C 语言程序设计基础讲义

张雷

September 9, 2023

目录

第 1 讲 数据

内容提要

- ❑ 程序
- ❑ 编程语言
- ❑ 数值数据和符号数据
- ❑ 变量

1.1 C 编程语言简介

在二十世纪中叶之前的数千年人类历史中，人类手工收集、分析和创建数据。数字计算机不仅是技术的革命，也是文明的革命，因为它们能够存储比世界上所有纸张所能容纳的更多的数据，并且比人类军队更快、更可靠地对这些数据进行计算。如今，我们依靠复杂的计算机程序以各种方式处理数据，从与亲人来回发送消息、组织文档和媒体中的数据，到运行物理、社会和生物系统的模拟。

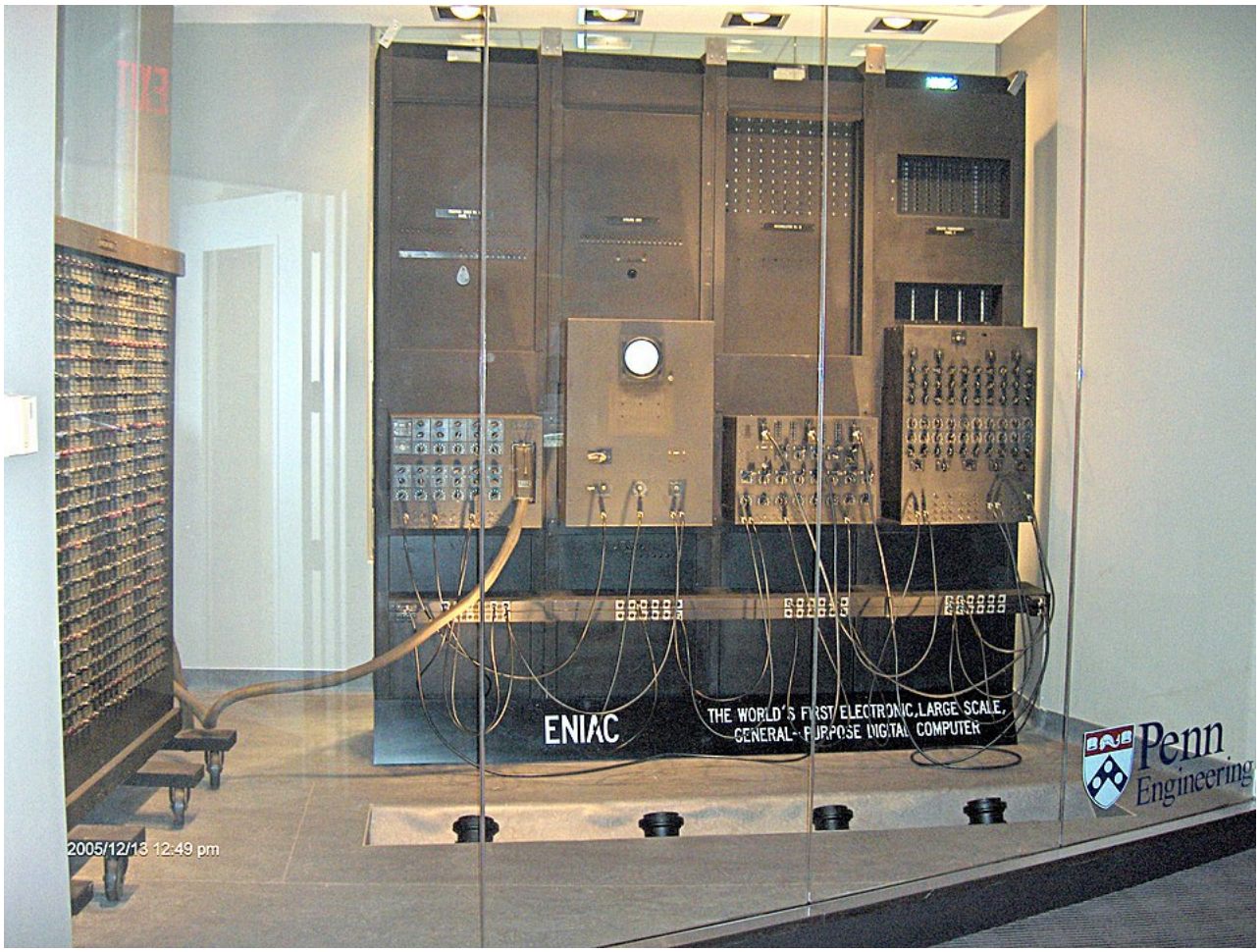

图 1.1: 人类第一台通用计算机 ENIAC 的图像。原始上传者是英文维基百科的 TexasDex。CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php ?curid=6557095 。)

然而，尽管计算机具有强大的计算能力，但它仍然存在一个根本性的限制：它们没有自主性，也没有固有的能力来决定要做什么。他们所能做的就是获取一组（可能非常复杂!）指令，我们称之为计算机程序，并执行这些指令——不多也不少。因此，如果我们作为计算机科学家想要利用计算机的强大力量，我们需要学习如何以计算机理解的方式给出这些指令。

我们需要学习如何与计算机对话。

什么是编程语言？ 正如中文英文等语言可以实现人与人之间的交流一样，*编程语言* 是向计算机传达一组指令的一种方式。与人类语言一样，编程语言由一组允许的单词以及将这些单词组合在一起形成有意义的短语的规则组成。编程语言必须足够精确才能被计算机理解，因此与任何人类语言的丰富复杂性相比，它使用相对较小的单词集和非常结构化的规则将它们组合在一起。学习编程语言一开始可能会令人沮丧，因为即使稍微偏离这些规则也会导致计算机无法理解我们编写的内容。但是，我们花在掌握编程语言规则上的时间和精力会得到丰厚的回报：计算机不仅会理解我们所说的内容，还会忠实地执行我们编写的任何指令。

程序只是我们希望告诉计算机执行的指令文本。我们将这种文本程序称为*源代码*，或简称*代码*，以区别于其他形式的文本。要使用特定的编程语言编写程序，我们需要了解该语言的两个关键属性。第一个是语言的*语法*，这是我们给管理语言中有效程序的规则的名称。编程语言语法类似于管理人类语言的语法规则，它告诉我们如何将单词组合在一起形成句子。计算机在执行程序之前，必须读取该程序的指令；编程语言的语法指定了这些指令的预期格式。

第二个概念是编程语言的*语义*，它指的是语言中不同指令含义的控制规则。这类似于人类语言中单词的含义。一旦计算机读取了程序中的指令，它就开始执行它们。语言语义指定计算机应该对每条指令做什么。

C 语言 正如当今世界上有数千种人类语言，每种语言都有自己的词汇、语法和风格约定一样，我们也有大量的编程语言可供选择。在本课程中，我们将使用 C 编程语言，它提供了

现在，我们的计算机硬件和操作系统软件都无法直接理解 C 编程语言。相反，C 语言的创建者编写了一个名为*C 语言编译器* 的程序，其工作是获取用 C 编程语言编写的程序并生成机器指令。因此，当您将 C “下载” 到计算机上时，您实际上下载并安装的是这个 C 语言编译程序。您可以将 C 语言编译器视为程序员（用 C 语言代码编写指令）与实际执行指令的计算机硬件之间的中介。

Shell 如今的计算机有着多种多样的交互接口让我们可以进行指令的的输入，从炫酷的图像用户界面（GUI），语音输入甚至是 AR/VR 都已经无处不在。这些交互接口可以覆盖 80% 的使用场景，但是它们也从根本上限制了您的操作方式——你不能点击一个不存在的按钮或者是用语音输入一个还没有被录入的指令。为了充分利用计算机的能力，我们不得不回到最根本的方式，使用*操作系统的文字接口*：Shell。

几乎所有您能够接触到的平台都支持某种形式的 shell，有些甚至还提供了多种 shell 供您选择。虽然它们之间有些细节上的差异，但是其核心功能都是一样的：它允许你执行程序，输入并获取某种半结构化的输出。本节课我们会使用 Bourne Again SHell, 简称 “bash”。这是被最广泛使用的一种 shell，它的语法和其他的 shell 都是类似的。打开 shell 提示符（您输入指令的地方），您首先需要打开终端。您的设备通常都已经内置了终端，或者您也可以安装一个，非常简单。

1.2 hello.c

程序 1.1

```c
#include<stdio.h>

int main(void)
{
    printf("To C, or not to C: that is the question.\n");
    return 0;
}
```

将上述程序保存为文本文件 hello.c，然后在 shell 程序中手动调用 gcc 编译器将其编译为可执行文件 hello：

```
gcc -o hello hello.c
```

也可以在 shell 中利用 make 工具自动完成编译过程：

```
make hello
```

本课程使用的集成开发环境 visual code，同时支持上面两种方式。

1.3 数据表示一：数值数据

数据就在我们身边，并且存储的数据量每天都在增加。在当今世界，决策必须由数据驱动，因此我们必须能够处理、分析和理解我们收集的数据。其他重要因素包括数据的安全性和隐私性。企业和政府需要回答一些重要问题，例如“这些数据应该存储在哪里?”；“这些数据应该如何存储?”；甚至，“这些数据是否应该被存储?”。如果决策必须由数据驱动，那么计算机是处理该数据的绝佳工具，

我们通过定义不同类别的数据并了解如何用 C 编程语言表示它们来开始认真学习计算机科学。在接下来的三个部分中，我们将回顾本课程中将大量使用的常见数据类型：数值数据、布尔数据、文本数据以及将多个数据组合成一个数据实体的各种形式的复合数据。我们将讨论这些独立于计算机或编程语言的数据类型，然后了解我们的数据理论概念与 C 语言中实际表示的内容之间微妙但至关重要的差异。

什么是数据类型？ 考虑到世界上数据的数量和种类，对人类和计算机来说，有办法对数据进行分类都是有用的。数据类型是对数据类别的精确描述，包含两部分：

1. 该类型数据的所有允许的值；
2. 我们可以允许对该类型的数据执行的操作。

例如，我们可以说一个人的年龄是一个自然数，这会告诉我们像 25 和 100 这样的值是可以预期的，而年龄 -2 或“David”将是无意义的。知道一个人的年龄是一个自然数还告诉我们可以执行哪些操作（例如，“年龄加 1”），并排除其他操作（例如，“按字母顺序对这些年龄进行排序”）。

重要的是，这些数据类型存在于程序语言之外——毕竟，我们拥有自然数的时间比拥有计算机的时间要长得多！同时，每种编程语言都有一种表示数据类型自己的方式，以便程序在执行计算时可以区分各种类型的数据。因此，在本节中，我们将介绍独立于编程语言的常见数据类型（有时称为抽象数据类型）的抽象版本，以及 C 语言中存在的相应的具体数据类型。许多术语和定义可能您在过去的学习中遇到过，但要小心——它们可能与您之前学到的略有不同，但是您需要完全准确地理解这些定义，这是非常重要的。

数值数据：自然数和整数 我们将从两种最常见的数值数据形式开始，它们表示没有小数部分的数字：

- 自然数是集合 $\{0, 1, \dots\}$ 中的一个值。我们使用符号 $\mathbb{N}$ 来表示自然数的集合。请注意，我们在计算机科学中的惯例是将 0 视为自然数!
- 整数是集合 $\{\dots, -2, -1, 0, 1, 2, \dots\}$ 中的一个值我们使用符号 $\mathbb{Z}$ 来表示整数的集合。

当然，自然数是整数的子集：每个自然数都是整数，但反之则不然。C 语言定义了 int 用来表示自然数和整数的数据类型。在 C 语言中，int 型数据就是带有可选负号的数字序列，如 110，-3421。

自然数和整数的算术运算 所有整数都支持熟悉的算术运算：加法 (4 + 5), 减法 (4 - 5), 乘法 (4 * 5)。它也支持除法，但这是一种特殊情况，我们将在下面更详细地讨论。

另一项您可能不太熟悉的算术运算是模运算，它指的是在一个整数除以另一个整数时产生的余数。我们将使用百分号表示模运算，写做 $x\%y$，意思是当 x 除以 y 产出的余数。例如 $10\%4 = 2$，$20\%2 = 0$。

C 编程语言支持这些算术运算，并使用了模仿数学中对应的各种运算符：

程序 1.2

```c
#include<stdio.h>

int main(void)
{
    printf("%d\n", 2 + 3);
    printf("%d\n", 2 - 5);
    printf("%d\n", -2 * 10);
    //this is modulo operation
    printf("%d\n", 10 % 4);
    return 0;
}
```

在最后一个打印命令之前，我们添加了一些附加文本：//this is modulo operation。在 C 语言中，我们使用代码中的字符//来开始注释，这是 C 语言编译器会忽略的代码。注释仅供人类阅读，是提供有关某些 C 代码的附加信息的有用方式。上面我们用它来解释运算符 % 的含义。

C 语言支持算术运算的优先级规则，在加法和减法之前执行乘法。同时就像数学一样，长表达式可能很难阅读。所以 C 语言还允许你使用括号将表达式组合在一起：

程序 1.3

```c
#include<stdio.h>

int main(void)
{
    printf("%d\n", 1 - -1);
    printf("%d\n", 1 + 4 + 2 * 3);
    printf("%d\n", (1 + (2 + 1)) - (2 + 3));
    return 0;
}
```

除法 当我们对两个整数进行加、减、乘和求幂时，结果始终是整数，因此 C 语言总是为这些运算生成一个 int 值。但是两个整数相除并不总是产生一个整数。在数学中很好办，因为我们知道如何表示分数。但这如何影响 C 语言程序的功能呢?

事实上，C 语言的除法运算符是运算符 /，称为整数除法。对于两个整数 x 和 y，x/y 的结果等于分数 $\frac{x}{y}$ 丢掉小数部分。以下是 C 语言的一些示例：

程序 1.4

```c
#include<stdio.h>

int main(void)
{
    printf("%d\n", 15 / 2);
    printf("%d\n", -15 / 2);
    return 0;
}
```

但是“真正的”除法运算 $15 \div 2 = 7.5$ 又如何表示呢？这种情况下的输出不是整数，而是一个不同数据类型的值：float 类型。C 语言使用该数据类型来表示任意实数，包括小数值。我们将稍后详细讨论实数值。但首先我们将讨论使用比较运算符的数值运算。

为了帮助您回顾，表1.1总结了迄今为止我们看到的五个整数算术运算符：

运算符	描述
a + b	产生 a 和的 b 总和
a - b	产生 a 减去 b 的结果
a * b	产生 a 乘以 b 的结果
a / b	产生 a 除以 b 的结果（截断）
a % b	a 除以 b 时产生余数

表 1.1: 整数算术运算总结

比较/关系运算符 比较两个数字时，我们有标准的数学符号 $=$ 和 $\neq$ 用于说明两个数字是否相等，以及符号 $<$, $>$, $\geq$,$\leq$ 描述两个数字中哪一个更大。

与算术运算一样，每个数学符号都有对应的 C 语言运算符：

运算符	描述
a == b	产生是否 a 和 b 相等
a != b	产生是否 a 和 b 不相等（与 == 相反）
a > b	产生是否 a 大于 b
a < b	产生是否 a 小于 b
a >= b	生成是否 a 大于或等于 b
a <= b	生成是否 a 小于或等于 b

表 1.2: 关系运算总结

这里有一些例子：

程序 1.5

```c
#include<stdio.h>

int main(void)
{
    printf("%d\n", 4 == 4);
    printf("%d\n", 4 != 6);
    printf("%d\n", 4 >= 1);
    return 0;
}
```

♣

小数和实数 现在我们来谈谈非整数。您在早期的学习中会熟悉的一些数字集合：

- 有理数是集合 $\{\frac{p}{q} | p, q \in \mathbb{Z}, q \neq 0\}$ ——即所有分数的集合。这包括像这样的数字 $\frac{3}{2}$ 和 $-\frac{4}{7}$ ，但也可以是整数，因为 $3 = \frac{3}{1}$。我们使用符号 $\mathbb{Q}$ 来表示有理数的集合。
- 无理数是具有无限且不重复小数的数。例子是 $\pi, \sqrt{2}, e$ 。我们使用符号 $\bar{\mathbb{Q}}$ 来表示无理数的集合。
- 实数要么是有理数，要么是无理数。我们使用符号 $\mathbb{R}$ 来表示实数集。

C 语言使用一种单独的数据类型 float 来表示非整数。float 常量是一个数字序列，后跟一个小数点 (.)，然后是另一个数字序列。以下是一些 float 类型数据的示例：$7.5, .123, -1000.00000001, 1e6$。

小数和实数的运算 从数学的角度来看，我们为整数描述的所有算术和关系比较运算也适用于 float 类型的值。这里有些例子：

程序 1.6

```c
#include<stdio.h>

int main(void)
{
    printf("%f\n", 2.5 + 3.4);
    printf("%f\n", 2.5 - 20.9);
    printf("%f\n", 3.5 / 2.5);
    printf("%d\n", 2.2 > 1.5);
    printf("%d\n", -2.2 > -1.5);
    return 0;
}
```

float 类型的局限性 这是一个有趣的例子。回想一下，$\frac{2}{3}$ 是一个无限循环小数。让我们看看如果我们使用所学的内容在 C 中计算 $\frac{2}{3}$ 会发生什么：

程序 1.7

```c
#include <stdio.h>

int main()
{
    printf("%.50f\n", 2.0 / 3.0 );
    //0.66666666666666662965923251249478198587894439697266
    printf("%d\n", 0.1 + 0.2 == 0.3 );
    //0

    float sum = 0.0;
    for (int i = 0; i < 100; i++)
    {
        sum += 0.1; // 重复加 0.1, 累积精度丢失
    }
    printf("%f\n",sum);

    return 0;
}
```

看到问题了吗？$\frac{2}{3}$ 其小数展开无限且循环。但是程序1.7作为在计算机上运行的程序，只有有限的计算机内存可供使用，因此无法准确表示 $\frac{2}{3}$，就像您在任何有限数量的纸张上无法写下 $\frac{2}{3}$ 所有的小数位一样。

注更准确地说，计算机使用二进制系统，其中包括数字在内的所有数据都表示为 0 和 1 的序列。这个 0 和 1 的序列是有限的，因为计算机内存是有限的，因此不能精确地表示 $\frac{2}{3}$ 。我们将在今年晚些时候讨论这种数字的二进制表示形式。

所以 C 语言程序输出的 float 值 0.66666666666666662965923251249478198587894439697266 是一个不精确的近似值。更戏剧性的（并使用我们的比较运算符进行练习）：$0.1 + 0.2 == 0.3$。这说明了 float 类型的一个基本限制：这种数据类型用于在 C 程序中表示实数，但不能总是准确地表示它们。相反，一个 float 值只是实数的值一个近似；有时这种近似是精确的，例如 2.5，但大多数时候并非如此。

混合 int 和 float 的算术运算 总结一下：对于两个 int 数值 x 和 y，所有表达式 $x+y$,$x-y$, $x*y$, x/y, $x\%y$ 都会产生 int 值。对于两个 float 数值，除了 % 不能通过编译，其它算术运算都会产生一个 float。

但是当我们混合这两种数据类型时会发生什么？给定一个 int 和一个 float 的算术运算始终会生成 float。即使在只有一个值为很长的算术表达式中只有一个 float 数值，整个表达式的计算结果也会取为 float 类型。即使结果值在数学上是整数，也是如此，如此例所示。

程序 1.8

```c
#include <stdio.h>

int main()
{
    printf("%f\n", 12 - 4 * 9.0 / (3 * 6) + 100);
    return 0;
}
```

3 和 3.0 在 C 中被存储为不同的数据类型，我们将在本课程稍后研究 int 和 float 如何存储在计算机内存中时详细探讨这一点。然而，即使和 3 具有 3.0 不同的数据类型，C 也会将它们识别为具有相同的值：

程序 1.9

```c
#include <stdio.h>

int main()
{
    printf("%d\n", 3.0 == 3);
    printf("%d\n", 3.0);
    return 0;
}
```

1.4 数据表示二：符号数据

在上一节中，我们研究了如何使用两种具体数据类型 int 和 float 在 C 语言中表示数值数据。在本节中，我们将介绍两种新的数据类型，它们代表了常见的单个数据，在下一节中我们将了解到如何将多个数据组合在一起成为一个复杂数据。

布尔数据 布尔值是集合 {**true**, **false**} 中的值。通常可以布尔值视为是/否问题的答案。例如，“这个人的年龄是否足以投票？”、“这个国家是内陆国家吗？”以及“这项服务免费吗？”。“布尔”这个词听起来可能有点奇怪！尽管“真”和“假”的概念自人类存在以来就已存在，但“布尔”一词是以逻辑学家乔治 · 布尔（George Boole，1815-1864）命名的，他是符号逻辑的早期先驱。

我们还在上一节中看到了布尔值作为比较运算的结果。当我们写 $3>1$ 的时候，该表达式的值不是数字，而是布尔值（在本例中为 true）。

在 C 语言中，布尔数据使用 bool 数据类型来表示。与我们刚刚看到的数值类型不同，它只有两种可能的值：true 和 false。要注意需要额外添加一个命令：#include <stdbool.h>

布尔数据的运算 可以使用不同类型的运算来组合布尔值。最常见的三个是：与或非。

- 逻辑与：给定两个布尔值，当两个值均为 true 时生成 true，否则生成 false。例如，“true 与 false”是 false，而“true 与 true”是 true。在符号逻辑中，该运算符由符号 $\wedge$ 表示。

- 逻辑或: 给定两个布尔值，当至少其中一个值为 true 时生成 true，否则生成 false。例如，“true 或 false”是 true，而“false 或 false”是 false。在符号逻辑中，该运算符由符号 ∨ 表示。
- 逻辑非：反转布尔值。“非 true”是 false，“非 false”是 true。在符号逻辑中，该运算符由符号 ¬ 表示。

关于逻辑或运算的一个注意事项是，当两个给定的布尔值都为 true 时，它会生成 true。这与异或（exclusive or）形成对比，后者在给定值之一恰好为 true 时生成 true。当两个给定值都为 true 时，异或运算将生成 false 。

在 C 语言中，我们有与上述运算符对应的三个逻辑运算符，分别是逻辑与 &&，逻辑或 ‖，逻辑非!。

程序 1.10

```c
#include <stdio.h>
#include<stdbool.h>

int main()
{
    printf("%d\n", !true);
    printf("%d\n", true && true);
    printf("%d\n", true && false);
    printf("%d\n", false || true);
    printf("%d\n", false || false);
    return 0;
}
```

正如我们看到算术操作符组成的表达式可以相互嵌套一样，我们可以将逻辑运算符组成的表达式组合起来，甚至包括算术比较运算符：

程序 1.11

```c
#include <stdio.h>
#include<stdbool.h>

int main()
{
    printf("%d\n", true && (false || true));
    //1
    printf("%d\n", (3 == 4) || (5 > 10));
    //0
    return 0;
}
```

下周，我们将更详细地讨论这些逻辑运算符并介绍其他一些逻辑运算符。

字符数据 本节中的第二个新数据类型用于表示字符。我们提到过计算机使用一系列的 0 和 1 来存储数据。这些 0 和 1 代表数字。那么，数字如何表示字符数据呢？曾几何时，人类通过打孔纸带（或简称打孔胶带）与计算机进行交互。磁带上特定位置的孔（或没有孔）代表 0 或 1（即二进制）。今天，我们将每个 0 或 1 称为“位”。显然，这比使用我们现代的输入外围设备（键盘、鼠标、触摸屏等）要繁琐得多。最终，确定了用孔表示字符（例如字母、数字）的标准。仅使用磁带上的 7 个位置，就可以表示 128 个不同的字符（$2^7 = 128$）。

该标准称为 ASCII（发音为 ass-key），并一直沿用至今。该标准涵盖所有英文字母（小写和大写）、数字、标点符号和各种其他符号（例如，用于换行）。例如，数字 65 映射到字母'A'，数字 33 映射到标点符号'!'。

但其他语言呢？计算机科学家将 ASCII 从长度为 7 的位序列扩展为长度为 8 的位序列，因此其域增加到大小

256（$2^8 = 256$）。这使得“扩展 ASCII”能够支持类似的基于拉丁语的语言中使用的一些其他字符，例如’é’(233)、’ö’(246)、’¥’(165) 以及其他有用的符号，例如’©’(169) 和’½’(189)。但对于不同语言（例如希腊语、普通话、阿拉伯语）中使用的字符又如何呢?

最新标准 Unicode 使用最多 32 位，为我们提供了一个超过 40 亿个不同的数字集合。事实上，这个数字比所有语言中使用的不同字符的数量还要大！这些数字中有几个未使用的数字。

但随着互联网的普及，这些未使用的数字被用来映射到表情符号。当然，这可能会导致一些翻译后原本含义丢失的问题。棕榈树表情符号在您的设备上可能与朋友的设备上显示不同。在极端情况下，您朋友的设备可能根本看不到棕榈树或看到完全不同的表情符号。该问题的一种解决方案是将新的表情符号提案给 Unicode（https://unicode.org/emoji/proposals.html）。但这也意味着计算机科学家需要通过更新软件来支持新批准的表情符号。

C 语言里用 char 类型来表示字符，一个字符常量是用单引号括起来的字符或转义符 (escape sequence)。但要注意，字符类型 char 在处理非 ASCII 字符时可能会有限制，因为它通常使用一个字节来表示字符，而 ASCII 字符只有 256 个。

转义符	表示的字符
\n	换行
\t	水平制表
\	反斜杠 (\)
\’	单引号 (’)
\”	双引号 (”)
\?	问号 (?)
\%	百分号 (%)

字符运算 char 类型的变量可以进行算术运算，例如加法和减法。字符在内部以其 ASCII 码值进行处理。例如，可以执行以下操作：

需要注意的是，虽然 char 类型可以用于存储字符，但在执行算术运算时，它们实际上是整数。

程序 1.12

```c
#include <stdio.h>

int main()
{
    printf("%c\n", 'A'); //A
    printf("%c\n", 65);  //A
    printf("%c\n", 'A'+32); //a
    return 0;
}
```

文本数据（字符串） 本节中的第三个新数据类型用于表示文本：从姓名到聊天日志再到莎士比亚的《罗密欧与朱丽叶》的文本。我们使用字符串数据类型表示文本，它是字符组成的一个序列。

所有 C 代码都是我们输入计算机的文本，那么我们如何区分代码文本和数据文本（例如人名）？在 C 语言中，字符串是由一系列字符组成的字符数组。字符串通常以空字符”（ASCII 码 0）作为结尾，这个空字符用于表示字符串的结束。一个字符串常量是用双引号括起来的字符序列。

程序 1.13

```c
#include <stdio.h>

int main()
{
    printf("%s", "hello world!\n");
    printf("%s", "\n");
    printf("%s", "");
    printf("%d\n", "hello"=="hello");
    printf("%d\n", "hello"=="world");
    return 0;
}
```

第三个例子很有趣：它是一个空字符串，一个没有字符的字符串。这在 C 中是完全允许的，并且在处理文本数据时这是一个令人惊讶的常见值，例如，调查数据里参与者将问题留空的那些值。

字符串操作 C 语言的字符串类型属于派生类型，因此并没有任何内置支持的操作。你可能会看到比较两个字符串：“hello” == “hello” 会返回 true，但我们后面将会看到，这里比较的其实是两个字符串在计算机内存中地址，因此和比较两个字符串的内容之间并没有任何关系。

1.5 数据表示三：数组

在过去的两节中，我们学习了三种常见的数据类型：数值、符号和文本。到目前为止，我们的重点是描述这些类型的单个数据：如何编写文字代码来表示这些值，以及我们可以对它们执行哪些操作。现在，我们将了解一个特殊的派生类型，数组，学习如何将多个数据聚集成为一个单个的复杂数据。

数组是零个或多个可能包含重复项的值的序列。与集合类似，数组中包含的值称为数组的元素。与集合不同，数组可以包含重复项，并且数组中的顺序很重要。事实上，在 C 语言中，字符串被简单地表示为字符数组，而不是单独的数据类型。

在数学中，数组使用方括号编写，数组中包含的每个元素用逗号分隔，例如 $[1,2,3,4]$。在 C 语言中，数组使用 {}，如 $\{1,2,3,4\}$。数组元素需要是同质的（所有元素具有相同的类型），不同数据类型的值不能放到一个数组中。

数组上的操作 在 C 语言中数组属于派生出的类型，没有内置的操作，如比较两个数组是否相等，两个数组之间相加等等，这些操作都需要用户自己实现。

程序 1.14

```c
#include <stdio.h>

int main()
{
    printf("these ops are illegal!\n");
    //printf("%d\n", {1,2,3} == {1,2,3});
    //printf("%d\n", {1,2,3} + {2,3,4});
    return 0;
}
```

1.6 在变量中存储数据

到目前为止，我们仅使用常量和运算符在 C 程序中编写表达式。但随着我们想要执行的计算变得更加复杂，仅依赖常量和运算符是非常麻烦的。我们可以编写非常复杂的嵌套表达式，但这使得我们的代码非常难以理解。

例如，假设我们在笛卡尔平面上给出了三个点 (1, 3), (2, 5), (10, -1)，形成一条路径，我们想要找到这条路径的长度。

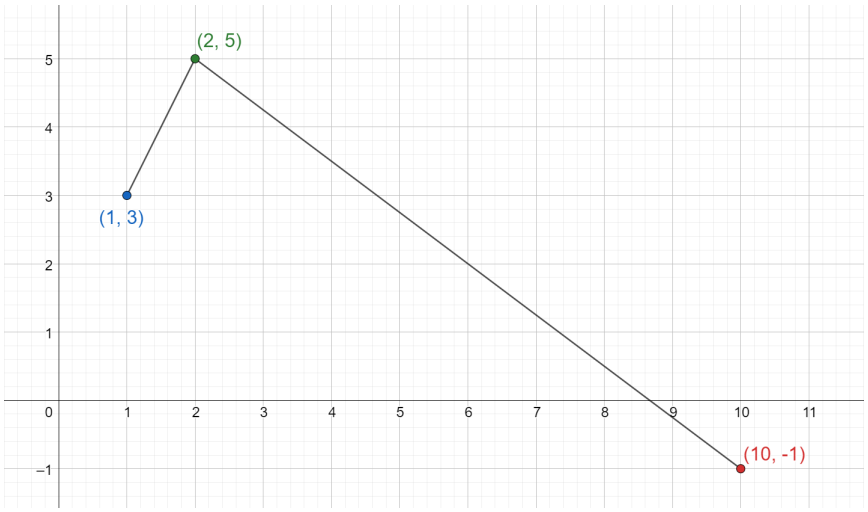

图 1.2: 显示连接点 (1, 3)、(2, 5) 和 (10, -1) 的路径的笛卡尔平面图

我们想使用这个公式来计算 (x_1, y_1) 和 (x_2, y_2) 两点之间距离:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

我们可以将路径长度计算编写为单个算术表达式，并让 C 对其进行计算:

程序 1.15

```c
#include <stdio.h>
#include <math.h>

int main()
{
    printf("%f\n", sqrt((2 - 1) * (2 - 1) + (5 - 3) * (5 - 3))
                 + sqrt((10 - 2)*(10 - 2) + ((-1) - 5) * ((-1) - 5)) );
    //12.236068
    return 0;
}
```

但输入这个表达式很容易出错并且难以理解。就像数学一样，我们可以通过将问题分解为中间步骤来改进我们的代码。在 C 语言中，我们可以通过将值与名称相关联来实现这一点，以便我们可以在后续计算中引用这些值。

变量 变量是由引用值的名称组成的一段代码。我们使用以下语法在 C 语言中创建变量:

<variable> = <expression>

这种形式的 C 语言代码，被称为赋值语句。C 程序执行赋值语句分为两步:

- 首先，计算 = 右侧的表达式，产生一个值
- 其次，该值绑定到（或“分配给”）左侧的变量。

执行赋值语句后，可以使用变量来引用值。下面是我们如何使用变量来简化上面的计算：

程序 1.16

```c
#include <stdio.h>
#include <math.h>

int main()
{
    float distance1 = sqrt((2 - 1) * (2 - 1) + (5 - 3) * (5 - 3));
    float distance2 = sqrt((10 - 2)*(10 - 2) + ((-1) - 5) * ((-1) - 5) ) ;
    float total_distance = distance1 + distance2;
    printf("%f\n", total_distance);
    return 0;
}
```

选择好的变量名 由于变量用于存储计算中的中间值，因此选择好的变量名称非常重要，以便您可以记住每个变量的用途。在我们上面的示例中，这似乎并不那么重要，因为只有两个变量，但是当您开始编写更大的程序时，您将不得不处理数十个（数百个）变量，选择好的名称至关重要。

现在，我们将介绍一些选择变量名称时应遵循的简单规则：

1. 所有变量名称只能使用小写字母、数字和下划线字符。所以 distance1，不是 Distance1。
2. 当变量名由多个单词组成时，每个单词之间用下划线分隔。不允许在变量名称中使用空格，因此使用下划线来显示分词符。例如，我们可以创建一个变量来引用总距离：

$$total\_distance = distance1 + distance2$$

   我们使用变量名 total_distance 而不是 totaldistance 或者 totalDistance。变量命名约定因编程语言而异。例如，使用 Java 编程语言时，totalDistance 将是此变量名称的标准选择。
3. 除了在某些数学上下文之外，避免使用单字母变量名和非标准首字母缩略词/缩写词。例如，我们可能使用 d1，d2 代替 distance1，distance2 因为 d 是我们在上面的公式中用于距离的变量。但是，我们不应该使用 td 代替 total_distance，因为新人不会立即理解 td 代表什么。

基于值的 C 语言内存模型 随着我们的程序变得越来越大，有一种方法来跟踪所使用的变量和数据是很有用的。内存模型是程序中变量和数据的结构化表示。这里的“内存”是指计算机实际用来存储数据的内存。在接下来的几周里，我们将使用基于值的 C 语言内存模型，它简单地使用一个表来表示每个变量及其值之间的关联。例如，上面示例的基于值的内存模型如下：

变量	值
distance1	2.236068
distance2	10.0
total_distance	12.236068

第 2 讲 函数

内容提要

- ❑ 标准库函数
- ❑ 函数作用域
- ❑ 局部变量

2.1 标准库函数

在上一章中，我们通过研究三个主要成分开始了 C 语言编程的学习：常量、运算符和变量。我们可以使用这些形式的 C 语言代码来表达复杂的计算，但随着我们想要执行的任务变得越来越复杂，我们需要编写的代码也变得越来越复杂。在本章中，我们将学习如何使用 C 语言中的函数将代码组织成有用的部分，这些部分可以独立编写和更新，并在我们的程序中一次又一次地重用。

复习：数学中的函数 在研究 Python 中的函数之前，我们首先回顾一下一年级 CS 暑期准备课程中与函数相关的一些数学定义。

让和被设置。一个功能是元素的映射到元素。称为函数的定义域，并且称为函数的余域。

函数可以有多个输入。套装用和，A -二元函数是一个函数，它需要参数，其中每个之间和，这 - 第一个参数必须是一个元素，以及哪里返回一个元素。我们有通用的英语术语来表示较小的值：一元、二元和三元函数分别采用一个、两个和三个输入。例如，函数定义为是一元函数，并且函数定义为

是一个三元函数。

我们可以对特定输入（其域的元素）调用数学函数，并通过将输入值代入函数定义并对表达式求值来计算相应的输出值。例如，使用函数和前面定义的，

和

Python 的内置函数我们已经看到 Python 有许多运算符，例如 + 和 in，可用于各种数据类型。这些运算符使用特殊符号表示数学函数（例如，通过 + 符号进行加法）。但由于这些运算符是在两个表达式之间编写的，因此它们仅限于表示二元函数。因此，Python 当然必须有一种超越我们目前研究过的运算符来表示函数的方法。

现在，我们将看到一些 Python 的内置函数，这些函数在 Python 程序中的任何位置都自动可用。例如，Python 有一个名为的内置函数 abs，它接受单个数字输入并返回其绝对值。但仅仅知道这个函数的存在还不够——我们如何实际使用它呢?

使用函数对给定输入进行操作的 Python 表达式称为函数调用，其语法与数学中的语法相同：

<function>(<argument>, <argument>, ...) 以下是使用的函数调用表达式的两个示例 abs：

»> abs(-10) Returns the absolute value of -10. 10 »> abs(100) 100 函数调用是编程的核心，并且附带了一些我们现在将介绍并在明年使用的新术语。

在函数调用表达式中，输入表达式称为函数调用的参数。例如，在表达式中 abs(-10)，我们说 “-10 是函数调用的参数”。当我们计算函数调用时，我们说参数被传递给函数。例如，在表达式中 abs(-10)，我们说 “-10 被传递给 abs”。当函数调用产生其输出值时，我们称函数调用返回该值，并将该值称为函数调用表达式的返回值。例如，我们说 “的返回值是 abs(-10)” 10。四舍五入数字这是数值函数的第二个示例 round。这比更复杂一点 abs，可以通过两种不同的方式使用：

给定单个参数 number x，round(x) 返回 int 四舍五入 x 到最接近的整数的值。

»> round(3.3) 3 »> round(-1.678) -2 给定参数 numberx 和非负 int d，round(x, d) 返回四舍五入到小数位 float 的值。xd

»> round(3.456, 2) 3.46 »> round(3.456, 0) This still returns a float 3.0 不仅仅是数字！到目前为止，在您的数学学习中，您主要研究了一元数值函数，即仅接受一个数值参数并返回另一个数字的函数。示例包括 sin 和 log

函数。然而，在编程中，使用对多种数据类型和大量参数进行操作的函数是很常见的。以下是一些内置 Python 函数的示例，这些函数不仅仅采用单个数字参数：

该 len 函数采用字符串或集合数据类型（例如，set、list）并返回其输入的大小。对于字符串来说，它的大小就是它包含的字符数；对于集合或列表，其大小是其元素的数量；对于字典来说，它的大小就是它的键值对的数量。

»> len(10, 20, 30) 3 »> len(") 0 »> len(['a', 'b', 'c', 'd', 'e']) 5 »> len('David': 100, 'Mario': 0) 2 该 sum 函数接受数字的集合（例如，aset 或其 list 元素都是数字）并返回数字的总和。

»> sum(10, 20, 30) 60 »> sum([-4.5, -10, 2, 0]) -12.5 »> sum([]) The sum of an empty collection is 0 0 该 sorted 函数接受一个集合并返回一个 list 包含与输入集合相同的元素的集合，并按升序排序。

»> sorted([10, 3, 20, -4]) [-4, 3, 10, 20] »> sorted(10, 3, 20, -4) Works with sets, too! [-4, 3, 10, 20] 该 max 函数有点特殊，因为它有两种使用方式。当使用两个或多个输入调用它时，这些输入必须是数字，在这种情况下 max 返回最大的一个。

»> max(2, 3) 3 »> max(3, -2, 10, 0, 1, 7) 10 但 max 也可以仅使用单个参数（非空数字集合）进行调用。在本例中，max 返回集合中的最大数字。

»> max(2, 3) 3 »> max([3, -2, 10, 0, 1, 7]) 10 该 min 函数与该函数类似 max ，只是它返回输入中的最小值。

»> min(2, 3) 2 »> max([3, -2, 10, 0, 1, 7]) -2 功能 type$typePythonclassPython$

»> type(3) <class 'int'> »> type(3.0) <class 'float'> »> type(True) <class 'bool'> »> type('David') <class 'str'> »> type(1, 2, 3) <class 'set'> »> type([1, 2, 3]) <class 'list'> »> type('a': 1, 'b': 2) <class 'dict'> 如果您不确定特定值的数据类型，您可以随时调用 type 它进行检查！

功能 help：我们将在本节中介绍的最后一个特殊的内置 Python 函数是 help，它采用单个参数并显示该参数的帮助文档。help 最常用于查找有关其他函数的更多信息：如果我们调用 help 一个函数，Python 解释器将显示有关如何使用该函数的信息。您会发现 Python 函数和数据类型的文档包含您还不熟悉的术语或概念。这是完全正常的！能够阅读并理解编程语言文档是计算机科学家的一项基本技能，并且您将在全年中获得经验。如果您在 Python help 文档中遇到不太理解的内容，请向我们询问！

»> help(abs) Help on built-in function abs in module builtins:

abs(x, /) Return the absolute value of the argument. 也可以调用 help 单独的值，例如 3 或 1, 2, 3；这样做将显示该值的数据类型的文档，例如 int 或 set。但是，我们不建议初学者这样做，因为显示的文档数量可能有点让人难以承受！相反，我们建议 help 至少在刚开始时用于特定功能。

关于嵌套函数调用的注意事项就像其他 Python 表达式一样，您可以在彼此之间编写函数调用，或者将它们与其他类型的表达式（例如算术表达式）混合。

»> max(abs(-100), 15, 3 * 20) 100 »> sorted(10, 2, 3) + sorted([-1, -2, -3]) [2, 3, 10, -3, -2, -1] 然而，正如我们之前在深度嵌套算术表达式中看到的那样，过多的嵌套会使 Python 表达式难以阅读和理解。因此，使用变量将一系列复杂的函数调用分解为中间步骤是一个很好的做法：

»> value1 = abs(-100) »> value2 = 15 »> value3 = 3 * 20 »» max(value1, value2, value3) 100 参考 CSC108 视频：功能（第 1 部分、第 2 部分、第 3 部分）附录 A.1 Python 内置函数参考

printf 函数

scanf 函数

2.2 局部变量和函数作用域

2.3 实际参数

2.4 函数测试

2.5 函数规范和正确性证明

2.6 例子：素数

第 3 讲 计算机科学中的形式逻辑

内容提要

- 命题逻辑
- 谓词逻辑
- 化简

3.1 命题逻辑

3.2 谓词逻辑

3.3 If 语句

3.4 化简 if 语句

第 4 讲 处理复杂数据：数组

4.1 重复执行：For 循环

4.2 嵌套 for 循环

4.3 while 循环

4.4 break 和 continue 语句

4.5 例子：约瑟夫问题

第 5 讲 归纳与递归

5.1 归纳证明

5.2 递归定义的函数

5.3 例子：汉诺塔

第 6 讲 分析算法运行时间

6.1 运行时间分析简介

6.2 比较渐近函数增长：Big-O 表示法

6.3 大 O、欧米茄、西塔：O, Ω, Θ

6.4 渐近增长和极限

6.5 分析算法运行时间

6.6 分析 While 循环

6.7 分析标准库中的数据类型操作

6.8 最坏情况运行时间分析

6.9 函数测试四：效率

第 7 讲 指针

第 8 讲 结构体：抽象和软件设计

第 9 讲 链表

9.1 链表简介

9.2 遍历链表

9.3 修改链表

9.4 链表的运行时间分析

第 10 讲 树

第 11 讲 案例研究：抽象语法树

11.1 抽象语法树简介

11.2 变量和环境

11.3 从表达式到语句

11.4 实践中的抽象语法树

第 12 讲 排序