%d\n",&i_arr[2]);
    system("pause");
}
```

运行结果如下：



```
D:\c\数组的基本使用.exe
1
2
3
2686792
2686768
2686772
2686776
请按任意键继续...
```

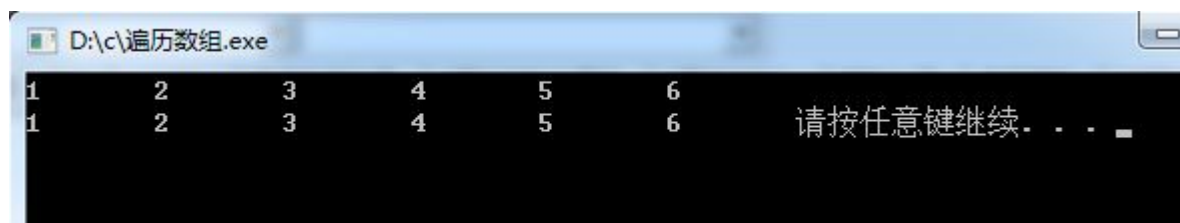
Tips：通过上面的运行结果发现：

- 1) C 语言对脚标越界不进行检查，当脚标越界时照样输出内存中的一个地址值
- 2) 数组各个成员在内存中连续的，肯定的 int 类型的元素占用了 4 个字节

2.5.2 案例-遍历数组

```
#include<stdio.h>
void printArray(int arr[],int length){
    int i=0;
    //C99 标准中 i 变量的定义不能写到 for () 中
    for(;i<length;i++){
        printf("%d\t",arr[i]);
    }
    printf("\n");
}
//因为数组的名字就是数组第一个脚标的位置，因为我们可以拿到第一个脚标
void printArray2(int *p,int length){
    int i=0;
    for(;i<length;i++){
        //每次将指针移动到下一个位置
        printf("%d\t",*(p+i));
    }
}
main(){
    int arr[] = {1,2,3,4,5,6};
    printArray(arr,6);
    printArray2(arr,6);
    system("pause");
}
```

运行结果如下：



2.6 内存结构

2.6.1 C 程序结构

一个 C 程序本质上都是由 BSS(Block Started by Symbol) 段、Data 段、Text 段三个组成的。

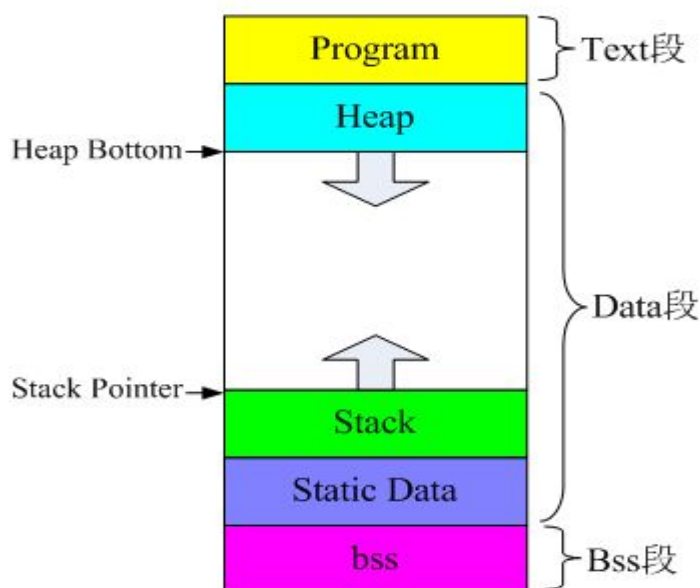
◆ BSS 段：在采用段式内存管理的架构中，BSS 段（Block Started by Symbol）通常是指用来存放程序中未初始化的全局变量的一块内存区域。BSS 是英文 Block Started by Symbol 的简称。BSS 段属于静态内存分配，即程序一开始就将其清零了。比如，在 C 语言之类的程序编译完成之后，已初始化的全局变量保存在.data 段中，未初始化的全局变量保存在.bss 段中。

◆ 数据段：在采用段式内存管理的架构中，数据段（data segment）通常是指用来存放程序中已初始化的全局变量的一块内存区域。数据段属于静态内存分配。

◆ 代码段：在采用段式内存管理的架构中，代码段（text segment）通常是指用来存放程序执行代码的一块内存区域。这部分区域的大小在程序运行前就已经确定，并且内存区域属于只读。在代码段中，也有可能包含一些只读的常数变量，例如字符串常量等。

Tips: text 和 data 段都在可执行文件中（在嵌入式系统里一般是固化在镜像文件中），由系统从可执行文件中加载；而 BSS 段不在可执行文件中，由系统初始化。

程序编译后生成的目标文件至少含有这三个段，这三个段的大致结构图如下所示：



其中 data 段包含三个部分：heap(堆)、stack(栈)和静态数据区。

◆ 堆 (heap)：堆是用于存放进程运行中被动态分配的内存段，它的大小并不固定，可动态扩张或缩减。当进程调用 malloc 等函数分配内存时，新分配的内存就被动态添加到堆上（堆被扩张）；当利用 free 等函数释放内存时，被释放的内存从堆中被剔除（堆被缩减）

◆ 栈 (stack)：栈又称堆栈，是用户存放程序临时创建的局部变量，也就是说我们函数括弧“{}”中定义的变量（但不包括 static 声明的变量，static 意味着在数据段中存放变量）。除此以外，在函数被调用时，其参数也会被压入发起调用的进程栈中，并且待到调用结束后，函数的返回值也会被存放回栈中。由于栈的先进后出特点，所以栈特别方便用来保存/恢复调用现场。从这个意义上讲，我们可以把堆栈看成一个寄存、交换临时数据的内存区。

stack 段存放函数内部的变量、参数和返回地址，其在函数被调用时自动分配，访问方式就是标准栈中的 LIFO 方式。（因为函数的局部变量存放在此，因此其访问方式应该是栈指针加偏移的方式）

2.6.2 案例-不同变量在内存中的区域

```
int a = 0; //全局初始化区
char *p1; //全局未初始化区

main()
{
    static int c =0; //全局（静态）初始化区
    int b; //栈
    char s[] = "abc"; //栈
    char *p2; //栈
    char *p3 = "123456"; //"123456\0"在常量区，p3 在栈上。
    p1 = (char *)malloc(10);
    p2 = (char *)malloc(20); //分配得来得 10 和 20 字节的区域就在堆区。
}
```

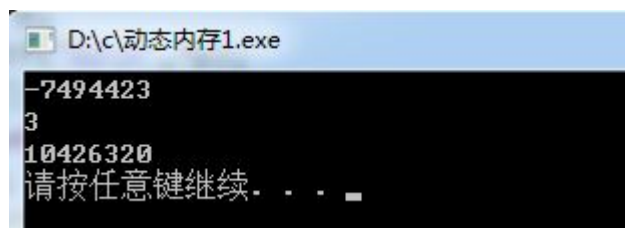
2.6.3 案例-动态内存的申请与回收

◆ 静态内存是由系统分配的,是栈内存中的连续内存空间，其运行效率非常高，且可以被系统自动回收。但是在某些情况下我们需要动态的申请一些内存空间，比如，在创建数组的时候我们不知道数组的长度是多少，那么我们就需要创建动态数组

◆ 动态内存是程序员手动申请的在堆内存中开辟的空间不一定是连续,,运行效率略慢,容易产生碎片需要手动回收

```
#include<stdio.h>
main(){
    int *p;
    printf("%d\n",*p);
    //动态申请 4 个字节内存，强转为 int*类型 。如果不动态申请内存，则程序运行报错
    p=(int*)malloc(4);
    *p=3;
    printf("%d\n",*p);
    //回收内存
    free(p);
    printf("%d\n",*p);
    system("pause");
}
```

运行结果如图：



Tips : 通过运行结果可以看到在申请内存前 p 指针是一个野指针指向一个不确定值，当调用 free(p)方法，指针又指向一个不确定值。

2.6.4 案例-动态分配数组长度

```
#include<stdio.h>
main(){
    // 动态录入数组
    // 先录入数组的长度
    int len;
    printf("请录入数组的长度\n");
    scanf("%d",&len);
    //动态申请内存空间
    int* iarray =malloc(sizeof(int)*len);
    int i;
    for(i=0;i<len;i++){
        printf("录入第%d个元素的值\n",i) ;
        scanf("%d",iarray+i);
    }
    printArray(iarray,len);
    int length;//扩展的长度
    printf("请录入要扩展数组的长度\n");
    scanf("%d",&length);

    // 参数 1 代表之前申请内存的地址 参数 2 重新多大的空间
    // 重新动态申请内存 扩展之前的内存
    iarray =realloc(iarray,sizeof(int)*(length+len));
    for(i=len;i<len+length;i++){
        printf("录入第%d个元素的值\n",i) ;
        scanf("%d",iarray+i);
    }
    printArray(iarray,len+length);
    printf("录入完成\n");
    system("pause");
}
```

Tips：上面的代码用到了两个函数，malloc (int) 和 realloc(int*,int)，第一个函数是申请一个内存空间，第二个函数是在已有空间基础上在增加部分内存空间。printArray 函数用于打印数组，在之前的代码中写过就不重复给出。

2.7 typedef&宏定义

2.7.1 typedef 的使用

typedef 作为类型定义关键字，用于在原有数据类型（包括基本类型、构造类型和指针等）的基础上，由用户自定义新的类型名称。

用法演示如下：

```
#include<stdio.h>
//使用 typedef 给 int 类型起个别名 I，然后 I 的使用跟 int 就类似
typedef int I;
typedef int NUM[100];
main(){
    I i = 1;
    //效果类似 int arr[100];
    NUM arr;
    printf("%d\n",i);
    system("pause");
}
```

Tips :

- ◆ (1) typedef 可以声明各种类型名，但不能用来定义变量。用 typedef 可以声明数组类型、字符串类型，使用比较方便。
- ◆ (2) 用 typedef 只是对已经存在的类型增加一个类型名，而没有创造新的类型。
- ◆ (3) 当在不同源文件中用到同一类型数据（尤其是像数组、指针、结构体、共用体等类型数据）时，常用 typedef 声明一些数据类型，把它们单独放在一个头文件中，然后在需要用到它们的文件中用#include 命令把它们包含进来，以提高编程效率。
- ◆ (4) 使用 typedef 有利于程序的通用与移植。有时程序会依赖于硬件特性，用 typedef 便于移植。

2.7.2 宏定义

#define 命令是 C 语言中的一个宏定义命令，它用来将一个标识符定义为一个字符串，该标识符被称为宏名，被

定义的字符串称为替换文本。

该命令有两种格式：一种是简单的宏定义，另一种是带参数的宏定义。

(1) 简单的宏定义：

`#define <宏名> <字符串>` 例：`#define PI 3.1415926`

(2) 带参数的宏定义

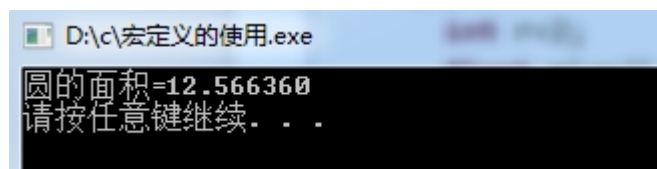
`#define <宏名> (<参数表>) <宏体>` 例：`#define MUL(A,B) ((A)*(B))`

一个标识符被宏定义后，该标识符便是一个宏名。这时，在程序中出现的是宏名，在该程序被编译前，先将宏名用被定义的字符串替换，这称为宏替换，替换后才进行编译，宏替换是简单的替换。

2.7.3 案例-宏定义的简单使用

```
#include<stdio.h>
#define PI 3.14159
#define MUL(A,B) ((A)*(B)*(B))
main(){
    int r=2;
    float mianji = MUL(PI,r);
    printf("圆的面积=%f\n",mianji);
    system("pause");
}
```

运行结果如下：



2.8 函数指针

函数指针是指向函数的指针变量。因而“函数指针”本身首先应是指针变量，只不过该指针变量指向函数。这正如用指针变量可指向整型变量、字符型、数组一样，这里是指向函数。如前所述，C 在编译时，每一个函数都有一个

入口地址，该入口地址就是函数指针所指向的地址。有了指向函数的指针变量后，可用该指针变量调用函数，就如同用指针变量可引用其他类型变量一样，在这些概念上是大体一致的。函数指针有两个用途：调用函数和做函数的参数。

```
#include<stdio.h>
//定义一个减法
int jianfa(int a,int b){
    return a-b;
}
int add(int a,int b){
    return a+b;
}
main(){
    //定义一个函数指针 返回值类型为 int,形参是两个 int 型
    int (*pf)(int x,int y);
    //将指针指向 add 函数
    pf=add;
    //调用函数计算
    int result = pf(1,2);
    printf("1+2=%d\n",result);
    pf=jianfa;
    result = pf(3,1);
    printf("3-1=%d\n",result);
    system("pause");
}
```

2.9 结构体

结构体是由基本数据类型构成的、并用一个标识符来命名的各种变量的组合。

定义结构变量的一般格式为：

```
struct 结构名

{

    类型 变量名;

    类型 变量名;

    ...
}
```

} 结构变量;

```
#include<stdio.h>
//定义一个名为的 Student 的结构体，并且创建一个变量名 student
struct Student{
    int age;
    float score;
    char* name;
}student;
main(){
    //创建一个变量为 s 的结构体对象，并赋值
    struct Student s = {23,89.5,"张三"};
    //给结构体变量 student 赋值
    student.age=24;
    student.score=80;
    student.name="李四";
    printf("张三的个人信息: \n");
    //调用结构体对象的属性
    printf("年龄=%d\n 分数=%f\n 姓名=%s\n",s.age,s.score,s.name);
    printf("李四的个人信息:\n");
    printf("年龄=%d\n 分数=%f\n 姓名=%s\n",student.age,student.score,student.name);
    system("pause");
}
```

2.10 枚举

C 语言的枚举跟 Java 的枚举功能是相似的。

方法一：先声明变量，再对变量赋值

```
#include<stdio.h>
/* 定义枚举类型 */
enum DAY {
    MON = 1, TUE, WED, THU, FRI, SAT, SUN
};

void main()
{
    /* 使用基本数据类型声明变量，然后对变量赋值 */
}
```

```
int x, y, z;
x = 10;
y = 20;
z = 30;
/* 使用枚举类型声明变量，再对枚举型变量赋值 */
enum DAY yesterday, today, tomorrow;
yesterday = MON;
today = TUE;
tomorrow = WED;
printf("%d %d %d \n", yesterday, today, tomorrow);
}
```

方法二：声明变量的同时赋初值

```
#include <stdio.h>
/* 定义枚举类型 */
enum DAY {
    MON = 1, TUE, WED, THU, FRI, SAT, SUN
};
void main()
{
    /* 使用基本数据类型声明变量同时对变量赋初值 */
    int x = 10, y = 20, z = 30;
    /* 使用枚举类型声明变量同时对枚举型变量赋初值 */
    enum DAY yesterday = MON,
    today = TUE,
    tomorrow = WED;
    printf("%d %d %d \n", yesterday, today, tomorrow);
}
```

方法三：定义类型的同时声明变量，然后对变量赋值

```
#include <stdio.h>
/* 定义枚举类型，同时声明该类型的三个变量，它们都为全局变量 */
enum DAY {
    MON = 1, TUE, WED, THU, FRI, SAT, SUN
} yesterday, today, tomorrow;
/* 定义三个具有基本数据类型的变量，它们都为全局变量 */
int x, y, z;
void main()
{
    /* 对基本数据类型的变量赋值 */
}
```

```
x = 10;
y = 20;
z = 30;
/* 对枚举型的变量赋值 */
yesterday = MON;
today = TUE;
tomorrow = WED;
printf("%d %d %d \n", x, y, z); //输出: 10 20 30
printf("%d %d %d \n", yesterday, today, tomorrow); //输出: 1 2 3
}
```

方法四：类型定义，变量声明，赋初值同时进行

```
#include <stdio.h>
/* 定义枚举类型，同时声明该类型的三个变量，并赋初值。它们都为全局变量 */
enum DAY
{
    MON = 1,
    TUE,
    WED,
    THU,
    FRI,
    SAT,
    SUN
}
yesterday = MON, today = TUE, tomorrow = WED;
/* 定义三个具有基本数据类型的变量，并赋初值。它们都为全局变量 */
int x = 10, y = 20, z = 30;
void main()
{
    printf("%d %d %d \n", x, y, z); //输出: 10 20 30
    printf("%d %d %d \n", yesterday, today, tomorrow); //输出: 1 2 3
}
```

2.11 联合体

当多个数据需要共享内存或者多个数据每次只取其一时，可以利用联合体(union)。

- ◆ 1)联合体是一个结构；
- ◆ 2)它的所有成员相对于基地址的偏移量都为 0；

- ◆ 3)此结构空间要大到足够容纳最"宽"的成员；
- ◆ 4)其对齐方式要适合其中所有的成员；

联合体分析：

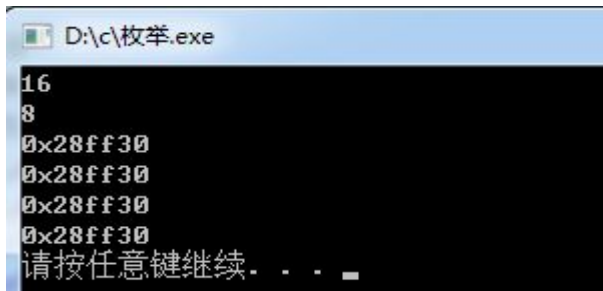
```
union
{
    char s[9];
    int n;
    double d;
} U;
```

在上面代码中：s 占 9 字节，n 占 4 字节，d 占 8 字节，因此其至少需 9 字节的空间。然而其实际大小并不是 9，用运算符 sizeof 测试其大小为 16。这是因为这里存在字节对齐的问题，9 既不能被 4 整除，也不能被 8 整除。因此补充字节到 16，这样就符合所有成员的自身对齐了。从这里可以看出联合体所占的空间不仅取决于最宽成员，还跟所有成员有关系，即其大小必须满足两个条件：1)大小足够容纳最宽的成员；2)大小能被其包含的所有基本数据类型的大小所整除。

测试程序：

```
#include <stdio.h>
main()
{
    //定义两个结构体
    union{char s[9];int n;double d;}U1;
    union{char s[5];int n;double d;}U2;
    printf("%d\n",sizeof(U1));
    printf("%d\n",sizeof(U2));
    printf("0x%x\n",&U1);
    printf("0x%x\n",&U1.s);
    printf("0x%x\n",&U1.n);
    printf("0x%x\n",&U1.d);
    system("pause");
}
```

运行上面代码结果如下：



```
D:\c\枚举.exe
16
8
0x28ff30
0x28ff30
0x28ff30
0x28ff30
请按任意键继续. . . _
```

通过上面代码测试发现：1) 联合体 U1 的内存占用长度是 16 个字节，因为 U1 的 s 占 9 个字节，double 占 8 个字节，按理说整个联合体应该是 9 才对，但是 9 不是 int（4 个字节）的整数倍，因此占用了两个 double 字节的长度。U2 内存占用长度是 8 个字节。2) 联合体的地址跟联合体内部成员的地址相同

至此，本文档完！

2015 年 1 月 26 日 星期一 15:32:32
北京市海淀区中关村软件园国际软件大厦