**《C 语言程序设计》**

**（第二版）**

**教案**

**北京出版社**

**课时分配表**

|  |  |  |  |
| --- | --- | --- | --- |
| **章序** | **课程内容** | **课时** | **备注** |
| **1** | **认识 C 程序** | **6** |  |
| **2** | **C 程序设计基础** | **7** |  |
| **3** | **顺序结构程序设计** | **5** |  |
| **4** | **选择结构程序设计** | **3** |  |
| **5** | **循环结构程序设计** | **6** |  |
| **6** | **数组** | **5** |  |
| **7** | **用户自定义函数** | **7** |  |
| **8** | **编译预处理** | **5** |  |
| **9** | **指针** | **9** |  |
| **10** | **构造数据类型** | **5** |  |
| **11** | **文件读写技术** | **6** |  |
| **12** | **C 语言综合实训** | **4** |  |
| **13** | **附录** | **4** |  |
| **总计** |  | **72** |  |

**第7课 用户自定义函数**

|  |  |  |
| --- | --- | --- |
| **课 题** | 用户自定义函数 | |
| **课 时** | 7课时（315 min）。 | |
| **教学目标** | **知识技能目标：**  1．了解模块化程序设计思想。  2．通过学习与练习掌握函数的定义和调用。  **思政育人目标：**  让学生通过用户自定义函数，能与他人进行良好的沟通，具备良好的沟通素养和合作意识。 | |
| **教学重难点** | **教学重点：**模块化程序设计思想  **教学难点：**函数的定义和调用 | |
| **教学方法** | 讲授法、问答法、讨论法 | |
| **教学用具** | 电脑、投影仪、多媒体课件、教材 | |
| **教学设计** | 第1节课：考勤（2min）--知识讲解（40min）--作业布置（3min）  第2节课：知识讲解（40min）--课堂小结（3min）--作业布置（2min）  第3节课：知识讲解（40min）--课堂小结（3min）--作业布置（2min）  第4节课：知识讲解（40min）--课堂小结（3min）--作业布置（2min）  第5节课：知识讲解（40min）--课堂小结（3min）--作业布置（2min）  第6节课：知识讲解（40min）--课堂小结（3min）--作业布置（2min）  第7节课：知识讲解（40min）--课堂小结（3min）--作业布置（2min） | |
| **教学过程** | **主 要 教 学 内 容 及 步 骤** | **设计意图** |
| **考勤**  **（2min）** | ■【教师】清点上课人数，记录好考勤  ■【学生】班干部报请假人员及原因 | 培养学生的组织纪律性,掌握学生的出勤情况 |
| **知识讲解**  （40min） | **【教师】**模块化程序设计思想  前面单元里介绍的程序，都是由一个 main( ) 函数构成的，程序的所有操作都在 main( ) 函数中完成。  一个较大的程序，一般都会分解成若干个模块编写，每个模块都负责完成一定的功能，这个模块又称作子程序。在 C 语言中，子程序的功能是由函数来完成的。图 7-1 是一个函数调用的示意图。  C 函数有以下特点：  1. 结构上，一个 C 程序可由一个 main 函数和若干个其他函数构成，由 main 函数调用其他函数，其他函数也可以相互调用（但不能调用main 函数），而且同一个函数也可以被重复调用多次。  2. 执行顺序上，C 程序从 main 函数开始，到 main 函数结束。  3. 函数定义上，函数之间不可以嵌套定义，但可以嵌套调用。  函数是实现程序模块化的必要手段。合理的编写函数，可以简化程序结构，提高程序的可读性，减少重复编码的工作量。    **【学生】**思考、讨论。 | **展示模块化程序设计思想，让学生更加仔细的阅读，从而激发学生的学习欲望。** |
| **作业布置**（3min） | **【教师】**布置课后作业  简述C 函数的特点。 | 通过课后练习，使学生巩固所学新知识 |
| **知识讲解**  （40min） | **【教师】**函数的定义和调用  **任务描述**  编写一个自定义函数 max，用来求两个整数的大者。用 main( ) 函数调用它。  **任务分析**  1. main( ) 函数的功能：  （1）输入两个整数；  （2）调用 max 函数；  （3）输出大者。  2. max( ) 函数的功能：  （1）接收主程序传来的两个整数；  （2）求两个数的大者；  （3）将大者返回给 main( ) 函数。  **预备知识**  **一、函数的定义**  函数定义的一般格式为：  函数类型 函数名 ( 形式参数表列 )    【说明】  （1）函数类型  函数类型规定了函数返回值的类型。当函数的类型与 return 语句中的表达式的数据类型不同时，就以函数类型为准，把返回值的类型自动转换成函数的类型。对不需要使用返回值的函数，应定义为 void 类型。C 语言规定，int 类型与 char 类型的函数在定义时可以省略函数类型，系统默认这些函数返回值的类型是 int 类型。  （2）函数名  函数名是由用户命名的标识符，命名规则与变量名相同。  （3）形式参数  形式参数简称形参，它出现在被调用函数中，必须为变量名。它的功能是接收由实参传来的值，以完成函数的功能。形式参数表列可以为空，表示没有参数，也可以由多个参数组成。当形式参数表列中有多个参数时，参数与参数之间用逗号隔开。  （4）函数体  大括号“{ }”内的部分叫函数体，它是一个复合语句，函数体可以由说明部分、执行部分和返回部分组成，函数体可以为空，调用空函数将不产生任何有效操作。  （5）说明部分  除形式参数外，函数体中的其他变量必须在使用前先定义。  （6）函数的返回值  return 语句的功能是终止函数的运行，并将函数值带回调用函数。它的一般形式为：  return 表达式 ;  或  return ( 表达式 );  如果调用函数需要使用被调函数的返回值，那么在被调函数中必须包含 return 语句。当然，如果不需要从被调函数那里得到值，被调函数中也可以没有 return 语句。在函数中允许有多个 return 语句，但每次调用只有第一个 return 语句被执行，因此只能返回一个函数值。  例如，下面是一个求两个整数之和的函数。  int sum(int a,int b)  {  int total;  total=a+b;  return (total);  }  **二、函数的调用**  在定义了函数后，就可以在其他函数体里调用这个函数。  1. 函数调用的格式  函数调用的一般格式如下：  函数名 ( 实际参数表列 );  【说明】  （1）实际参数简称实参，它出现在调用函数中。实参可以是常量、变量、表达式、函数等，无论实参是何种类型的量，在进行函数调用时，它们都必须具有确定的值。当发生函数调用时，主调函数把实参的值传给被调函数的形参。  （2）如果调用无参函数，小括号内为空，但括号不能省；如果有多个实参，各实参之间用逗号隔开。  （3）实参与形参在内存中分占不同的存储单元，它们的名字可以相同，也可以不同。  （4）形参和实参在数量、类型、顺序上应严格一致，但字符型和整型可以兼容，在函数调用时，实参的值赋给与之对应的形参。  （5）形参和实参均只在自己的函数内有效。  求两个整数之和的函数调用方法如下。  void main()  {  int sum(); /\* 函数声明语句 , 此处可以不要此句 \*/  int x,y,z;  printf(" 请输入两个整数 :");  scanf("%d,%d",&x,&y);  z=sum(x,y); /\* 函数调用语句 \*/  printf(" 两个整数的和为 :%d\n",z);  }  2. 被调函数的声明  在主调函数中，要对在本函数中将要调用的函数先进行声明。  【注意】函数的“声明”与函数“定义”不是一回事。  “定义”是指对函数功能的确立，包括指定函数名，函数值类型、形参及其类型、函数体等，它是一个完整、独立的函数单位。而“说明”则是对已定义的函数的返回值进行类型说明（或称“声明”），它只包括函数名，函数值类型、以及一个空的括号，不包括形参和函数体。  函数说明的一般格式如下：  类型标识符 函数名 ( );  对被调用函数进行说明的作用是告诉系统：在本函数中将要用到某函数是××类型，也就是说明该函数返回值的类型，以便在主调函数中按此类型对函数值作相应处理。  C 语言规定：以下几种情况可以不在调用函数前对被调函数作类型说明。  （1）如果函数的类型是整型或字符型，可以不必进行说明，系统对它们自动按整型说明。  （2）如果被调用函数的定义出现在主调函数之前，可不必加以说明。  （3）如果已在所有函数定义之前，在文件开头，在函数的外部已说明了函数类型，则在各个主调函数中不必对所调用的函数再作类型说明。例如：  char letter(); /\* 以下 3 行在所有函数之前 \*/  fl oat f();  int arr();  void main()  {…} /\* 不必说明它所调用的函数类型 \*/  char letter(c1,c2) / \* 定义 letter 函数 \*/  char c1,c2;  {…}  fl oat f(x,y) /\* 定义 f 函数 \*/  fl oat x,y;  {…}  int arr(j,k) /\* 定义 arr 函数 \*/  fl oat j,k;  {…}  除了以上三种情况，都应该按上述介绍的方法，对所调用的函数作类型说明，否则编译时就会出错。  **【学生】**思考、讨论。 | **通过教师讲解，了解函数的定义和调用的基本理论知识。** |
| **课堂小结**  （3min） | 【**教师**】**回顾和总结本节课的知识点。**  **这节课我们一起学习了函数的定义和调用，让学生了解函数的类型以及特点。** | 通过对所学知识的回顾，培养学生的归纳总结能力 |
| **作业布置**（2min） | **【教师】**布置课后作业  编写一个 fac 函数，求 n!。用 main( ) 函数调用它。 | 通过课后练习，使学生巩固所学新知识 |
| **知识讲解**（45min） | **【教师】**函数间的参数传递  **任务描述**  有一个一维数组 score，内放 10 个学生成绩，求平均成绩（用数组作为函数参数）。  **任务分析**  1. main( ) 函数的功能：  （1）定义数组 score，并输入 10 个学生的成绩；  （2）调用 average 函数；  （3）输出平均成绩。  2. average( ) 函数的功能：  （1）接收主程序传来的 score 数组的首地址；  （2）计算平均值并返回。  **预备知识**  C 语言中，参数的传递方式有两种：值传递和地址传递。  值传送的特点是，实参对形参的数据传送是单向的，函数调用时，实参仅仅将其值传递给形参。当实参为变量时，形参和实参分别占着不同的存储单元，因此形参的值发生改变不会影响到实参的值。  地址传送是指传送给形参的是变量的地址，此时，形参和实参共同占用同一段主存单元，因此，只要修改形参的值，实际等同于修改了实参的值。  程序分析：  （1）值传送方式  main( ) 函数调用 swap 函数时，将变量 a 和 b 的值传递给对应的形参 x、y（x 和 y是整型变量）。在 swap 函数中，交换了形参 x、y 的值。但由于在 C 语言中，参数中的数据只能由实参单向传递给形参，形参数据的变化并不影响对应实参的值，因此不能通过 swap 函数将 main( ) 函数中的变量 a 和 b 的值交换。因此，程序的运行结果为：  a=10,b=20  值传递方式的好处是，减少了调用函数和被调用函数之间的数据依赖，增强了函数自身的独立性。  （2）地址传送方式  main( ) 函数调用 swap 函数时，将变量 a 和 b 的地址传递给对应的形参 x、y（x 和y 是指针变量，详见单元九）。此时形参和实参共占同一段内存单元。在 swap 函数中，交换了形参 x 和 y 的值，等同于交换了变量 a 和 b 的值。因此，程序的运行结果为：  a=20,b=10  在 C 语言中，数组名代表数组的首地址。因此，除了用指针变量作为参数可实现地址传送外，还可以用数组名作为函数参数。  **任务实施**  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* 程序 : program7\_2.c \*/  /\* 功能 : 一维数组 score 内放 10 个学生成绩 , 求平均成绩。( 用数组名作为参数 )\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include <stdio.h>  #defi ne N 10  fl oat average(fl oat array[],int n)  {  int i;  fl oat sum=0;  for(i=0;i<N;i++)  sum=sum+array[i];  return(sum/n);  }  void main()  {  fl oat score[N],aver;  int i;  printf(" 请输入 10 名同学的成绩 :\n");  for(i=0;i<10;i++)  scanf("%f",&score[i]);  aver=average(score,N);  printf(" 平均成绩为 : %.1f\n",aver);  }  **【学生】**思考、讨论。 | **教师通过函数间的参数传递展示，让学生了解函数间的参数传递的基本理论知识。** |
| **课堂小结**  （3min） | 【**教师**】**回顾和总结本节课的知识点。**  **这节课我们一起学习了函数间的参数传递，让学生能够积极主动地配合别人，与人为善，责任心强，具有强烈的团队合作意识和良好的沟通能力。** | 通过对所学知识的回顾，培养学生的归纳总结能力 |
| **作业布置**（2min） | 【**教师**】**布置课后作业**  **简述C 语言中参数的传递方式。** | 通过课后练习，使学生巩固所学新知识 |
| **知识讲解**（45min） | **【教师】**函数的嵌套调用  **任务描述**  利用函数嵌套调用的方法，计算12+22+32+…+n2的值。  **任务分析**  程序中涉及 3 个函数，分别是 main( ) 函数，sum 函数和 square 函数。  1. main( ) 函数的作用是负责输入 n 值、对 sum 函数的调用以及最后结果的输出；  2. sum 函数负责调用 square 函数并求12+22+32+…+n2；  3. square 函数负责求n2的值，并将值返回。  **预备知识**  C 语言中不允许作嵌套的函数定义。因此各函数之间是平行的，不存在上一级函数和下一级函数的问题。但 C 语言允许在一个函数的定义中出现对另一个函数的调用。这样就出现了函数的嵌套调用。即在被调函数中又调用其他函数。  执行过程是：执行 main 函数中调用a函数的语句时，即转去执行 a 函数，在 a 函数中调用 b 函数时，又转去执行 b 函数，b 函数执行完毕返回 a 函数的断点继续执行，a 函数执行完毕返回 main 函数的断点继续执行。  **任务实施**  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* 程序 : program7\_3.c \*/  /\* 功能 : 利用函数嵌套方法,计算12+22+32+…+n2的值。\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  long square(long a)  { long b;  b=a\*a;  return b;  }  long sum(long n)  { long c=0,d,i;  for(i=1;i<=n;i++)  {  d=square(i);  c=c+d;  }  return c;  }  void main()  { long result,n;  printf("please input n:");  scanf("%d",&n );  result=sum(n);  printf("the result is %d\n",result);  }  程序运行结果  please input n:5 ↙  the result is 55  **【学生】**思考、讨论。 | **教师通过函数的嵌套调用展示，让学生了解函数的嵌套调用的基本理论知识。** |
| **课堂小结**  （3min） | 【**教师**】**回顾和总结本节课的知识点。**  **这节课我们一起学习了函数的嵌套调用，让学生学会思考，能够与人为善，具备协作意识、创新思维与进取精神。** | 通过对所学知识的回顾，培养学生的归纳总结能力 |
| **作业布置**（2min） | 【**教师**】**布置课后作业**  **简述 main( ) 函数的作用。** | 通过课后练习，使学生巩固所学新知识 |
| **知识讲解**（45min） | **【教师】**函数的递归调用  **任务描述**  有 5 个人坐在一起，问第 5 个人多少岁。他说比第 4 个人大 2 岁。问第 4 个人多少岁，他说比第 3 个人大 2 岁。问第 3 个人多少岁，他说比第 2 个人大 2 岁。问第 2 个人多少岁，他说比第 1 个人大 2 岁。最后问第 1 个人，他说 10 岁。请问第 5 个人多少岁。  **任务分析**  一个函数在它的函数体内调用它自身称为递归调用，这种函数称为递归函数。  一个问题能用递归的方法解决，必须具备两个条件：  （1）可以将一个问题不断转换为具有相同解法的规模较小的问题；  （2）必须有明确的递归结束条件。  该任务显然是一个递归问题。  若要求得第 5 个人的年龄，必须知道第 4 个人的年龄，若要求第 4 个人的年龄，就必须知道第 3 个人的年龄，而第 3 个人的年龄又取决于第 2 个人的年龄，第 2 个人的年龄又取决于第 1 个人的年龄，每一个人的年龄又都比其前一个人的年龄大 2 岁。于是有：  age（5）=age（4）+2  age（4）=age（3）+2  age（3）=age（2）+2  age（2）=age（1）+2  age（1）=10  可以看到，当 n>1 时，求第 n 个人的年龄的公式是相同的。因此可以用一个函数来表示上述关系。    该例题的求解过程如图 7-3 所示。    可见，一个递归的问题可以分为“递推”和“回推”两个阶段。如果要求递归过程不是无限制地进行下去，必须具有一个结束递归过程的条件，此例中，这个递归结束条件是：  age(1)=10。  **【学生】**思考、讨论。 | **教师通过文函数的递归调用展示，让学生了解函数的递归调用的基本理论知识。** |
| **课堂小结**  （3min） | 【**教师**】**回顾和总结本节课的知识点。**  **这节课我们一起学习了函数的递归调用，让学生发扬斗争精神，能够在递归调用过程中开展批评与自我批评，不断完善自我。** | 通过对所学知识的回顾，培养学生的归纳总结能力 |
| **作业布置**（2min） | 【**教师**】**布置课后作业**  **简述递归函数。** | 通过课后练习，使学生巩固所学新知识 |
| **知识讲解**（45min） | **【教师】**变量的作用域  **任务描述**  分析下列程序的执行结果。  #include <stdio.h>  int n=10; /\* 全局变量 \*/  void func1()  {  int n=20; /\* 局部变量 \*/  printf("func1 n: %d\n", n);  }  void func2()  {  printf("func2 n: %d\n", n);  }  void func3()  {  printf("func3 n: %d\n", n);  }  void main()  {  int n=30; /\* 局部变量 \*/  func1();  func2();  func3();  { /\* 代码块由 {} 包围 \*/  int n = 40; /\* 复合语句中的变量 ( 局部变量 )\*/  printf("block n: %d\n", n);  }  printf("main n: %d\n", n);  }  **预备知识**  在 C 程序中，变量的作用域是有范围限制的。在空间上，反映为变量的作用域——局部变量和全局变量；在时间上，反映为变量的生存期——动态变量和静态变量（详见单元七任务 7）。本任务介绍局部变量和全局变量。  **一、局部变量**  局部变量也称内部变量。局部变量是在函数内进行定义说明的。其作用域仅限于函数内，离开该函数后再使用该变量是非法的。  关于局部变量的作用域要说明以下几点：  （1）主函数 main 中定义的变量也是局部变量，也只能在 main 函数中使用。同时，主函数中也不能使用其他函数中定义的变量。因为主函数也是一个函数，它与其他函数是平行的。这一点与其他语言不同，应予以注意。  （2）函数的形参也是局部变量。形参变量是属于被调函数的局部变量，实参变量是属于主调函数的局部变量。  （3）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，也不会发生混淆。  （4）在复合语句中也会定义变量，其作用域只在复合语句范围内有效。  **二、全局变量**  全局变量也称外部变量，它是在函数外部定义的变量。它不属于哪一个函数，而属于一个源程序文件。其作用域是整个源程序。在函数中使用全局变量，一般应作全局变量说明。只有在函数内经过说明的全局变量才能使用。全局变量的说明符为 extern。但在一个函数之前定义的全局变量，在该函数内使用可以不再加以说明。  对于全局变量需要注意以下几点：  （1）对于局部变量的定义和说明可以不加区分，而对于外部变量则不然，外部变量的定义和外部变量的说明并不是一回事。外部变量定义必须在所有函数之外，且只能定义一次，其一般形式为：  extern 类型说明符 变量名；  其中，extern 可以省略不写。  例如：  int a,b;  等效于：  extern int a,b;  外部变量在定义时就已经分配了内存单元，外部变量定义可进行初始赋值，外部变量说明则不能再赋初始值，只是表明在函数内要使用某外部变量。  （2）外部变量可加强函数模块之间的数据联系，但是因为函数要依赖这些变量，因而使得函数的独立性降低。从模块化程序设计的观点来看，这是不利的。因此，在不必要时尽量不要使用全局变量。  （3）在同一源文件中，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。  **任务实施**  程序运行结果：  func1 n: 20  func2 n: 10  func3 n: 10  block n: 40  main n: 30  程序分析：  （1）对于 func1( )，输出结果为 20，显然使用的是函数内部的 n，而不是外部的 n；func2( ) 也是相同的情况。  （2）当全局变量和局部变量同名时，在局部范围内全局变量被“屏蔽”，不再起作用。或者说，变量的使用遵循“就近”原则，如果在当前作用域中存在同名变量，就不会向更大的作用域中去寻找变量。  （3）func3( ) 输出 10，使用的是全局变量，因为在 func3( ) 函数中不存在局部变量 n，所以编译器只能到函数外部，也就是在全局作用域中去寻找变量 n。  （4）由 { } 包围的代码块也拥有独立的作用域，printf( ) 使用它自己内部的变量 n，输出 40。  （5）C 语言规定，只能从小的作用域向大的作用域中去寻找变量，而不能反过来，使用更小的作用域中的变量。对于 main( ) 函数，即使代码块中的 n 离输出语句更近，但它仍然会使用 main( ) 函数开头定义的 n，所以输出结果是 30。  **【学生】**思考、讨论。 | **教师通过变量的作用域展示，让学生了解变量的作用域的基本理论知识。** |
| **课堂小结**  （3min） | 【**教师**】**回顾和总结本节课的知识点。**  **这节课我们一起学习了变量的作用域，让学生能准确区分全局变量和局部变量，会分析它们在程序中的作用范围。** | 通过对所学知识的回顾，培养学生的归纳总结能力 |
| **作业布置**（2min） | 【**教师**】**布置课后作业**  **简述全局变量。** | 通过课后练习，使学生巩固所学新知识 |
| **知识讲解**（45min） | **【教师】**变量的生存期  **一、变量的生存期**  变量在调用时，必须分配内存单元（或寄存器），所以变量就有生存期的问题。变量什么时候分配内存单元（或寄存器），什么时候释放内存单元（或寄存器），都是值得考虑的问题。这样就引入了变量生存期的概念。  变量从开始分配单元（或寄存器）到释放所分配的单元（或寄存器）的期间被称为变量的生存期。  变量按其生存期可分为动态存储变量和静态存储变量。  静态存储变量就是在定义变量时就分配好存储单元并一直保持不变，直至整个程序结束。全局变量即属于此类存储方式。  动态存储变量是在程序执行过程中，使用它时才分配存储单元，使用完立即释放。  典型的例子就是函数的形式参数，在函数定义时并不给函数的形参分配内存单元，只是在被调用时才予以分配，调用函数完毕后立即释放。如果一个函数被多次调用，则会反复地分配、释放形参变量的存储单元。  从以上分析可知，静态存储变量是一直存在的，而动态存储变量则时而存在时而消失。  **二、变量的存储类型**  为了表示变量的存储位置、作用域和生存周期这 3 种属性，C 语言中将变量按存储类型分为 4 类：auto，register，extern，static。  下面分别介绍这 4 种存储方式。  1. 自动变量 auto  auto 存储类型是 C 语言中应用最广泛的类型。C 语言规定，函数内凡未加存储类型说明的变量均视为自动变量，即自动变量可以省去说明符 auto。在前面各单元的程序中所定义的变量凡未加存储类型说明符的都是自动变量。例如：  {  int i,j,k;  char c;  ……  }  等价于  {  auto int i,j,k;  auto char c;  ……  }  自动变量有以下几个特点：  （1）自动变量的作用域仅限于定义该变量的个体。在函数中定义的自动变量，只在该函数内有效；在复合语句中定义的自动变量只在复合语句内有效。  （2）自动变量属于动态存储方式，只有在使用它（即定义该变量的函数被调用时）才给它分配存储单元，开始它的生存期。函数调用结束，释放存储单元，结束生存期。因此，函数调用结束之后，自动变量的值不保留。在复合语句中定义的自动变量，在退出复合语句之后也不能使用。  （3）由于自动变量的作用域和生存期都局限于定义它的个体（函数或复合语句内），因此，不同的个体中允许使用同名的变量而不会混淆。  （4）对构造类型的自动变量，如数组等，不可作初始化赋值。  2. 寄存器变量 register  当对一个变量频繁读写时，必须反复访问内存储器，从而花费大量的存取时间。为此，C 语言提供了另一种变量，即寄存器变量。这种变量存放在 CPU 的寄存器中，使用时，不需要访问内存，而直接从寄存器中读写，这样可提高执行效率。寄存器变量的说明符是 register。对于循环次数较多的循环控制变量及循环体内反复使用的变量均可定义为寄存器变量。  对寄存器变量还有以下几点说明：  （1）只有局部自动变量和形式参数才可以定义为寄存器变量。因为寄存器变量属于动态存储方式。凡需要采用静态存储方式的变量都不能定义为寄存器变量。  （2）在 Turbo C、MS C 等微机上使用的 C 语言中，实际上是把寄存器变量当成自动变量处理的，因此速度并不能提高。而在程序中允许使用寄存器变量是为了和标准 C保持一致。即使能真正使用寄存器变量的机器，由于寄存器中 CPU 的个数是有限的，因此使用寄存器变量的个数也是有限的。  3. 外部变量 extern  在一个文件中，定义在所有函数之外的变量称为外部变量。前面介绍全局变量时已经介绍过外部变量，这里再补充说明外部变量的几个特点：  （1）外部变量和全局变量是对同一类变量的两种不同角度的提法。全局变量是从它的作用域出发的，外部变量是从它的存储方式出发的，表示了它的生存期。  （2）当一个源程序由若干个源文件组成时，在一个源文件中定义的外部变量在其他源文件中也有效。  4. 静态变量 static  静态变量是静态分配的变量，其类型说明符为 static。在编译时，在特定的存储区为其分配存储空间，所分配的存储空间在程序中自始至终都归该变量使用。静态变量属于静态存储方式，但是属于静态存储方式的不一定就是静态变量。例如，外部变量虽然属于静态存储方式，但不一定是静态变量，必须由 static 加以定义后才能成为静态外部变量，或称静态全局变量。静态变量分为静态局部变量和静态全局变量两种。  （1）静态局部变量  在局部变量的说明前再加上 static 说明符就构成静态局部变量。  例如：  static int a;  static int array[3]={1,2,3};  静态局部变量属于静态存储方式，它具有以下特点：  ① 静态局部变量在函数定义内，不像自动变量那样，当调用时就存在，退出函数时就消失。静态局部变量始终存在着，即它的生存期为整个源程序。  ② 静态局部变量的生存期虽然为整个源程序，但是其作用域仍然与自动变量相同。即只能在定义它的函数内使用该变量。退出该函数后，尽管静态局部变量还继续存在，但不能使用它。  ③ 允许对构造类静态局部变量赋初值。若未赋以初值，则系统自动赋以 0 值。  ④ 对基本类型的静态局部变量若在说明时未赋以初值，则系统自动赋以 0 值。而对自动变量不赋初值，则其值是不定的。  根据静态局部变量的特点，可以看出它是一种生存期为整个源程序的变量。虽然离开定义它的函数后不能使用，但如果再次调用定义它的函数时，它又可继续使用，而且保留了前次被调用后留下的值。因此，当多次调用一个函数且要求在调用之间保留某些变量的值时，可以考虑采用静态局部变量。虽然用全局变量也能达到上述目的，但全局变量有时会造成意外的副作用，因此建议采用静态局部变量。  （2）静态全局变量  在全局变量（外部变量）的说明之前再冠以 static，就构成了静态全局变量。全局变量本身就是静态存储方式，静态全局变量当然也是静态存储方式。它们在存储方式上并无不同，区别在于：非静态全局变量的作用域是整个源程序，当一个源程序由多个源文件组成时，非静态的全局变量在各个源文件中都是有效的，而静态的全局变量则限制了其作用域，即只在定义该变量的源文件内有效，在同一源程序的其他源文件中就不能使用它。由于静态全局变量的作用域局限于一个源文件内，只能为该源文件内的函数共用，因此可以避免在其他源文件中引起错误。  **知识拓展**  用函数实现单元五“知识拓展”的算术练习程序。  根据用户选择的菜单项，由计算机为小学生随机出10 道算术  题，每题10 分。小学生输入答案后，程序自动判断正确与否，最后给出小学生答对题目的总分。算术运算有加、减、乘、除四种。  程序分析：  本例采用函数来实现。在单元五“知识拓展”的基础上，将加、减、乘、除4 种算术运算分别写成add( )、sub( )、mul( )、div( )4 个函数，main 函数为主控程序，根据用户的选择，调用相应的函数。其程序模块结构如图7-4 所示。  程序运行后的主界面及选择加、减运算后的运行界面分别如图7-5、图7-6、图7-7所示。    请根据运行情况，阅读理解下列程序。  源程序：  #include <stdio.h>  #include <stdlib.h>  #include <time.h>  void add(); /\* 函数声明 \*/  void sub();  void mul();  void div();  void main()  {  int op;  srand(time(NULL));  while(1)  {  printf("\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n\n");  printf(" 1---ADD(+) \n");  printf(" 2---SUB(-) \n");  printf(" 3---MUL(\*) \n");  printf(" 4---DIV(/) \n");  printf(" 0---exit \n");  printf("\n\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n");  printf(" 请选择(0-4):");  scanf("%d",&op);  switch(op)  {  case 1: add();break;  case 2: sub();break;  case 3: mul();break;  case 4: div();break;  case 0: exit(0); /\* 函数exit(0) 的作用是正常退出程序\*/  }  }  }  void add()  {  int i,num1,num2,answer,grade=0;  for(i=1;i<=10;i++)  { num1=10+rand()%90;  num2=10+rand()%90;  printf("%d+%d=",num1,num2);  scanf("%d",&answer);  if(answer==num1+num2) grade=grade+10;  }  printf(" 总分:%d",grade);  }  void sub()  {  int i,num1,num2,answer,grade=0;  for(i=1;i<=10;i++)  { num1=10+rand()%90;  num2=10+rand()%90;  printf("%d-%d=",num1,num2);  scanf("%d",&answer);  if(answer==num1-num2) grade=grade+10;  }  printf(" 总分:%d",grade);  }  void mul()  { }  void div()  { }  【测一测】请学生  **【学生】**思考、讨论。 | **教师通过变量的生存期展示，让学生了解变量的生存期的基本理论知识。** |
| **课堂小结**  （3min） | 【**教师**】**回顾和总结本节课的知识点。**  **这节课我们一起学习了变量的生存期，让学生能结合实际需求，在程序的时空性能上合理取舍，树立大局观。** | 通过对所学知识的回顾，培养学生的归纳总结能力 |
| **作业布置**（2min） | 【**教师**】**布置课后作业**  **简述静态局部变量特点。** | 通过课后练习，使学生巩固所学新知识 |
| **教学反思** | 要鼓励学生大胆设想，帮助学生萌发创新意识，产生创新的欲望和兴趣，同时采用探索式和任务驱动式教学，培养学生的创新能力。 | |