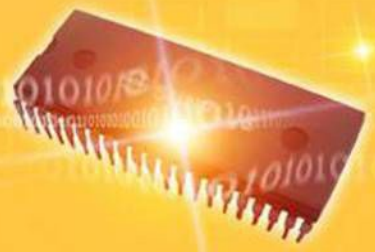
凌阳教育
www.sunplusedu.com
值得信赖的教育品牌

嵌入式系统工程师



Tel: 400-705-9680 , Email: edu@sunplusapp.com , BBS: bbs.sunplusedu.com

数组、函数

Tel: 400-705-9680 , Email: edu@sunplusapp.com , BBS: bbs.sunplusedu.com

大纲

- 数组的基本概念及应用
  - 数组的基本概念
  - 数值数组(一维、二维)
  - 字符数组（一维、二维）
- 函数的基本概念与应用
- 变量的存储类别

大纲

- 数组的基本概念及应用
  - 数组的基本概念
  - 数值数组(一维、二维)
  - 字符数组（一维、二维）
- 函数的基本概念与应用
- 变量的存储类别

数组的基本概念

- 在程序设计中，为了方便处理数据

  把具有**相同类型**的若干**变量**按**有序形式**组织起来—称为**数组**

- 数组属于**构造**数据类型

  1. 一个数组可以分解为多个数组元素

     这些数组元素可以是基本数据类型或构造类型.

     ```
     int a[10];   struct stu boy[10];
     ```

  2. 按数组元素类型的不同，数组可分为

     数值数组、字符数组、指针数组、结构数组等类别

     ```
     int a[10]   char s[10]   char *p[10];
     ```

数组的基本概念

① int a[10] = {1, 2, 3, 4, 5, 6};

② int b[4][2]={ {1, 2}, {3, 4}, {5, 6}, {7, 8} };

③ char string1[8] = { 'a' ,' b' ,' c' };

④ char string2[8] = { "abcdefg" };

⑤ char *q[10] = {&a[0], &a[1], &a[2]};

⑥ struct student boy[5];

⑦ int (*sum[10])(int x, int y);

大纲

- 数组的基本概念及应用
  - 数组的基本概念
  - 数值数组(一维、二维)
  - 字符数组（一维、二维）
- 函数的基本概念与应用
- 变量的存储类别

数值数组(一维、二维)

- **一维数值数组:**

```
short a[10];   int b[10];    long c[10];
float b[10];   double e[10];
```

- **二维数值数组:**

```
int    a[2][3]={ {80,75,78},{61,65,99} };
float b[2][3]={
                    {80.5, 75.5, 78.5},
                    {61.5, 65.5, 99.5}
              };
```

数值数组(一维、二维)

➢ 一维数组的定义:

1. 符合标识符的书写规定(数字、英文字母、下划线)
2. 数组名不能与其它变量名相同，以下是错误的:

```
int main( )
{
    int num;
    float num[10];
}
```

3. 方括号中常量表达式表示数组元素的**个数**

   如int a[3]表示数组a有3个元素

   其下标从0开始计算，因此3个元素分别为a[0],a[1],a[2]

4. 不能在方括号中用变量来表示元素的个数

```
int n=10;
int b[n];//这种表达式是错误的
```

数值数组(一维、二维)

➢ 一维数组的初始化:

—在定义数组的同时进行赋值，称为初始化

1. 定义数组时可以逐个列出数组的值

例如: int a[5]={1, 2, 3, 4, 5};

2. 给全部元素赋值时,可以不给出数组元素的个数(下标值)

例如: int a[5]={1,2,3,4,5};

可写为: int a[ ]={1,2,3,4,5};

3. 可以只给部分元素赋初值

例如: int a[10]={0,1,2,3,4};

只给a[0]~a[4]5个元素赋值，后5个元素自动赋0

4. 全局数组若不初始化，编译器将其初始化为零.

局部数组若不初始化，内容为随机值.

01.array_1.c

```c
#include <stdio.h>
int main()
{
    int i;
    float chinese_score[3]={85.5,78.5,90.5};
    float english_score[]={90,88,50};

    float math_score[5]={65,80,78};
    float computer_score[5];

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

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

    printf("请输入计算机的成绩(5个)");
    for(i=0;i<5;i++)
        scanf("%f",&computer_score[i]);

    for(i=0;i<5;i++)
        printf("%f\n",computer_score[i]);
    return 0;
}
```

二维数组

➢ 二维数组

C语言允许构造**多维数组**，多维数组元素有多个下标，以标识它在数组中的位置，所以也称为多下标变量;

在实际问题中有很多量是二维的或多维的，例如代数中的矩阵、图像采集与显示;

本小节将绍二维数组，多维数组可由二维数组类推而得到.

二维数组

➢ 二维数组的定义

```
int a[3][4];
```

1. 命名规则同一维数组

2. 定义了一个**三行四列**的数组，数组名为a

   其元素类型为整型，该数组的元素个数为3×4个，即：

   a[0][0], a[0][1], a[0][2], a[0][3]
   a[1][0], a[1][1], a[1][2], a[1][3]
   a[2][0], a[2][1], a[2][2], a[2][3]

3. 二维数组在概念上是二维的

   其下标在两个方向上变化，对其访问一般需要两个下标

4. 二维数组实际的硬件存储器是连续编址的

   即放完一行之后顺次放入第二行

二维数组

➢ 二维数组的初始化

例如：数组a[2][3]

1.按行**分段赋值**可写为：

```
int a[2][3]={ {80,75,92},{61,65,71} };
```

2.按行**连续赋值**可写为：

```
int a[2][3]={ 80,75,92,61,65,71 };
```

二维数组

- 注意：（同一维数组）

  1. 定义数组时可以逐个列出数组的值（初始化）
  2. 可以只给部分元素赋初值，未初始化则为0
  3. 全局数组若不初始化，编译器将其初始化为零.
     局部数组若不初始化，内容为随机值.
  4. 给全部数据初始化时，**行下标**可以省略

```
int a[][3]={ {80,75,92},{61,65,71} };
```

凌阳教育 www.sunplusedu.com

二维数组

```c
#include <stdio.h>
int main()
{
    float a[5][3]={{80,75,56},{59,65,71},{59,63,70},{85,45,90},{76,77,45}};
    int i,j,person_low[3]={0};
    float s=0,lesson_aver[3]={0};
    for(i=0;i<3;i++)
    {
        for(j=0;j<5;j++)
        {
            s=s+a[j][i];
            if(a[j][i]<60)
                person_low[i]++;
        }
        lesson_aver[i]=s/5;
        s=0;
    }
    printf("各科的平均成绩:\n");
    for(i=0;i<3;i++)
        printf("%.2f\n",lesson_aver[i]);
    printf("各科不及格的人数:\n");
    for(i=0;i<3;i++)
        printf("%d\n",person_low[i]);
    return 0;
}
```

02.array_2.c

二维数组：　五行、三列

行代表人：　老大到老五

列代表科目：语、数、外

大纲

- 数组的基本概念及应用
  - 数组的基本概念
  - 数值数组(一维、二维)
  - 字符数组（一维、二维）
- 函数的基本概念与应用
- 变量的存储类别

字符数组（一维、二维）

- 字符数组的定义

```
char c1[] ={ 'c' ,' ' ,' p' ,' r' ,' o' ,' g' };
char c2[] =  "c prog" ;

char a[][5] = {
                { 'B' ,' A' ,' S' ,' I' ,' C' },
                { 'd' ,' B' ,' A' ,' S' ,' E' }
              };
 char a[][6] = { "hello" , "world" };
```

字符数组（一维、二维）

➢ 字符数组的引用

1. 用字符串方式赋值比用字符逐个赋值要多占1个字节，用于存放字符串结束标志‘\0’；
2. 上面的数组c2在内存中的实际存放情况为：

'c'	' '	'p'	'r'	'o'	'g'	'\0'

注：'\0'是由C编译系统自动加上的

3. 由于采用了'\0'标志，字符数组的输入输出将变得简单方便.

字符数组

- 例

```
int main( )
{
    char str[15];
    printf( “input string:\n” );
    scanf( “%s” ,str);
    printf( “output:%s\n” ,str);
    return 0;
}
```

字符数组

```c
#include <stdio.h>
int main()
{
    float a[5][3]={{80,75,56},{59,65,71},{59,63,70},{85,45,90},{76,77,45}};
    char lesson[][10]={"语文","数学","外语"};
    int i,j,person_low[3]={0};
    float s=0,lesson_aver[3]={0};
    for(i=0;i<3;i++)
    {
        for(j=0;j<5;j++)
        {
            s=s+a[j][i];
            if(a[j][i]<60)
                person_low[i]++;
        }
        lesson_aver[i]=s/5;
        s=0;
    }
    for(i=0;i<3;i++)
        printf("%s的平均成绩%.2f\n",lesson[i],lesson_aver[i]);
    printf("\n");
    for(i=0;i<3;i++)
        printf("%s不及格的人数%d\n",lesson[i],person_low[i]);
    return 0;
}
```

03.array_char_2.c
加入字符串提示

字符数组

- C专门为字符数组的输入输出设置了一组函数

```
#include <stdio.h>
int main( )
{
    char str[15]="";
    gets(str);
    puts(str);
    return 0;
}
```

- gets(str)与scanf(“%s” ,str)的区别:

  gets(str)允许输入的字符串含有空格.

  scanf(“%s” ,str)不允许含有空格.

大纲

- 数组的基本概念及应用
- 函数的基本概念与应用
  - 函数的基本概念与定义
  - 函数的声明
  - 多文件函数
- 变量的存储类别

函数的基本概念

- C程序是由函数组成的，函数是程序的基本模块，通过**对函数模块的调用**实现特定的功能.
- 函数从定义的角度来看，可分为:

库函数和用户定义函数

- 库函数:

  编译器提供的可在c源程序中调用的函数。为两类:

  一类是**c语言标准库**规定的库函数

  一类是**编译器**提供的的库函数

- 用户自定义函数:

  用户把**自己的算法**编成一个个相对独立的函数模块，通过调用的方法来使用函数.

函数的基本概念

➢ 库函数

1. 库函数并不是C语言的一部分，它是由人们根据需要编制并提供给用户使用的

2. ANSI C标准给大家提供了一些建议使用的、通用的标准库函数，这些库函数在大多数编译系统中所应用

3. 每一个C编译系统也会单独提供一批库函数，不同的编译系统所提供的库函数的数目和函数名以及功能并不完全相同.

函数的基本概念

- 输入、输出函数: include <stdio.h>
  printf/scanf/getchar/puts 等
- 字符串函数: include <string.h>
  strcat/strcmp/strcpy 等
- 动态存储分配函数
  malloc/calloc/free/realloc 等
- 数学处理函数: include <math.h>
  绝对值/正弦/余弦/开方等

注: 可以参考 《linux C函数.chm》

函数的基本概念

➢ 自定义函数

1. 函数的一般形式:

```
[函数类型] 函数名(参数类型 参数名)
{
    数据定义部分;
    执行语句部分;
}
```

函数说明

函数体

说明:

1. [函数类型]：函数返回值的类型，默认为int型
2. 函数名由用户定义的标识符，代表函数的首地址
3. 无参函数，函数名后的括号中一般为void

函数的基本概念

- 最简单的函数（无参、无返回值）

04.fun1.c

```c
#include <stdio.h>
void hello(void)
{
    printf("hello world\n");
}

int main(void)
{
    hello();
    return 0;
}
```

函数的基本概念与应用

➢有参、有返回值的函数:

05.fun2.c

```
1  #include <stdio.h>
2  int max(int x,int y)
3  {
4      return x>y?x:y;
5  }
6  int main(void)
7  {
8      int a=10,b=20,max_num;
9      max_num=max(a,b);
10     printf("max_num=%d\n",max_num);
11     return 0;
12 }
```

说明:

1. 实参可以是常量、变量或表达式.
2. 在函数被调之前，形参不占内存，必须指定形参的类型.
3. 实参对形参的数据传递是“值传递”，即单向传递.
4. 用return语句返回值，不指定函数类型，默认返回整型值.

大纲

- 数组的基本概念及应用
- 函数的基本概念与应用
  - 函数的基本概念与定义
  - 函数的声明
  - 多文件函数
- 变量的存储类别

函数的声明

- **函数的声明原因**
  - C语言编译系统是从上往下编译源文件的
  - 被调函数放在主调函数后面，前面就该有声明，不然C由上往下的编译系统将无法识别
  - 函数声明是对调用函数的说明，以通知系统在其它函数中所调用的函数是什么类型，并宣告自己的存在
- **函数声明与定义的区别**
  - 函数定义：是指对函数功能的确立，指定函数名、函数返回值类型、参数类型、参数名、函数体.
  - 函数声明：只对已定义的函数进行说明，指定函数名、函数返回值类型、参数类型.

函数的声明

➢ 函数声明格式： 06.fun3.c

```
1  #include <stdio.h>
2  int max(int x,int y);    //形式1
3  //int max(int ,int );    //形式2
4  int main(void)
5  {
6      int a=10,b=20,max_num;
7      max_num=max(a,b);
8      printf("max_num=%d\n",max_num);
9      return 0;
10 }
11 int max(int x,int y)
12 {
13     return x>y?x:y;
14 }
```

函数的声明

- 函数声明的注意事项:

1. 主函数与被调函数在同一个文件中:

   被调函数在主函数之前定义，可以不声明

2. 主函数与被调函数不在同一个文件中:

   调用前必须进行函数声明.

   加入关键字: extern

3. 库函数的声明

   使用库函数必须包含相应的头文件

4. 一个函数只能被定义一次，但可以声明多次

大纲

- 数组的基本概念及应用
- 函数的基本概念与应用
  - 函数的基本概念与定义
  - 函数的声明
  - 多文件函数
- 变量的存储类别

多文件函数

- **多文件函数**
  - 将不同功能的函数单独分离开，在相应的源文件中定义
  - 需要的时候，把它与主调函数的主程序代码一起编译，达到**代码重用**的效果
  - 让程序设计者将重心放在**更加创新**的东西上，不要在**既有的基础**上浪费过多精力。

多文件函数

➢ 多文件函数:

main.c

```
#include <stdio.h>
int main(void)
{
    int a=10,b=20,max_num,min_num;
    max_num=max(a,b);
    min_num=min(a,b);
    printf("max_num=%d\n",max_num);
    printf("min_num=%d\n",min_num);
    return 0;
}
```

fun.c

```
int max(int x,int y)
{
    return x>y?x:y;
}
int min(int x,int y)
{
    return x<y?x:y;
}
```

多文件函数

多文件函数

上面我们将max和min函数的实现放在了fun.c文件中，在main.c的main函数中进行调用，达到了分文件的目的

但是仍有一个问题——声明

main.c中因为没有max、min函数的实现，所以**无法得知max、min的类型**，所以需要在main.c中进行函数声明

- 不同文件的声明有两种方式:

  直接声明法、头文件声明法

多文件函数

- 函多文件函数： 07.fun

main.c

```
#include <stdio.h>
extern int max(int x,int y);
extern int min(int x,int y);
int main(void)
{
    int a=10,b=20,max_num,min_num;
    max_num=max(a,b);
    min_num=min(a,b);
    printf("max_num=%d\n",max_num);
    printf("min_num=%d\n",min_num);
    return 0;
}
```

fun.c

```
int max(int x,int y)
{
    return x>y?x:y;
}
int min(int x,int y)
{
    return x<y?x:y;
}
```

- 把函数声明直接放在每个使用该函数的源文件中
- 函数调用关系简单时建议使用

多文件函数

- 函多文件函数:

main.c

```c
#include <stdio.h>
#include "fun.h"
int main(void)
{
    int a=10,b=20,max_num,min_num;
    max_num=max(a,b);
    min_num=min(a,b);
    printf("max_num=%d\n",max_num);
    printf("min_num=%d\n",min_num);
    return 0;
}
```

fun.c

```c
int max(int x,int y)
{
    return x>y?x:y;
}
int min(int x,int y)
{
    return x<y?x:y;
}
```

fun.h

```c
1 extern int max(int x,int y);
2 extern int min(int x,int y);
```

- 把函数声明放在头文件中，在主函数中包含相应头文件
- 函数调用关系复杂时使用（推荐）

多文件函数

- **练习:**

  使用分函数、分文件的方法实现:

  从屏幕上输入一个学生的成绩(0-100):

  对学生成绩进行评定:

  <=60为"E"

  60~69为"D"

  70~79为"C"

  80~89为"B"

  90以上为"A"

  <0或>100提示成绩输入出错

  提示:参数为输入的成绩，返回值为评定结果，所有信息打印均在主函数完成

大纲

- 数组的基本概念及应用
- 函数的基本概念与应用
- 变量的存储类别
  - 局部变量
  - 全局变量
  - const变量
  - register变量
  - volite变量

变量的存储类别

- **局部变量:**

  1.普通局部变量（自动变量）

  - 在一个函数内定义，只在函数范围内有效
  - 在复合语句中定义，只在复合语句中有效
  - 随着函数调用的结束或复合语句的结束而消亡
  - 初值：如果没有赋初值，内容为随机

  2.静态局部变量 static

  - 作用域：定义的函数内有效
  - 生命周期：在定义的整个周期，静态局部变量始终存在着，即使退出函数，仍然存在
  - 初值：若未赋以初值，则由系统自动赋值；数值型变量自动赋初值0，字符型变量赋空字符。

变量的存储类别

➢ 08.static.c

```
1  #include <stdio.h>
2  void fun(void)
3  {
4      static int f = 1;
5      int w = 1;
6      w = w + 2;
7      f = f + 2;
8      printf("w=%d, f=%d\n",w,f);
9
10 }
11 int main()
12 {
13     int i;
14     for(  i = 0 ; i <5 ;i++)
15         fun();
16     return 0;
17 }
```

变量的存储类别

➢ 全局变量

1. 普通全局变量

- 在函数外定义，可被本文件及其它文件中的函数所共用，若其它文件中的函数调用此变量，须用extern声明
- 生命周期：在程序运行的整个周期都存在
- 不同文件的全局变量不可重名

2. 静态全局变量 static

- 在函数外定义，作用范围被限制在所定义的文件中
- 不同文件静态全局变量可以重名，但作用域不冲突
- 生命周期：整个程序运行的周期

变量的存储类别

- 09.var_static_extern

main.c

```
1  #include <stdio.h>
2  extern int va;
3  extern int getG(void);
4  extern int getO(void);
5  int main(void)
6  {
7      printf("va=%d\n",va);
8      printf("getO=%d\n",getO());
9      printf("getG=%d\n",getG());
10     printf("%d", va*getO()*getG());
11 }
```

fun1.c

```
1  int va = 7;
2  int getG(void)
3  {
4      int va = 20;
5      return va;
6  }
```

fun2.c

```
1  static int va = 18;
2  static int getG(void)
3  {
4      return va;
5  }
6  int getO(void)
7  {
8      return getG();
9  }
```

变量的存储类别

- **注意事项:**

1. 允许在不同的函数中使用相同的变量名，它们代表不同的对象，分配不同的单元，互不干扰

2. 同一源文件中,允许全局变量和局部变量同名，在局部变量的作用域内，全局变量不起作用.

变量的存储类别

- const型变量
  - 一个变量声明为const变量，意味着该变量是一个常量，不可以被修改
- 注意:
  - const变量在定义的时候进行初始化
  - 在使用过程中，const修饰的变量只能做右值而不能做左值
  - 一般用于修饰一些不想程序中对其值发生改变的变量

变量的存储类别

- **register型变量**
  - 告诉系统register修饰的变量将被频繁使用，对其分配地址时尽量将其分配在寄存器中,以提高访问速度.
- **注意事项:**
  1. 这个修饰词只是告知cpu**尽量**将变量分配在寄存器中，不一定真的分配（可能优化处理）
  2. register变量必须是一个单个的值，并且其长度应小於或等於整型的长度。
  - 局部变量和形参可以作为register变量，全局变量\静态变量不行
  - register变量可能不存放在内存中，不能用取址符运算符“&”来获取register变量的地址

变量的存储类别

- volatile型变量（易失变量）
  - 表示变量是易失的，易变的
  - 强制访存操作，防止编译器去优化，告诉编译器每次必须去内存中取值，而不是从寄存器或者缓存
- 使用情况：
  1. 并行设备的硬件寄存器（如：状态寄存器）
  2. 一个中断服务子程序中会访问到的非自动变量(全局变量、静态变量)
  3. 多线程应用中被几个任务共享的变量

凌阳教育
www.sunplusedu.com

值得信赖的教育品牌

Tel: 400-705-9680 , Email: edu@sunplusapp.com , BBS: bbs.sunplusedu.com