02 C语言概述

内容提要

- 运算符：=
- 函数：main()，printf()
- 编写一个简单的 C 程序
- 创建整形变量，为其赋值，并在屏幕上显示该值
- 换行字符
- 如何在程序中加入注释，建立包含多个函数的程序，发现程序中的错误
- 什么是关键字

简单的C程序示例

1 简单的C程序示例

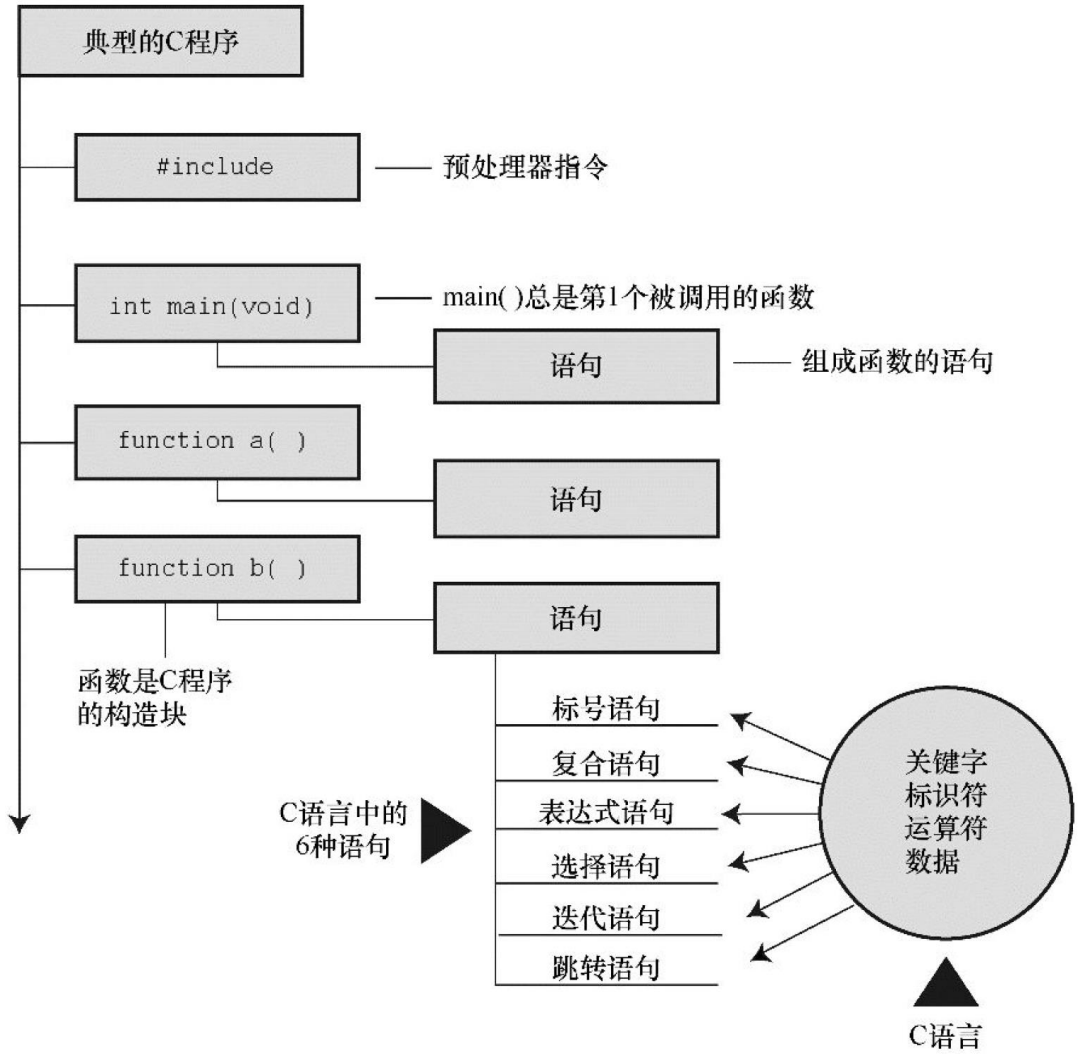
- 程序清单2.1 first.c

- 重要的最初概念
  - 程序（的大致）结构
  - 变量
    - 变量申明
    - 变量赋值
  - 信息输出到屏幕

```c
// first.c
#include <stdio.h>
int main(void)  /* a simple program */
{
    int num;    /* define a variable called num */
    num = 1;    /* assign a value to num */

    printf("I am a simple "); /* use the printf */
    printf("computer.\n");
    printf("My favorite number is %d.\n", num);

    return 0;
}
```

示例解释

2 示例说明



- 1. declaration
- 2. assignment
- 3. function
- 4. control
- 5. null

- 1.关键字 2.标识符 3.运算符 4.数据

2.1 快速概要

- #include<stdio.h> 包含stdio.h
- stdio.h是C编译器软件包的标准部分，它提供键盘输入和屏幕输出的支持。

```
int main(void)
/* 一个简单的C程序 */ ←注释
//一行结束的注释
{ ←函数体开始，左花括号表示函数定义开始，右花括号（}）表示函数定义结束
    int num; ←声明
    num = 1; ←赋值表达式语句
    printf("I am a simple "); ←调用一个函数
    printf("My favorite number is %d because it is first.\n", num);
    return 0; ←return语句，看作是结束main()函数的要求
} ←结束
```

```c
// first.c
#include <stdio.h>
int main(void) /* a simple program */
{
    int num;    /* define a variable called num */
    num = 1;    /* assign a value to num */

    printf("I am a simple "); /* use the printf() */
    printf("computer.\n");
    printf("My favorite number is %d.\n",num);

    return 0;
}
```

2.2 第二遍 程序细节

- #include 指示和头文件
- main()
- 注释

#include指示和头文件

- 预处理（preprocessing）
  - 在编译前要对源代码做一些准备工作，即预处理（preprocessing）
- #include 语句：C预处理指令（preprocessor directive）
- #include <stdio.h>
  - 相当于把stdio.h文件中的所有内容都输入该行所在的位置
- stdio.h头文件
  - C程序顶部的信息集合。提供了标准输入/输出头文件等信息
  - 在C程序顶部的信息集合被称为头文件（header）

- ANSI/ISO C规定了C编译器必须提供哪些头文件
  - 使用哪些函数需要包含哪些头文件
- 为什么不内置输入输出语句？
  - 精简C。经济使用资源（C语言的哲学之一）

main()函数

- C语言的基础逻辑单位是函数
  - 有且有一个main函数
  - 其它函数

- int main(void)
  - 不考虑例外情况，C程序总是从称为 main()的函数开始执行
  - 返回值，给操作系统
    - 表明程序运行结果的状态
- void main()
  - C99和C11标准中不认可

注释

- 被/* */两个符号括起来的部分是程序的注释
  - 一行
  - 也可以多行

- // 注释限制在一行内
  - // 这种注释只能写成一行
  - int rigue; // 这种注释也可置于此。

- /*和*/配对

花括号，程序体和代码块

- C函数用花括号标记函数体的开始和结束
  - 花括号 { }， 可以
  - 圆括号（ ）和方括号 [ ]，不行
- 花括号还可将函数中的多条语句合并为一个单元或块

```
{
}
```

声明与变量 - 关键字、标识符、变量

- 关键字（keyword）
  - 语言定义的单词，不能做其他用途。如 int
- 标识符（identifier）
  - 变量、函数或其他实体的名称。如 num
- int num;
  - 声明语句（declaration statement）
  - 在函数中有一个名为 num 的变量
  - int说明 num 是一个整数类型
  - 编译器使用声明变量 num在内存中分配一个合适的存储空间
- 在C语言中，所有变量都必须先声明才能使用
  - 必须列出程序中用到的所有变量名及其类型
  - C99遵循C++的惯例，允许把声明放在代码块中的任何位置

```
1.  int main() // 目前的C规则
2.  {
3.      // 一些语句
4.      int doors;
5.      doors = 5; // 第1次使用doors
6.      // 其他语句
7.      int dogs;
8.      dogs = 3; // 第1次使用dogs
9.      // 其他语句
10. }
```

变量声明

- 数据类型
  - 把变量声明为整型或字符类型，计算机才能正确地存储、读取和解释数据
- 变量命名
  - 小写字母、大写字母、数字、下划线（_）；区分大小写
  - 第1个字符必须：字母或下划线，不能是数字
  - 注意
    - 变量名应使用有意义的变量名或标识符
    - C99和C11允许使用更长的标识符名，但编译器只识别前63个字符。对于外部标识符（参阅第12章）
- 声明变量的理由【为什么要引入变量机制？】
  - 通过申明变量来获取存储数据的内存空间
  - 如果事先未声明变量，C程序将无法使用标识符，无法通过编译
  - 变量名方便查找和理解程序的用途（比其它内存空间表达方式，如内存地址，要好）

变量的赋值

- num = 1; // = （赋值运算符）
  - 赋值表达式语句
  - 把值1赋给变量 num
  - 声明语句，内存中为变量 num 分配了空间
  - 赋值语句，把值存储在之前预留的位置
- 【赋值是程序设计中将计算结果保存的最基础和重要的操作】
  - 最常用的，变量赋值
  - 地址方式间接访问内存
    - 拷贝数据到指定的内存
    - 地址方式间接访问内存的赋值

printf()函数

- printf()函数
  - 实际参数（actual argument）
  - 形式参数（简称形参）是函数中用于存储值的变量
- 调用（call）或请求（invoke）一个函数
  - 输入函数名，把所需的参数填入圆括号
  - 当程序运行到这一行时，控制权将转给该函数。当函数完成了它所要做的工作，控制权换返回给原来的函数主调函数（calling function）
- 换行符是转义字符（Escape Sequence）
  - \开始
  - \t代表Tab键，\b代表Backspace键（退格键）
- %d相当于是一个占位符，其作用是指明输出num值的位置

```
1. printf("I am a simple ");
2. printf("computer.\n");
3. printf("My favorite number is %d because it is
   first.\n", num);
```

```
printf("That's mere contrariness");
```

实际参数

return 语句

- return 返回语句是函数的最后一个语句
- 有返回值的C函数要使用一个 return 语句
- int表明main()函数应返回一个整数

- 强烈建议：main()函数中保留return语句

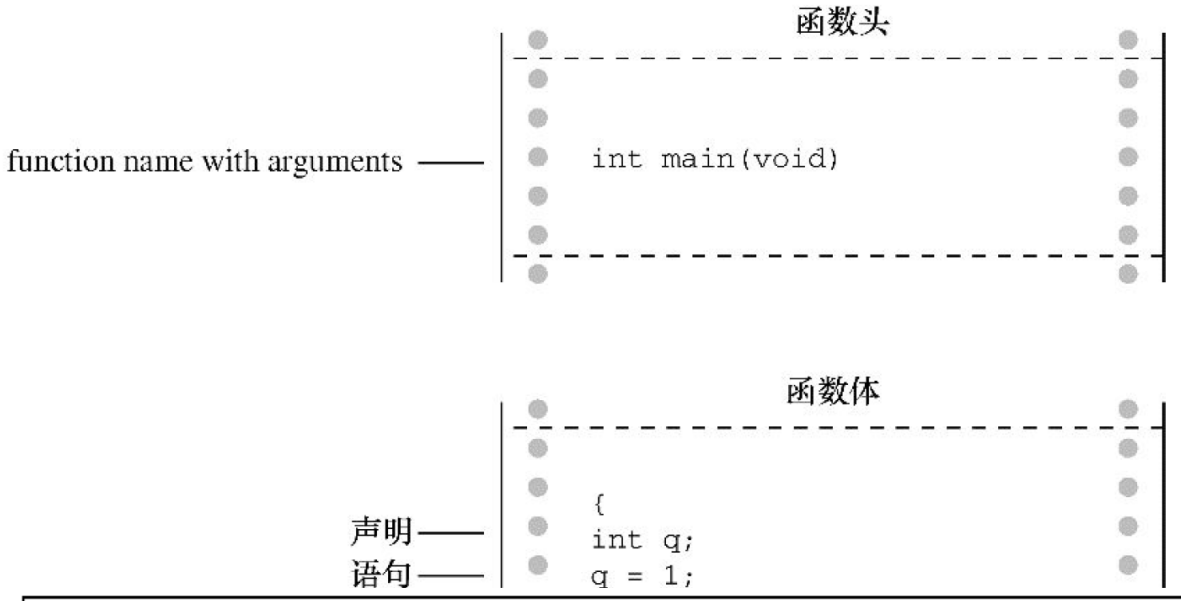
```c
// first.c
#include <stdio.h>
int main(void)
{
    int num;
    num = 1;

    printf("I am a simple ");
    printf("computer.\n");
    printf("My favorite number is %d.\n", num);

    return 0;
}
```

一个简单程序的结构

3 一个简单程序的结构

- 程序（program）由一个或多个函数组成
  - 其中必须一个名为 main()的函数。

- 函数：函数头和函数体组成
  - 函数头包括函数名、传入该函数的信息类型和函数的返回类型。通过函数名后的圆括号可识别出函数，圆括号里可能为空，可能有参数
  - 函数体（body）位于花括号（{}）中并由一系列语句组成，每个语句以一个分号结束




```
#include <stdio.h>
int main(void)
{
      语句
      return 0;
}
```

（大部分语句都以分号结尾。）

提高程序可读的技巧

4 提高程序可读的技巧

- 选择有意义的变量名
- 使用注释
- 在函数中用空行分隔概念上的多个部分
- 每条语句各占一行

```
int main(void) /* 把2英寻（测水深的单位）转换成英尺*/

{
int feet, fathoms;  ——————————— 使用有意义的变量名
                    ——————————— 使用空行
fathoms=2;
feet=6*fathoms;  ——————————— 每行一条语句
printf("There are %d feet in %d fathoms!\n", feet, fathoms);
return 0;
}
```

进一步使用C

5 进一步使用C

- 程序清单2.2 fathm_ft.c
- 声明了多个变量，进行了乘法运算，然后输出两个变量的值

```c
// fathm_ft.c -- converts 2 fathoms to feet

#include <stdio.h>
int main(void)
{
    int feet, fathoms;

    fathoms = 2;
    feet = 6 * fathoms;
    printf("There are %d feet in %d fathoms!\n", feet, fathoms);
    printf("Yes, I said %d feet!\n", 6 * fathoms);

    return 0;
}
```

5.1 说明

- 在开始处用一个注释（新形式的注释）
  - 说明了文件的名称和程序的目的
  - 在以后浏览或打印程序时很有帮助

```
1.  // fathm_ft.c -- converts 2 fathoms to feet

2.  #include <stdio.h>
3.  int main(void)
4.  {
5.      int feet, fathoms;
6.
7.      fathoms = 2;
8.      feet = 6 * fathoms;
9.      printf("There are %d feet in %d fathoms!\n", feet,
    fathoms);
10.     printf("Yes, I said %d feet!\n", 6 * fathoms);
11.
12.     return 0;
13. }
```

5.2 多个声明

- 一个声明语句里声明了两个变量而不是一个。
- 在声明语句中需要用一个逗号把两个变量分开

```
1.  // fathm_ft.c -- converts 2 fathoms to feet

2.  #include <stdio.h>
3.  int main(void)
4.  {
5.      int feet, fathoms;
6.
7.      fathoms = 2;
8.      feet = 6 * fathoms;
9.      printf("There are %d feet in %d fathoms!\n", feet,
    fathoms);
10.     printf("Yes, I said %d feet!\n", 6 * fathoms);
11.
12.     return 0;
13. }
```

5.3 乘法

- 用*表示乘法

```c
// fathm_ft.c -- converts 2 fathoms to feet

#include <stdio.h>
int main(void)
{
    int feet, fathoms;

    fathoms = 2;
    feet = 6 * fathoms;
    printf("There are %d feet in %d fathoms!\n", feet, fathoms);
    printf("Yes, I said %d feet!\n", 6 * fathoms);

    return 0;
}
```

5.4 打印多个值

- 待输出的变量列于双引号的后面。变量之间要用逗号隔开

```c
// fathm_ft.c -- converts 2 fathoms to feet

#include <stdio.h>
int main(void)
{
    int feet, fathoms;

    fathoms = 2;
    feet = 6 * fathoms;
    printf("There are %d feet in %d fathoms!\n", feet, fathoms);
    printf("Yes, I said %d feet!\n", 6 * fathoms);

    return 0;
}
```

多个函数

6 多个函数

- 2.3 tow_fuc.c
- butler()函数在程序中出现了3次
  - 函数原型。告知编译器要用到的该函数
  - 函数调用。在main()函数中函数调用的形式出现
  - 函数定义。函数本身的源代码
- 函数原型是一种声明形式，告知编译器正在使用某函数，也被称为函数声明（function declaration）
  - 函数原型还指明了函数的属性
- 何时执行butler()函数取决于它在main()中被调用的位置，而不是butler()的定义在文件中的位置
- C标准建议，为用到的所有函数提供函数原型。
- 标准include文件（包含文件）为标准库函数提供了函数原型

```
1.  //* two_func.c -- a program using two functions in one
    file */
2.  #include <stdio.h>
3.  void butler(void);/*ANSI/ISO C function prototyping */
4.  int main(void)
5.  {
6.      printf("I will summon the butler function.\n");
7.      butler();
8.      printf("Yes. Bring me some tea and writeable
    DVDs.\n");
9.
10.     return 0;
11. }

12. void butler(void)/* start of function definition */
13. {
14.     printf("You rang, sir?\n");
15. }
```

调试

7 调试

- 程序的错误通常叫做bug，发现和修正这些错误的过程叫做调试（debug）

- 1 语法错误
- 2 语义错误
  - 程序逻辑错误。编译器无法检测语义错误

```
1.  /*  nogood.c -- a program with errors */
2.  #include <stdio.h>
3.  int main(void)
4.  (
5.   int n, int n2, int n3;
6.
7.  // this program has several errors
8.   n = 5;
9.   n2 = n * n;
10.  n3 = n2 * n2;
11.  printf("n = %d, n squared = %d, n cubed = %d\n", n,
    n2, n3)
12.
13.  return 0;
14.  )
```

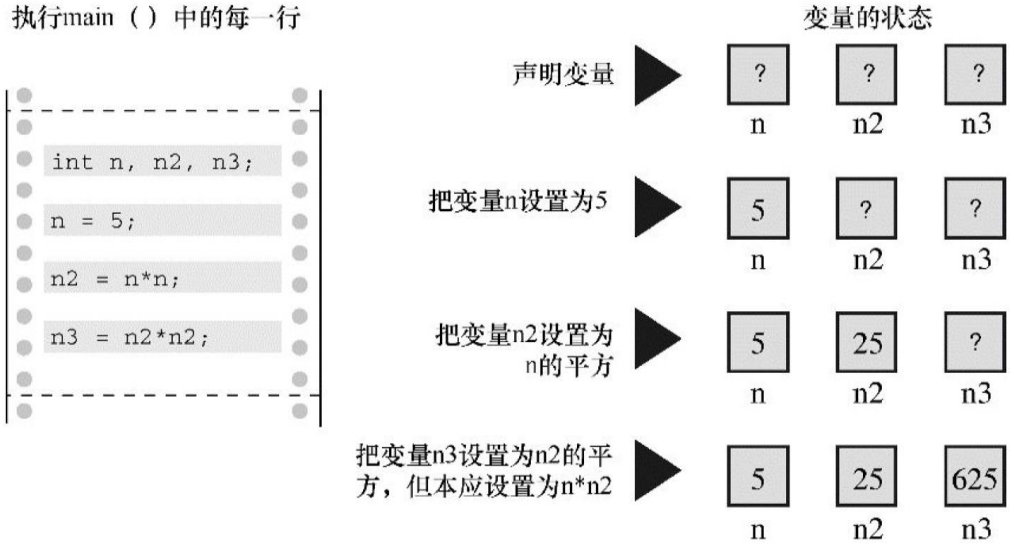
7 调试 - 语法错误

- 违反C语言的规则
  - main()函数体使用圆括号来代替花括号
  - 变量声明
  - printf()语句末尾漏掉了分号
- 方法
  - 首先，在编译之前，浏览源代码看是否能发现一些明显的错误。
  - 接下来，查看编译器是否发现错误，检查程序的语法错误是它的工作之一。
- TIPS
  - 在编译程序时，编译器发现错误会报告错误信息，指出每一处错误的性质和具体位置
  - 编译器也有出错的时候。也许某处隐藏的语法错误会导致编译器误判

```
1.  /*  nogood.c -- a program with errors */
2.  #include <stdio.h>
3.  int main(void)
4.  (
5.   int n, int n2, int n3;
6.
7.  // this program has several errors
8.   n = 5;
9.   n2 = n * n;
10.  n3 = n2 * n2;
11.  printf("n = %d, n squared = %d, n cubed = %d\n", n,
     n2, n3)
12.
13.  return 0;
14.  )
```

7 调试 - 语义错误

- 语义错误是指意思上的错误
  - 遵循了C规则，但是结果不正确，
- n3 = n2 * n2;
- 更系统的发现语义错误方法
  - 把自己想象成计算机，跟着程序的步骤一步一步地执行




```c
/*  nogood.c -- a program with errors */
#include <stdio.h>
int main(void)
(
 int n, int n2, int n3;

// this program has several errors
 n = 5;
 n2 = n * n;
 n3 = n2 * n2;
 printf("n = %d, n squared = %d, n cubed = %d\n", n,
n2, n3)

 return 0;
 )
```

7 调试 - 程序状态

- 通过逐步跟踪程序的执行步骤，记录每个变量，便可监视程序的状态。
- 程序状态（program state）是在程序的执行过程中，某给定点上所有变量值的集合。它是计算机当前状态的一个快照。
- 定位语义错误
  - 自己模拟计算机逐步执行程序
  - 在程序中的关键点插入额外的 printf()语句，以监视制定变量值的变化。
  - 使用调试器。调试器（debugger）是一种程序，让你一步一步运行另一个程序，并检查该程序变量的值

```
1.  /*  nogood.c -- a program with errors */
2.  #include <stdio.h>
3.  int main(void)
4.  (
5.   int n, int n2, int n3;
6.
7.  // this program has several errors
8.   n = 5;
9.   n2 = n * n;
10.  n3 = n2 * n2;
11.  printf("n = %d, n squared = %d, n cubed = %d\n", n,
     n2, n3)
12.
13.  return 0;
14.  )
```

关键字和保留标识符

8 关键字和保留标识符

- 关键字
  - C语言中的词汇。不能将它们用作标识符
  - 使用关键字不当（如，用关键字作为变量名），编译器会将其视为语法错误。

- 保留标识符（reserved identifier）
  - C语言已经指定了它们的用途或保留它们的使用权
  - 保留标识符包括那些以下划线字符开头的标识符和标准库函数名，如printf()。

表2.2 ISO C关键字

auto	extern	short	while
break	float	**signed**	***_Alignas***
case	for	sizeof	***_Alignof***
char	goto	static	***_Atomic***
const	if	struct	***_Bool***
continue	*inline*	switch	***_Complex***
default	int	typedef	***_Generic***
do	long	union	***_Imaginary***
double	register	unsigned	***_Noreturn***
else	restrict	**void**	***_Static_assert***
enum	return	**volatile**	***_Thread_local***

关键概念

9 关键概念

- 理解什么是C程序
  - 程序是对希望计算机采取何种行为的描述
  - 编译器负责完成把描述转换成底层的机器语言的细节工作
- 由于编译器不具有真正的智能，所以必须用编译器能理解的术语表达意图
  - 这些术语就是C语言标准规定的形式规则
- 编译器希望收到特定格式的指令
  - 程序员的工作就是在一个编译器能成功处理的框架内表达关于程序应采取何种行为的想法

10 总结

- C程序由一个或多个C函数组成
  - 每个C程序必须包含一个main()函数，这是C程序要调用的第1个函数
  - 简单的函数由函数头和后面的一对花括号组成，花括号中是由声明、语句组成的函数体
  - 在C语言中，大部分语句都以分号结尾
  - 声明语句为变量指定变量名，并标识该变量中存储的数据类型
  - 变量名是一种标识符
  - 赋值表达式语句把值赋给变量，或者更一般地说，把值赋给存储空间
  - 函数表达式语句用于调用指定的已命名函数。调用函数执行完毕后，程序会返回到函数调用后面的语句继续执行
- printf()函数用于输出想要表达的内容和变量的值
- 语法是一套规则，用于管理语言中各有效语句组合在一起的方式
  - 语句的语义是语句要表达的意思
  - 编译器可以检测出语法错误，但是程序里的语义错误只有在编译完之后才能从程序的行为中表现出来。检查程序是否有语义错误要跟踪程序的状态，即检查程序每执行一步后所有变量的值
- 最后，关键字是C语言的词汇