

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO BÀI TẬP LỚN**

**ĐỀ TÀI: CHƯƠNG TRÌNH ORDER CÁC MÓN ĂN TRONG NHÀ HÀNG**

Giảng viên hướng dẫn: Th.S TRẦN THỊ DUNG

Người thực hiện: ĐẶNG VĂN NGHIÊM

Lớp: CÔNG NGHỆ THÔNG TIN

Khoá: 63

TP. Hồ Chí Minh, tháng 04 năm 2023



**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**

Logo

Description automatically generated

**BÁO CÁO BÀI TẬP LỚN**

**ĐỀ TÀI: CHƯƠNG TRÌNH ORDER CÁC MÓN ĂN TRONG NHÀ HÀNG**

Giảng viên hướng dẫn: Th.S TRẦN THỊ DUNG

Nhóm sinh viên thực hiện: NGUYỄN TRUNG TÍN

PHẠM THÀNH NHÂN

ĐẶNG VĂN NGHIÊM

LÊ HOÀNG PHÚC

Lớp: CÔNG NGHỆ THÔNG TIN

Khoá: 63

TP. Hồ Chí Minh, tháng 04 năm 2023

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH**

**CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM**

Độc lập – Tự do – Hạnh phúc

## BÀI TẬP LỚN

BỘ MÔN: **CÔNG NGHỆ THÔNG TIN**

\*\*\*

**Mã sinh viên:** 6351071048 **Họ tên SV:** Đặng Văn Nghiêm

**Khóa:** 63 **Lớp:** Công Nghệ Thông Tin

**1. Tên đề tài bài tập lớn:**

***CHƯƠNG TRÌNH ORDER CÁC MÓN ĂN TRONG NHÀ HÀNG***

**2. Mục đích, yêu cầu:**

**a. Mục đích:**

* Xây dựng chương trình order các món ăn trong nhà hàng
* Xây dựng các hàm hỗ trợ thực hiện một số chức năng nhập, hiển thị, tìm kiếm, sắp xếp, cập nhật, xóa, ghi menu vào file quản lí sách cũng như khách hàng order món ăn trong nhà hàng.

**b. Yêu cầu:**

* Tìm hiểu cách sử dụng các cấu trúc struct, danh sách liên kết để tối ưu hoá chương trình.

**3. Nội dung và phạm vi đề tài:**

**a. Nội dung đề tài:**

* Nêu lí do cần sử dụng đến chương trình order các món ăn trong nhà hàng.
* Nghiên cứu và triển khai việc viết các hàm.
* Kiểm thử chương trình.

**b. Phạm vi đề tài:**

**4. Công cụ và ngôn ngữ lập trình:**

**a. Công cụ:** Dev C++, Visual Studio Code.

**b. Ngôn ngữ lập trình:** C

**5. Các kết quả chính dự kiến sẽ đạt được và ứng dụng:**

Chương trình order các món ăn trong nhà hàng sẽ giúp khách hàng thực hiện được một số chức năng sau:

* Phần 1: Nhập thông tin của khách hàng
* Phần 2: Xuất thông tin khách hàng

- Phần 3: Cho phép khách hàng xem menu các món nướng

* Đọc menu các món nướng từ file.
* Khách hàng có thể chọn món sau khi xem menu

- Phần 4: Cho phép khách hàng xem menu các món lẫu

* Đọc menu các món lẫu từ file
* Khách hàng có thể chọn món sau khi xem menu

- Phần 5: Cho phép khách hàng xem menu các món nước

* Đọc menu các món nước từ file
* Khách hàng có thể order các món sau khi xem menu

- Phần 6: Mục gợi ý cho khách hàng

* Khách hàng có thể xem các mục gợi ý cho từng món ăn trong menu món lẫu, nướng như thành phần dinh dưỡng, từ đó chọn món ăn phù hợp.

- Phần 7: Khách hàng xác nhận lại món ăn của mình chọn

- Phần 8: Cho khách hàng xem bill dự kiến

- Phần 9: Đánh giá của khách hàng

**6. Giáo viên và cán bộ hướng dẫn:**

Họ tên: TRẦN THỊ DUNG.

Đơn vị công tác: Bộ môn Công Nghệ Thông Tin – Trường Đại học Giao thông Vận tải phân hiệu tại TP HCM

Điện thoại: Email:

|  |  |
| --- | --- |
| **Ngày 15 tháng 04 năm 2023 BM Công Nghệ Thông Tin** | **Đã giao nhiệm vụ TKTN Giáo viên hướng dẫn** |
|  | **ThS. Trần Thị Dung** |

Đã nhận nhiệm vụ Bài Tập Lớn

Sinh viên: ĐẶNG VĂN NGHIÊM Kí Tên

Điện thoại: 0379309749 Email: vannghiemdang2004@gmail.com

## LỜI CẢM ƠN

Qua thời gian học tập và rèn luyện tại trường Trường Đại học Giao thông Vận tải phân hiệu tại TP HCM, đến nay em đã được trang bị nhữung kĩ năng, kiến thức cơ bản để có thể hoàn thành được bài tập lớn do giảng viên giao.

Cảm ơn tập thể các thầy cô giáo Bộ môn Công Nghệ Thông Tin và các thầy cô thỉnh giảng đã giảng dạy, luôn quan tâm và không ngần ngại dành thời gian để chỉ bày và giải đáp những thắc mắc của chúng em trong những tiết học và cả những lúc ngoài giờ.

Và cảm ơn thạc sĩ Trần Thị Dung đã luôn quan tâm nhiệt tình hướng dẫn, giúp đỡ chúng em trong quá trình triển khai và thực hiện bài tập lớn. Cô cũng luôn nhắc nhở, giúp đỡ mỗi khi chúng em gặp khó khăn, nhờ vậy mà em đã hoàn thành bài tập lớn của nhóm mình đúng thời hạn được giao. Nếu không có sự hướng dẫn nhiệt tình của cô thì có lẽ chúng em đã khó có thể thực hiện được bài tập đúng theo mong muốn của mình.

Nhóm chúng em đã bỏ ra nhiều thời gian để tìm hiểu và trang bị thêm kiến thức nhằm phục vụ cho việc thực hiện ý tưởng, nhưng chắc chắn rằng nhóm chúng em sẽ không thể tránh khỏi những sai sót không đáng có vì kiến thức còn hạn chế. Chúng em hi vọng rằng sẽ nhận được những lời góp ý quý báu của cô để chúng em có thể hoàn thiện ý tưởng của nhóm một cách tốt nhất có thể.

#### TP. Hồ Chí Minh, ngày 15 tháng 04 năm 2023

**Sinh viên thực hiện**

**Đặng Văn Nghiêm**

## NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

...........................................................................................................................................

...........................................................................................................................................

...........................................................................................................................................

...........................................................................................................................................

...........................................................................................................................................

...........................................................................................................................................

...........................................................................................................................................

...........................................................................................................................................

...........................................................................................................................................

...........................................................................................................................................

........................................................................................................................................…

#### **Tp. Hồ Chí Minh, ngày 15 tháng 04 năm 2023**

**Giáo viên hướng dẫn**

**ThS. Trần Thị Dung**

**Mục lục**

[BÀI TẬP LỚN 2](#_Toc15953)

[LỜI CẢM ƠN 5](#_Toc4314)

[NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN 6](#_Toc29223)

[**PHẦN 1: DỊCH SÁCH 9**](#_Toc18806)

**CHƯƠNG 5………………………………………………………………………………………………………………………………………………………………..9**

[**5.1 Giới thiệu 9**](#_Toc23)

[**5.2 Các mô-đun chương trình trong C 9**](#_Toc23544)

[**5.3 Hàm thư viện toán học 10**](#_Toc26808)

[**5.4 Chức năng 11**](#_Toc16959)

[**5.5 Định nghĩa hàm 12**](#_Toc21383)

[**5.6 Nguyên mẫu chức năng: 16**](#_Toc11906)

[**5.7 Gọi hàm Stack và Stack Frame 18**](#_Toc30669)

[**5.8 Tiêu đề 21**](#_Toc14637)

[**5.9 Truyền đối số theo giá trị và theo tham chiếu 21**](#_Toc22904)

[**5.10 Tạo số ngẫu nhiên 21**](#_Toc22212)

[**5.11 Ví dụ: Trò chơi may rủi 25**](#_Toc25099)

[**5.12 Các lớp lưu trữ 26**](#_Toc7582)

[**5.13 Quy tắc phạm vi 28**](#_Toc24026)

[**5.14 Đệ quy 29**](#_Toc25273)

[**5.15 Ví dụ Sử dụng Đệ quy: Dãy Fibonacci 32**](#_Toc26648)

[**5.16 Đệ quy so với Lặp lại 34**](#_Toc11768)

[**5.17 Lập trình C an toàn 36**](#_Toc25745)

[**i) Hàm được sử dụng để tạo số ngẫu nhiên. 42**](#_Toc18825)

[**a) Hàm được mô tả trong Bài tập 5.4(a). 43**](#_Toc23560)

[**b) Hàm được mô tả trong Bài tập 5.4(b). 43**](#_Toc3399)

[**c) Hàm mô tả trong Bài tập 5.4(c). 43**](#_Toc30852)

[**d) Hàm được mô tả trong Bài tập 5.4(d). 43**](#_Toc20048)

[**b) int smallest( int x, int y, int z ) 44**](#_Toc2328)

[**b) int smallest( int x, int y, int z ); 44**](#_Toc23592)

[**c) void instructions( void ); 44**](#_Toc28912)

[**d) float intToFloat( int number ); 44**](#_Toc33)

[**a) x = fabs( 7,5 ); 45**](#_Toc17311)

[**b) x = sàn( 7,5 ); 45**](#_Toc22757)

[**c) x = fabs( 0,0 ); 45**](#_Toc8451)

[**d) x = trần( 0,0 ); 45**](#_Toc31563)

[**e) x = fabs( -6,4 ); 45**](#_Toc13633)

[**f) x = trần( -6,4 ); 45**](#_Toc19169)

[**g) x = trần( -fabs( -8 + sàn( -5,5 ) ) ); 45**](#_Toc9638)

[**Chương 8: KÍ TỰ VÀ CHUỖI 56**](#_Toc20442)

[**CHƯƠNG 7: CON TRỎ 62**](#_Toc14694)

[**CHƯƠNG 12 CẤU TRÚC DỮ LIỆU 70**](#_Toc15671)

[**Mục tiêu 70**](#_Toc17818)

[**12.1 Giới thiệu 71**](#_Toc12641)

[**12.2 Cấu trúc tự tham chiếu 71**](#_Toc24042)

[**12.3 Cấp phát bộ nhớ động 71**](#_Toc19614)

[**12.4 Danh sách liên kết 71**](#_Toc6086)

[**12.6 Hàng đợi 71**](#_Toc21495)

[**12.7 Cây 71**](#_Toc23575)

[12.7.1 Chèn nút chức năng 71](#_Toc3216)

[12.7.2 Traversal: Hàm theo thứ tự, 71](#_Toc25551)

[12.7.3 Loại bỏ trùng lặp 71](#_Toc6784)

[12.7.4 Tìm kiếm cây nhị phân 71](#_Toc7632)

[12.7.5 Các phép toán cây nhị phân khác 71](#_Toc18347)

[**12.8 Lập trình C an toàn 71**](#_Toc28906)

[**12.1 Giới thiệu 71**](#_Toc31278)

[**12.2 Cấu trúc tự tham chiếu 73**](#_Toc8974)

[**12.3 Cấp phát bộ nhớ động 73**](#_Toc8023)

[**12.4 Danh sách liên kết 75**](#_Toc14062)

[12.4.1 Chức năng chèn 83](#_Toc14151)

[12.4.2 Chức năng xóa 84](#_Toc23441)

[12.4.3 Hàm printList 85](#_Toc20137)

[**12.5 ngăn xếp 85**](#_Toc9646)

[12.5.1 Đẩy chức năng 91](#_Toc24360)

[12.5.2 Bật chức năng 92](#_Toc18639)

[12.5.3 Các ứng dụng của ngăn xếp 93](#_Toc5255)

[**12.6 Hàng đợi 94**](#_Toc30341)

[12.6.1 Chức năng xếp hàng 100](#_Toc9098)

[12.6.2 Hàm dequeue 101](#_Toc31793)

[**12.7 Cây 101**](#_Toc2317)

[**12.8 Lập trình C an toàn 109**](#_Toc15410)

[**PHẦN 2: CHƯƠNG TRÌNH 110**](#_Toc31492)

[1. Mô tả bài toán 110](#_Toc17790)

[2. Giao diện chương trình 111](#_Toc25136)

**PHẦN 1: DỊCH SÁCH**

**CHƯƠNG 5 : CHỨC NĂNG**

**Mục tiêu**

- Xây dựng các chương trình theo mô đun từ các phần nhỏ được gọi là hàm.

- Sử dụng các hàm toán học phổ biến trong trong thư viện chuẩn C.

- Tạo chức năng mới.

- Sử dụng các cơ chế truyền thông tin giữa các chức năng.

- Tìm hiểu cách cơ chế gọi/trả lại chức năng được hỗ trợ bởi ngăn xếp lệnh gọi hàm và khung ngăn xếp.

- Sử dụng kĩ thuật mô phỏng dựa trên số ngẫu nhiên thế hệ.

- Viết và sử dụng các chức năng gọi chính nó.

**5.1 Giới thiệu**

Hầu hết các chương trình máy tính giải quyết các vấn đề trong thế giới thực đều lớn hơn nhiều so với các chương trình chuyên nghiệp.

gram được trình bày trong một số chương đầu tiên. Kinh nghiệm đã chỉ ra rằng cách tốt nhất để loại bỏ

phát triển và duy trì một chương trình lớn là xây dựng nó từ các phần hoặc mô-đun nhỏ hơn, mỗi phần

trong đó dễ quản lí hơn chương trình gốc. Kỹ thuật này được gọi là chia và chinh phục. Chương này mô tả một số tính năng chính của ngôn ngữ C tạo điều kiện thuận lợi cho thiết kế, thực hiện, vận hành và bảo trì các chương trình lớn.

**5.2 Các mô-đun chương trình trong C**

Các mô-đun trong C được gọi là các hàm. Các chương trình C thường được viết bằng cách kết hợp các chương trình mới các hàm bạn viết bằng các hàm đóng gói sẵn có sẵn trong thư viện chuẩn C. Chúng tôi không thảo luận về cả hai loại chức năng trong chương này. Thư viện chuẩn C cung cấp một bộ sưu tập phong phú các chức năng để thực hiện các phép tính toán học thông thường, thao tác chuỗi, thao tác ký tự, nhập/xuất và nhiều thao tác hữu ích khác. Điều này làm cho bạn công việc dễ dàng hơn vì các chức năng này cung cấp nhiều khả năng bạn cần.

Thực hành lập trình tốt 5.1

Làm quen với bộ sưu tập hàm phong phú trong thư viện chuẩn C. Quan sát kỹ thuật phần mềm 5.1

Tránh phát minh lại bánh xe. Khi có thể, hãy sử dụng các hàm thư viện chuẩn C thay vì viết chức năng mới. Điều này có thể làm giảm thời gian phát triển chương trình.

Mẹo về tính di động 5.1

Sử dụng các chức năng trong thư viện chuẩn C giúp làm cho các chương trình dễ mang theo hơn. Ngôn ngữ C và thư viện chuẩn đều được chỉ định bởi tiêu chuẩn C và cả hai đều được cung cấp hệ thống C tiêu chuẩn (ngoại trừ một số thư viện được chỉ định là tùy chọn). Các chức năng printf, scanf và pow mà chúng tôi đã sử dụng trong các chương trước là các hàm thư viện chuẩn.

Bạn có thể viết các hàm để xác định các tác vụ cụ thể có thể được sử dụng tại nhiều điểm trong một

chương trình. Đôi khi chúng được gọi là các hàm do lập trình viên định nghĩa. Thực tế các câu lệnh xác định hàm chỉ được viết một lần và các câu lệnh bị ẩn từ các chức năng khác.

Các hàm được gọi bởi một lệnh gọi hàm, trong đó chỉ định tên hàm và quy trình

vides thông tin (dưới dạng đối số) mà chức năng cần để thực hiện nhiệm vụ được chỉ định của nó. Một

tương tự phổ biến cho điều này là hình thức quản lý phân cấp. Một ông chủ (chức năng gọi- tion hoặc người gọi) yêu cầu một worker (hàm được gọi) thực hiện một tác vụ và báo cáo lại khi nhiệm vụ đã hoàn thành (Hình 5.1). Ví dụ, một chức năng cần hiển thị thông tin trên màn hình gọi hàm worker printf để thực hiện tác vụ đó, sau đó printf hiển thị thông tin kết nối và báo cáo lại—hoặc trả về—cho chức năng gọi khi nhiệm vụ của nó hoàn thành.

Chức năng ông chủ không biết chức năng công nhân thực hiện các nhiệm vụ được chỉ định như thế nào. Các worker có thể gọi các hàm worker khác và ông chủ sẽ không biết về điều này. Chúng ta sẽ sớm thấy làm thế nào việc “che giấu” các chi tiết triển khai này thúc đẩy công nghệ phần mềm tốt. Hình 5.1 hiển thị chức năng của ông chủ giao tiếp với một số chức năng của công nhân theo thứ bậc thái độ. Lưu ý rằng Worker1 đóng vai trò là chức năng của ông chủ đối với worker4 và worker5. Các mối quan hệ giữa các chức năng có thể khác với cấu trúc phân cấp thể hiện trong hình này.

**5.3 Hàm thư viện toán học**

Các hàm thư viện toán học cho phép bạn thực hiện một số phép tính toán học phổ biến.

Chúng tôi sử dụng một số trong số chúng ở đây để giới thiệu khái niệm hàm. Ở phần sau của cuốn sách, chúng ta sẽ thảo luận về nhiều chức năng khác trong thư viện chuẩn C. Các chức năng thường được sử dụng trong một chương trình bằng cách viết tên của chức năng theo sau là dấu ngoặc đơn bên trái, theo sau là đối số (hoặc danh sách được phân tách bằng dấu phẩy của đối số) của hàm theo sau là dấu ngoặc đơn bên phải. Ví dụ, để tính toán và in căn bậc hai của 900,0 bạn có thể viết

printf( "%.2f", sqrt( 900.0 ) );

Khi câu lệnh này thực thi, hàm thư viện toán học sqrt được gọi để tính bình phương gốc của số chứa trong ngoặc đơn (900.0). Số 900.0 là đối số của hàm sqrt. Câu lệnh trước sẽ in ra 30.00. Hàm sqrt lấy đối số kiểu double và trả về kết quả kiểu double. Tất cả các chức năng trong toán học thư viện trả về giá trị dấu phẩy động trả về kiểu dữ liệu gấp đôi. Lưu ý rằng giá trị kép ues, như giá trị float, có thể được xuất ra bằng cách sử dụng đặc tả chuyển đổi %f.

Mẹo ngăn ngừa lỗi 5.1

Bao gồm tiêu đề toán học bằng cách sử dụng chỉ thị tiền xử lý #include <math.h> khi

sử dụng các hàm trong thư viện toán học.

Các đối số của hàm có thể là hằng, biến hoặc biểu thức. Nếu c1 = 13,0, d = 3,0

và f = 4.0, thì mệnh đề tính toán và in căn bậc hai của 13,0 + 3,0 \* 4,0 = 25,0, cụ thể là 5,00.

printf( "%.2f", sqrt( c1 + d \* f ) );

Hình 5.2 tóm tắt một ví dụ nhỏ về các hàm trong thư viện toán học C. Trong hình, các biến x và y có kiểu double. Tiêu chuẩn C11 bổ sung thêm một loạt các floating-khả năng điểm và số phức.

**5.4 Chức năng**

Các chức năng cho phép bạn mô đun hóa một chương trình. Tất cả các biến được định nghĩa trong định nghĩa hàm là các biến cục bộ - chúng chỉ có thể được truy cập trong hàm mà chúng được định nghĩa.

Hầu hết các chức năng đều có một danh sách các tham số cung cấp phương tiện để truyền đạt thông tin

giao tiếp giữa các chức năng. Các tham số của hàm cũng là các biến cục bộ của hàm đó.

Quan sát kỹ thuật phần mềm 5.2

Trong các chương trình chứa nhiều chức năng, hàm main thường được thực hiện dưới dạng một nhóm các lệnh gọi đến các chức năng thực hiện phần lớn công việc của chương trình.

Có một số động lực để “chức năng hóa” một chương trình. Sự phân chia và con- cách tiếp cận làm quen cho việc phát triển chương trình dễ quản lí hơn. Một động lực khác là khả năng sử dụng lại phần mềm- sử dụng các chức năng hiện có làm khối xây dựng để tạo chương trình mới.

Khả năng sử dụng lại phần mềm là một yếu tố chính trong phong trào lập trình hướng đối tượng mà

bạn sẽ tìm hiểu thêm khi bạn học các ngôn ngữ bắt nguồn từ C, chẳng hạn như C++, Java C# (phát âm là “ C sharp”). Với cách đặt tên và định nghĩa hàm tốt, các chương trình có thể được tạo ra từ các chức năng được tiêu chuẩn hóa để hoàn thành các nhiệm vụ cụ thể, thay vì được xây dựng bằng cách sử dụng mã tùy chỉnh. Điều này được gọi là trừu tượng. Chúng tôi sử dụng trừu tượng mỗi thời gian chúng tôi sử dụng các chức năng thư viện tiêu chuẩn như printf, scanf và pow. Động lực thứ ba là để tránh lặp lại mã trong một chương trình. Mã đóng gói như một chức năng cho phép mã được thực hiện từ các vị trí khác trong một chương trình chỉ bằng cách gọi hàm.

Quan sát kỹ thuật phần mềm 5.3

Mỗi chức năng nên được giới hạn để thực hiện một nhiệm vụ duy nhất, được xác định rõ ràng và

tên chức năng nên thể hiện nhiệm vụ đó. Điều này tạo điều kiện cho sự trừu tượng hóa và thúc đẩy phần mềm khả năng tái sử dụng.

Quan sát kỹ thuật phần mềm 5.4

Nếu bạn không thể chọn một cái tên ngắn gọn thể hiện chức năng của nó, thì có thể là chức năng của bạn đang cố thực hiện quá nhiều tác vụ đa dạng. Nó thường là tốt nhất để phá vỡ một chức năng như vậy thành nhiều chức năng nhỏ hơn—điều này đôi khi được gọi là phân tách.

**5.5 Định nghĩa hàm**

Mỗi chương trình chúng tôi đã trình bày bao gồm một chức năng được gọi là chính được gọi là tiêu chuẩn chức năng thư viện để hoàn thành nhiệm vụ của mình. Bây giờ chúng ta xem xét cách viết các hàm tùy chỉnh.

Hãy xem xét một chương trình sử dụng một hàm bình phương để tính toán và in các bình phương của

số nguyên từ 1 đến 10 (Hình 5.3).

Hàm square được gọi hoặc gọi trong main trong câu lệnh printf (dòng 14) printf(" %d ", vuông( x ) ); //gọi hàm square nhận một bản sao giá trị của x trong tham số y (dòng 21). Sau đó vuông tính toán y \* y. Kết quả được trả lại cho chức năng printf trong main nơi square đã được gọi (dòng 14) và printf hiển thị kết quả. Quá trình này được lặp đi lặp lại 10 lần sử dụng câu lệnh for.

Định nghĩa của hàm square (dòng 21–24) cho thấy rằng square cần một số nguyên tham số y. Từ khóa int trước tên hàm (dòng 21) chỉ ra rằng square trả về một kết quả số nguyên. Câu lệnh return trong hình vuông chuyển giá trị của biểu thức y \* y (nghĩa là kết quả của phép tính) trở lại hàm gọi.

Line 5 int vuông( int y ); //nguyên mẫu hàm là một nguyên mẫu chức năng. int trong ngoặc đơn thông báo cho trình biên dịch rằng hình vuông mong đợi là một nguyên mẫu chức năng. int trong ngoặc đơn thông báo cho trình biên dịch rằng hình vuông mong đợi để nhận một giá trị số nguyên từ người gọi. Int ở bên trái của hình vuông tên hàm thông báo cho trình biên dịch rằng hình vuông trả về kết quả số nguyên cho người gọi. Trình biên dịch đề cập đến nguyên mẫu hàm để kiểm tra xem có bất kỳ lời gọi nào đến hình vuông (dòng 14) chứa đúng kiểu trả về, số lượng đối số chính xác và kiểu đối số chính xác, và rằng đối số guments theo đúng thứ tự. Nguyên mẫu chức năng được thảo luận chi tiết trong Phần 5.6.

Định dạng của một định nghĩa hàm là tên hàm kiểu giá trị trả về ( danh sách tham số )

{

các định nghĩa

các câu lệnh

}

Tên hàm là bất kỳ mã định danh hợp lệ nào. Kiểu giá trị trả về là kiểu dữ liệu của sult trả lại cho người gọi. Khoảng trống kiểu giá trị trả về chỉ ra rằng một hàm không sult trả lại cho người gọi. Khoảng trống kiểu giá trị trả về chỉ ra rằng một hàm không trả về một giá trị. Cùng với nhau, kiểu giá trị trả về, tên hàm và danh sách tham số là một số - lần được gọi là tiêu đề chức năng.

Mẹo ngăn ngừa lỗi 5.2

Kiểm tra xem các chức năng của bạn được cho là trả về giá trị có làm như vậy không. Kiểm tra xem chức năng của bạn có những thứ không được phép trả về giá trị thì không.

Danh sách tham số là một danh sách được phân tách bằng dấu phẩy chỉ định các tham số nhận được bởi

chức năng khi nó được gọi. Nếu một chức năng không nhận được bất kỳ giá trị nào, danh sách tham số là

khoảng trống. Một loại phải được liệt kê rõ ràng cho từng tham số.

Lỗi lập trình thường gặp 5.1

Chỉ định các tham số chức năng cùng loại như double x, y thay vì double x, dou- ble y dẫn đến lỗi biên dịch.

Lỗi lập trình thường gặp 5.2

Đặt dấu chấm phẩy sau dấu ngoặc đơn bên phải kèm theo danh sách tham số của hàm định nghĩa là một lỗi cú pháp.

Lỗi lập trình thường gặp 5.3

Việc xác định lại tham số dưới dạng biến cục bộ trong hàm là lỗi biên dịch.

Thực hành lập trình tốt 5.2

Mặc dù làm như vậy không sai, nhưng đừng sử dụng cùng tên cho các đối số của hàm và các tham số tương ứng trong định nghĩa hàm. Điều này giúp tránh sự mơ hồ.

Các định nghĩa và câu lệnh trong dấu ngoặc nhọn tạo thành thân hàm, cũng là được gọi là một khối. Các biến có thể được khai báo trong bất kỳ khối nào và các khối có thể được lồng vào nhau.

Lỗi lập trình thường gặp 5.4

Định nghĩa một hàm bên trong một hàm khác là một lỗi cú pháp.

Thực hành lập trình tốt 5.3

Việc chọn tên hàm có ý nghĩa và tên tham số có ý nghĩa làm cho chương trình dễ đọc hơn và giúp tránh sử dụng quá nhiều bình luận.

Quan sát kỹ thuật phần mềm 5.5

Các chức năng nhỏ thúc đẩy khả năng tái sử dụng phần mềm.

Quan sát kỹ thuật phần mềm 5.6

Các chương trình nên được viết dưới dạng tập hợp các hàm nhỏ. Điều này làm cho các chương trình dễ dàng hơn để viết, gỡ lỗi, bảo trì và sửa đổi.

Quan sát kỹ thuật phần mềm 5.7

Một chức năng yêu cầu một số lượng lớn tham số có thể đang thực hiện quá nhiều tác vụ. Xem xét việc chia chức năng thành các chức năng nhỏ hơn để thực hiện các nhiệm vụ riêng biệt. Các tiêu đề chức năng phải vừa trên một dòng nếu có thể.

Quan sát kỹ thuật phần mềm 5.8

Nguyên mẫu hàm, tiêu đề hàm và lệnh gọi hàm đều phải thống nhất về số lượng, loại và thứ tự của các đối số và tham số cũng như loại giá trị trả về. Có ba cách để trả lại quyền kiểm soát từ một chức năng được gọi đến điểm mà tại đó một chức năng đã được gọi. Nếu hàm không trả về kết quả, điều khiển được trả về đơn giản khi đạt đến dấu ngoặc phải kết thúc hàm hoặc bằng cách thực hiện câu lệnh

return;

Nếu hàm trả về kết quả, câu lệnh return expression; trả về giá trị của biểu thức cho người gọi. main’s Return Type

Lưu ý rằng main có kiểu trả về int. Giá trị trả về của main được sử dụng để chỉ ra chương trình có thực thi chính xác hay không. Trong các phiên bản trước của C, chúng tôi đặt rõ ràng return 0; ở cuối main—0 cho biết chương trình đã chạy thành công. Tiêu chuẩn C chỉ ra chính đó hoàn toàn trả về 0 nếu bạn bỏ qua câu lệnh trước đó—như chúng ta đã làm một vấn đề xảy ra trong quá trình thực hiện chương trình của bạn. Để biết thông tin về cách báo cáo một lỗi chương trình, hãy xem tài liệu dành cho môi trường hệ điều hành cụ thể của bạn.

Ví dụ thứ hai của chúng tôi sử dụng một chức năng tối đa do lập trình viên xác định để xác định và tái

biến số lớn nhất trong ba số nguyên (Hình 5.4). Các số nguyên được nhập bằng scanf (dòng 15).

Tiếp theo, chúng được chuyển đến giá trị lớn nhất (dòng 19), xác định số nguyên lớn nhất. Giá trị này

được trả về chính bởi câu lệnh return ở mức tối đa (dòng 36). Giá trị trả về là sau đó được in trong câu lệnh printf (dòng 19).

1 // Hình 5.4: fig05\_04.c

2 // Tìm giá trị lớn nhất của ba số nguyên.

3 #include <stdio.h>

4

5 int maximum( int x, int y, int z ); //nguyên mẫu hàm

6

7 // hàm main bắt đầu thực hiện chương trình

8 int chính (khoảng trống)

9 {

Hình 5.4 | Tìm số lớn nhất của ba số nguyên. (Phần 1/2.)

10 số nguyên 1; // số nguyên đầu tiên do người dùng nhập vào

11 int số2; // số nguyên thứ hai do người dùng nhập vào

12 số int3; // số nguyên thứ ba do người dùng nhập vào

13

14 printf("%s", "Nhập ba số nguyên: " );

15 scanf( "%d%d%d", &số1, &số2, &số3 );

16

17 // số1, số2 và số3 là đối số

18 // đến lệnh gọi hàm tối đa

19 printf("Tối đa là: %d\n", );

20 } // kết thúc chính

**5.6 Nguyên mẫu chức năng:**

Nhìn sâu hơn một tính năng quan trọng của C là nguyên mẫu hàm. Tính năng này được vay mượn từ C++.

Trình biên dịch sử dụng các nguyên mẫu hàm để xác thực các lời gọi hàm. Các phiên bản đầu tiên của C đã làm không thực hiện loại kiểm tra này, vì vậy có thể gọi các chức năng không đúng cách mà không trình biên dịch phát hiện lỗi. Các cuộc gọi như vậy có thể dẫn đến lỗi thời gian thực hiện nghiêm trọng hoặc các lỗi không nghiêm trọng gây ra các vấn đề tinh vi, khó phát hiện. nguyên mẫu chức năng tương ứng khắc phục sự thiếu sót này.

Thực hành lập trình tốt 5.4

Bao gồm các nguyên mẫu chức năng cho tất cả các chức năng để tận dụng ca-kiểm tra kiểu của C khả năng. Sử dụng các chỉ thị tiền xử lý #include để có được các nguyên mẫu chức năng cho stan- chức năng thư viện dard từ các tiêu đề cho các thư viện thích hợp hoặc để có được các tiêu đề chứa các nguyên mẫu hàm cho các hàm do bạn và/hoặc các thành viên trong nhóm của bạn phát triển.

Nguyên mẫu hàm cho cực đại trong Hình 5.4 (dòng 5) là int maximum( int x, int y, int z ); //nguyên mẫu hàm

Nó nói rằng tối đa nhận ba đối số kiểu int và trả về kết quả kiểu int. Lưu ý rằng nguyên mẫu của hàm giống với dòng đầu tiên của định nghĩa cực đại.

Thực hành lập trình tốt 5.5

Tên tham số đôi khi được bao gồm trong nguyên mẫu hàm (tùy chọn của chúng tôi) cho doc- mục đích thuyết minh. Trình biên dịch bỏ qua những tên này.

Lỗi lập trình thường gặp 5.5

Quên dấu chấm phẩy ở cuối nguyên mẫu hàm là một lỗi cú pháp.

Lỗi biên dịch

Một lệnh gọi hàm không khớp với nguyên mẫu hàm là một lỗi biên dịch. Một lỗi cũng được tạo ra nếu nguyên mẫu hàm và định nghĩa hàm không đồng nhất.

Ví dụ, trong Hình 5.4, nếu nguyên mẫu hàm đã được viết: void maximum( int x, int y, int z ); trình biên dịch sẽ tạo ra lỗi vì kiểu trả về void trong nguyên mẫu hàm loại sẽ khác với loại trả về int trong tiêu đề hàm. Ép buộc đối số và “Quy tắc chuyển đổi số học thông thường”. Một tính năng quan trọng khác của các nguyên mẫu hàm là sự ép buộc của các đối số, tức là, buộc các đối số vào loại thích hợp. Ví dụ, hàm thư viện toán học sqrt có thể được gọi với một đối số nguyên mặc dù nguyên mẫu hàm trong <math.h> chỉ định một tham số kép và chức năng sẽ vẫn hoạt động bình thường. tuyên bố printf( "%.3f\n", sqrt( 4 ) ); đánh giá chính xác sqrt(4) và in ra giá trị 2.000. Nguyên mẫu hàm gây ra trình biên dịch để chuyển đổi một bản sao của giá trị số nguyên 4 thành giá trị kép 4.0 trước khi bản sao được chuyển đến sqrt. Nói chung, các giá trị đối số không tương ứng chính xác với tham số các loại trong nguyên mẫu hàm được chuyển đổi thành loại thích hợp trước khi hàm được gọi. Những chuyển đổi này có thể dẫn đến kết quả không chính xác nếu quy tắc chuyển đổi số học thông thường của C là đa không theo doi. Các quy tắc này chỉ định cách các giá trị có thể được chuyển đổi sang các loại khác mà không làm mất nhập dữ liệu. Trong ví dụ sqrt của chúng tôi ở trên, một int được tự động chuyển đổi thành double với- thay đổi giá trị của nó (vì gấp đôi có thể đại diện cho một phạm vi giá trị lớn hơn nhiều so với int). Tuy nhiên, double được chuyển đổi thành int sẽ cắt bớt phần phân số của double giá trị, do đó thay đổi giá trị ban đầu. Chuyển đổi kiểu số nguyên lớn thành số nguyên nhỏ các loại (ví dụ: dài đến ngắn) cũng có thể dẫn đến giá trị thay đổi.

Các quy tắc chuyển đổi số học thông thường sẽ tự động áp dụng cho các biểu thức chứa giá trị của hai loại dữ liệu (còn được gọi là biểu thức loại hỗn hợp) và được xử lý cho bạn bởi trình biên dịch. Trong một biểu thức kiểu hỗn hợp, trình biên dịch tạo một bản sao tạm thời của giá trị cần chuyển đổi sau đó chuyển đổi bản sao thành loại “cao nhất” trong biểu thức—giá trị ban đầu không thay đổi. Các quy tắc chuyển đổi số học thông thường đối với biểu thức kiểu hỗn hợp chứa ít nhất một giá trị dấu phẩy động là:

• Nếu một trong các giá trị là long double, giá trị còn lại sẽ được chuyển đổi thành long double.

• Nếu một trong các giá trị là gấp đôi, giá trị còn lại sẽ được chuyển đổi thành gấp đôi.

• Nếu một trong các giá trị là float, giá trị còn lại sẽ được chuyển thành float.

Nếu biểu thức kiểu hỗn hợp chỉ chứa các kiểu số nguyên, thì phép chuyển đổi số học thông thường- sions chỉ định một tập hợp các quy tắc khuyến mãi số nguyên. Trong hầu hết các trường hợp, các kiểu số nguyên thấp hơn.

Hình 5.5 được chuyển đổi sang loại cao hơn trong hình. Mục 6.3.1 của tài liệu chuẩn C- ument xác định chi tiết đầy đủ của toán hạng số học và cấu hình số học thông thường quy tắc phiên bản. Hình 5.5 liệt kê các kiểu dữ liệu dấu chấm động và số nguyên với mỗi kiểu thông số kỹ thuật chuyển đổi printf và scanf.

Việc chuyển đổi các giá trị thành các loại thấp hơn trong Hình 5.5 có thể dẫn đến các giá trị không chính xác, do đó, Piller thường đưa ra cảnh báo cho những trường hợp như vậy. Một giá trị chỉ có thể được chuyển đổi thành loại thấp hơn bằng cách gán rõ ràng giá trị cho một biến loại thấp hơn hoặc bằng cách sử dụng toán tử ép kiểu. Argu- ments trong một lệnh gọi hàm được chuyển đổi thành các loại tham số được chỉ định trong nguyên mẫu hàm gõ như thể các đối số được gán trực tiếp cho các biến thuộc các kiểu đó. Nếu hình vuông của chúng ta hàm sử dụng tham số int (Hình 5.3) được gọi với đối số dấu phẩy động, đối số được chuyển đổi thành int (loại thấp hơn) và hình vuông thường trả về giá trị không chính xác.

Ví dụ: square(4,5) trả về 16, không phải 20,25.

Lỗi lập trình thường gặp 5.6

Chuyển đổi từ loại dữ liệu cao hơn trong phân cấp khuyến mãi sang loại thấp hơn có thể thay đổi giá trị dữ liệu. Nhiều trình biên dịch đưa ra cảnh báo trong những trường hợp như vậy. Nếu không có nguyên mẫu chức năng cho một chức năng, trình biên dịch sẽ tạo chức năng riêng của nó nguyên mẫu bằng cách sử dụng lần xuất hiện đầu tiên của hàm—có thể là định nghĩa hàm hoặc gọi đến chức năng. Điều này thường dẫn đến cảnh báo hoặc lỗi, tùy thuộc vào trình biên dịch.

Mẹo ngăn ngừa lỗi 5.3

Luôn bao gồm các nguyên mẫu hàm cho các hàm bạn xác định hoặc sử dụng trong chương trình của mình để giúp ngăn ngừa lỗi biên dịch và cảnh báo.

Quan sát kỹ thuật phần mềm 5.9

Một nguyên mẫu hàm được đặt bên ngoài bất kỳ định nghĩa hàm nào áp dụng cho tất cả các lệnh gọi đến chức năng xuất hiện sau nguyên mẫu chức năng trong tệp. Một nguyên mẫu chức năng được đặt trong một chức năng chỉ áp dụng cho các cuộc gọi được thực hiện trong chức năng đó.

**5.7 Gọi hàm Stack và Stack Frame**

Để hiểu cách C thực hiện lời gọi hàm, trước tiên chúng ta cần xem xét một cấu trúc dữ liệu (nghĩa là tập hợp các mục dữ liệu liên quan) được gọi là ngăn xếp. Hãy nghĩ về một ngăn xếp tương tự như một đống bát đĩa. Khi một món ăn được đặt trên đống, nó thường được đặt ở trên cùng (gọi tắt là khi đẩy đĩa vào ngăn xếp). Tương tự như vậy, khi một món ăn được lấy ra khỏi đống, nó được gọi là cấu trúc dữ liệu nhập sau xuất trước (LIFO)—mục cuối cùng được đẩy (chèn) vào ngăn xếp là mục đầu tiên được bật (xóa) khỏi ngăn xếp.

Một cơ chế quan trọng để sinh viên khoa học máy tính hiểu được là chức năng ngăn xếp cuộc gọi (đôi khi được gọi là ngăn xếp thực thi chương trình). Cấu trúc dữ liệu này— làm việc “đằng sau hậu trường”—hỗ trợ cơ chế gọi/trả lại chức năng. Nó cũng hỗ trợ chuyển việc tạo, duy trì và hủy biến tự động của từng chức năng được gọi khả năng. Chúng tôi đã giải thích hành vi nhập sau, xuất trước (LIFO) của ngăn xếp với xếp chồng đĩa của chúng tôi ví dụ. Như chúng ta sẽ thấy trong hình. 5.7–5.9, hành vi LIFO này chính xác là những gì một chức năng thực hiện khi quay lại chức năng đã gọi nó.

Khi mỗi chức năng được gọi, nó có thể gọi các chức năng khác, chức năng này có thể gọi chức năng khác- tions—tất cả trước khi bất kỳ hàm nào trả về. Mỗi chức năng cuối cùng phải trả lại quyền kiểm soát cho chức năng đã gọi nó. Vì vậy, chúng ta phải theo dõi các địa chỉ trả về mà mỗi chức năng cần trả lại quyền điều khiển cho chức năng đã gọi nó. Ngăn xếp cuộc gọi chức năng là hoàn hảo cấu trúc dữ liệu để xử lý thông tin này. Mỗi khi một chức năng gọi một chức năng khác, một mục nhập được đẩy vào ngăn xếp. Mục nhập này, được gọi là khung ngăn xếp, chứa kết quả trả về địa chỉ mà hàm được gọi cần để quay lại hàm gọi. Nó cũng con- có một số thông tin bổ sung mà chúng tôi sẽ sớm thảo luận. Nếu hàm được gọi trả về, thay vào đó gọi một chức năng khác trước khi quay lại, khung ngăn xếp cho lệnh gọi chức năng sẽ xuất hiện, và điều khiển chuyển đến địa chỉ trả về trong khung ngăn xếp được bật lên.

Mỗi chức năng được gọi luôn tìm thấy thông tin cần thiết để trả về cho người gọi của nó tại đầu ngăn xếp cuộc gọi. Và, nếu một chức năng thực hiện cuộc gọi đến một chức năng khác, khung ngăn xếp cho cuộc gọi chức năng mới chỉ đơn giản là đẩy vào ngăn xếp cuộc gọi. Do đó, địa chỉ trả về được yêu cầu bởi chức năng mới được gọi để trả về trình gọi của nó hiện nằm ở đầu ngăn xếp.

Các khung ngăn xếp có một trách nhiệm quan trọng khác. Hầu hết các chức năng đều có tự động biến matic—các tham số và một số hoặc tất cả các biến cục bộ của chúng. biến tự động cần phải tồn tại trong khi một chức năng đang thực thi. Chúng cần duy trì hoạt động nếu chức năng thực hiện cuộc gọi đến các chức năng khác. Nhưng khi một hàm được gọi trở lại trình gọi của nó, hàm được gọi các biến tự động của hàm cần phải “biến mất”. Khung ngăn xếp của hàm được gọi là một nơi hoàn hảo để dành bộ nhớ cho các biến tự động. Khung ngăn xếp đó chỉ tồn tại miễn là chức năng được gọi đang hoạt động. Khi chức năng đó trở lại—và không cần nữa các biến tự động cục bộ của nó—khung ngăn xếp của nó được bật ra khỏi ngăn xếp và các biến tự động cục bộ đó biến matic không còn được biết đến với chương trình.

Tất nhiên dung lượng bộ nhớ trong máy tính là hữu hạn nên chỉ có dung lượng nhất định bộ nhớ có thể được sử dụng để lưu trữ các khung ngăn xếp trên ngăn xếp lệnh gọi hàm. Nếu có thêm chức năng các cuộc gọi xảy ra hơn có thể có các khung ngăn xếp của chúng được lưu trữ trên ngăn xếp cuộc gọi chức năng, một lỗi nghiêm trọng được gọi là tràn ngăn xếp xảy ra.

Ngăn xếp cuộc gọi chức năng đang hoạt động Bây giờ hãy xem xét cách ngăn xếp cuộc gọi hỗ trợ hoạt động của một hàm vuông được gọi là theo đường chính (dòng 8–13 của Hình 5.6). Đầu tiên, hệ điều hành gọi chính—điều này sẽ đẩy một ngăn xếp frame vào ngăn xếp (hiển thị trong Hình 5.7). Khung ngăn xếp cho biết chính cách quay lại hệ điều hành (nghĩa là chuyển đến địa chỉ trả về R1) và chứa không gian cho au- biến tomatic (nghĩa là a, được khởi tạo thành 10).

Chức năng chính—trước khi quay trở lại hệ điều hành—hiện đang gọi chức năng vuông ở dòng 12 của Hình 5.6. Điều này khiến khung ngăn xếp cho hình vuông (dòng 16–19) bị được đẩy vào ngăn xếp lời gọi hàm (Hình 5.8). Khung ngăn xếp này chứa địa chỉ trả về hình vuông đó cần trở về chính (tức là R2) và bộ nhớ cho biến tự động của hình vuông có thể (tức là, x).

1 // Hình 5.6: fig05\_06.c

2 // Minh họa ngăn xếp lời gọi hàm

3 // và ngăn xếp các khung bằng cách sử dụng một hàm vuông.

4 #include <stdio.h>

5

6 int vuông( int ); // nguyên mẫu cho hàm square

7

8 int chính()

9 {

10 int a = 10; // giá trị thành bình phương (biến tự động cục bộ trong chính)

11

12 printf("%d bình phương: %d\n", a, ); // hiển thị bình phương

13 } // kết thúc chính

14

15 // trả về bình phương của một số nguyên

16 int square( int x ) // x là biến cục bộ

17 {

18 trả lại x \* x; // tính bình phương và trả về kết quả

19 } // kết thúc hàm bình phương

10 bình phương: 100

Hình 5.6 | Biểu diễn ngăn xếp lệnh gọi hàm và khung ngăn xếp bằng cách sử dụng hình vuông hàm.

Sau khi square tính bình phương của đối số của nó, nó cần quay lại main—và không còn cần bộ nhớ cho biến x tự động của nó. Vì vậy, ngăn xếp được bật lên — đưa ra bình phương vị trí trả về trong chính (tức là R2) và làm mất biến tự động của hình vuông.

Hình 5.9 cho thấy ngăn xếp lời gọi hàm sau khi khung ngăn xếp của hình vuông được bật lên.

Chức năng chính hiện hiển thị kết quả gọi hình vuông (dòng 12). đạt đến đóng dấu ngoặc phải của main khiến khung ngăn xếp của nó bị bật ra khỏi ngăn xếp, mang lại cho main địa chỉ mà nó cần để trả về hệ điều hành (tức là R1 trong Hình 5.7) và gây ra lỗi bộ nhớ cho biến tự động của main (tức là a) không khả dụng.

Bây giờ bạn đã thấy cấu trúc dữ liệu ngăn xếp có giá trị như thế nào trong việc triển khai cơ chế chính anism hỗ trợ thực hiện chương trình. Cấu trúc dữ liệu có nhiều ứng dụng quan trọng trong khoa học máy tính. Chúng tôi thảo luận về ngăn xếp, hàng đợi, danh sách, cây và các cấu trúc dữ liệu khác trong Chương 12.

**5.8 Tiêu đề**

Mỗi thư viện chuẩn có một tiêu đề tương ứng chứa nguyên mẫu hàm cho tất cả các chức năng trong thư viện đó và các định nghĩa về các loại dữ liệu và hằng số cần thiết bởi các chức năng đó. Hình 5.10 liệt kê theo thứ tự bảng chữ cái một số tiêu đề thư viện chuẩn có thể được đưa vào các chương trình. Tiêu chuẩn C bao gồm các tiêu đề bổ sung. Thuật ngữ “macro” được sử dụng nhiều lần trong Hình 5.10 sẽ được thảo luận chi tiết trong Chương 13.

Bạn có thể tạo tiêu đề tùy chỉnh. Các tiêu đề do lập trình viên xác định cũng nên sử dụng .h phần mở rộng tên tệp. Tiêu đề do lập trình viên xác định có thể được bao gồm bằng cách sử dụng #include chỉ thị tiền xử lý. Ví dụ: nếu nguyên mẫu cho hàm vuông của chúng tôi được định vị trong tiêu đề square.h, chúng tôi sẽ đưa tiêu đề đó vào chương trình của mình bằng cách sử dụng cách sau lệnh ở đầu chương trình:

#include "square.h"

Mục 13.2 trình bày thông tin bổ sung về việc bao gồm các tiêu đề.

**5.9 Truyền đối số theo giá trị và theo tham chiếu**

Trong nhiều ngôn ngữ lập trình, có hai cách để truyền đối số—truyền theo giá trị và chuyển qua tham chiếu. Khi các đối số được truyền theo giá trị, một bản sao giá trị của đối số được thực hiện và chuyển đến hàm được gọi. Thay đổi bản sao không ảnh hưởng đến bản gốc giá trị của biến trong trình gọi. Khi một đối số được truyền bằng tham chiếu, người gọi cho phép được gọi là hàm để sửa đổi giá trị của biến ban đầu. Pass-by-value nên được sử dụng bất cứ khi nào chức năng được gọi không cần sửa đổi giá trị của biến ban đầu của người gọi. Điều này ngăn ngừa các tác dụng phụ ngẫu nhiên (biến sửa đổi) gây cản trở lớn đến sự phát triển của hệ thống phần mềm chính xác và đáng tin cậy.tems. Pass-by-reference chỉ nên được sử dụng với các chức năng được gọi đáng tin cậy cần sửa đổi biến ban đầu.

Trong C, tất cả các đối số được truyền theo giá trị. Như chúng ta sẽ thấy trong Chương 7, có thể mô phỏng mô phỏng chuyển qua tham chiếu bằng cách sử dụng toán tử địa chỉ và toán tử gián tiếp. Trong Chương 6, chúng ta sẽ thấy rằng các đối số mảng được tự động chuyển qua tham chiếu để thực hiện lý do mance. Chúng ta sẽ thấy trong Chương 7 rằng điều này không mâu thuẫn. Hiện tại, chúng tôi tập trung đánh giá trên pass-by-giá trị.

**5.10 Tạo số ngẫu nhiên**

Bây giờ chúng ta sẽ có một bản tóm tắt ngắn gọn và, hy vọng, mang tính giải trí sang mô phỏng và chơi trò chơi- ing. Trong phần này và phần tiếp theo, chúng ta sẽ phát triển một chương trình chơi trò chơi có cấu trúc độc đáo bao gồm nhiều chức năng. Chương trình sử dụng hầu hết các câu lệnh điều khiển mà chúng tôi đã đã học. Yếu tố may rủi có thể được đưa vào các ứng dụng máy tính bằng cách sử dụng Hàm thư viện chuẩn C rand từ tiêu đề <stdlib.h>.

Hãy xem xét tuyên bố sau:

i = rand();

Hàm rand tạo ra một số nguyên trong khoảng từ 0 đến RAND\_MAX (một hằng số biểu tượng bị phạt trong tiêu đề <stdlib.h>). Tiêu chuẩn C nói rằng giá trị của RAND\_MAX phải ở ít nhất là 32767, là giá trị lớn nhất cho số nguyên hai byte (tức là 16 bit). Người chuyên nghiệp- gam trong phần này đã được thử nghiệm trên Microsoft Visual C++ với giá trị RAND\_MAX tối đa ue là 32767 và trên GNU gcc với giá trị RAND\_MAX tối đa là 2147483647. Nếu rand thực sự tạo ra các số nguyên một cách ngẫu nhiên, mọi số từ 0 đến RAND\_MAX đều có giá trị bằng nhau cơ hội (hoặc xác suất) được chọn mỗi khi rand được gọi.

Phạm vi giá trị do rand trực tiếp tạo ra thường khác với giá trị cần thiết trong một ứng dụng cụ thể. Ví dụ, một chương trình mô phỏng việc tung đồng xu có thể yêu cầu chỉ 0 cho “đầu” và 1 cho “sấp”. Một chương trình tung xúc xắc mô phỏng một con súc sắc sáu mặt sẽ yêu cầu số nguyên ngẫu nhiên từ 1 đến 6.

Lăn xúc xắc sáu mặt

Để chứng minh rand, hãy phát triển một chương trình mô phỏng 20 lần cuộn của một con súc sắc sáu mặt và in giá trị của mỗi cuộn. Nguyên mẫu hàm cho hàm rand nằm trong <stdlib.h>. Chúng tôi sử dụng toán tử còn lại (%) kết hợp với rand như sau

rand() % 6

để tạo ra các số nguyên trong phạm vi từ 0 đến 5. Điều này được gọi là chia tỷ lệ. Số 6 được gọi là yếu tố nhân rộng. Sau đó, chúng tôi thay đổi phạm vi số được tạo bằng cách thêm 1 vào trước đó của chúng tôi kết quả. Đầu ra của Hình 5.11 xác nhận rằng các kết quả nằm trong phạm vi từ 1 đến 6—giá trị thực tế kết quả. Đầu ra của Hình 5.11 xác nhận rằng các kết quả nằm trong phạm vi từ 1 đến 6—giá trị thực tế các giá trị ngẫu nhiên được chọn có thể khác nhau tùy theo trình biên dịch.

1 // Hình 5.11: fig05\_11.c

2 // Các số nguyên ngẫu nhiên được thay đổi tỷ lệ được tạo bởi 1 + rand() % 6.

3 #include <stdio.h>

4 #include <stdlib.h>

5

6 // hàm main bắt đầu thực hiện chương trình

7 int chính (khoảng trống)

số 8 {

9 không dấu int i; // quầy tính tiền

10

11 // lặp 20 lần

12 cho ( i = 1; i <= 20; ++i ) {

13

14 // chọn số ngẫu nhiên từ 1 đến 6 và xuất ra

15 printf("%10d", );

16

17 // nếu bộ đếm chia hết cho 5, bắt đầu dòng đầu ra mới

18 nếu ( i % 5 == 0 ) {

19 lần đặt("" );

20 } // kết thúc nếu

21 } // kết thúc cho

22 } // kết thúc chính

6 6 5 5 6

5 1 1 5 3

6 6 2 4 2

6 2 3 4 1

Hình 5.11 | Các số nguyên ngẫu nhiên được thay đổi tỷ lệ được tạo bởi 1 + rand() % 6. (Phần 2/2.)

Lăn xúc xắc sáu mặt 6.000.000 lần

Để chỉ ra rằng những con số này xảy ra xấp xỉ với khả năng như nhau, hãy mô phỏng 6.000.000 cuộn súc sắc với chương trình của Hình 5.12. Mỗi số nguyên từ 1 đến 6 nên xuất hiện khoảng 1.000.000 lần.

Như đầu ra của chương trình cho thấy, bằng cách chia tỷ lệ và dịch chuyển, chúng tôi đã sử dụng hàm rand để mô phỏng thực tế việc lăn một con súc sắc sáu mặt. Lưu ý việc sử dụng đặc tả chuyển đổi %s nhanh hơn để in các chuỗi ký tự "Khuôn mặt" và "Tần suất" làm tiêu đề cột (dòng 53). Sau khi chúng ta nghiên cứu về mảng trong Chương 6, chúng ta sẽ chỉ ra cách thay thế trạng thái công tắc 26 dòng này- đề cập một cách tao nhã với một tuyên bố một dòng. Ngẫu nhiên Trình tạo số ngẫu nhiên

Thực hiện chương trình của Hình 5.11 một lần nữa tạo ra

6 6 5 5 6

5 1 1 5 3

6 6 2 4 2

6 2 3 4 1

Lưu ý rằng chính xác cùng một chuỗi giá trị đã được in. Làm thế nào đây có thể là số ngẫu nhiên- bạn? Trớ trêu thay, tính lặp lại này lại là một đặc tính quan trọng của hàm rand. Khi de- làm lỗi một chương trình, khả năng lặp lại này là cần thiết để chứng minh rằng các sửa chữa đối với một chương trình làm việc đúng cách.

Hàm rand thực sự tạo ra các số giả ngẫu nhiên. Gọi rand liên tục tạo ra một dãy số có vẻ như là ngẫu nhiên. Tuy nhiên, trình tự lặp lại chính nó mỗi khi chương trình được thực thi. Khi một chương trình đã được gỡ lỗi kỹ lưỡng, nó có thể được tạo điều kiện để tạo ra một chuỗi số ngẫu nhiên khác nhau cho mỗi lần thực hiện. Điều này được gọi là ngẫu nhiên hóa và được thực hiện với chức năng thư viện tiêu chuẩn srand. Hàm srand lấy một đối số nguyên không dấu và hạt giống hàm rand thành tạo ra một dãy số ngẫu nhiên khác nhau cho mỗi lần thực hiện chương trình.

Chúng ta minh họa hàm srand trong Hình 5.13. Hàm srand lấy một số nguyên không dấu giá trị như một đối số. Trình xác định chuyển đổi %u được sử dụng để đọc giá trị int không dấu với scanf. Nguyên mẫu hàm cho srand được tìm thấy trong <stdlib.h>. Hãy chạy chương trình vài lần và quan sát kết quả. Lưu ý rằng một sự khác biệt chuỗi các số ngẫu nhiên được lấy mỗi khi chương trình được chạy, với điều kiện là một hạt giống khác nhau được cung cấp.

Để chọn ngẫu nhiên mà không cần nhập hạt giống mỗi lần, hãy sử dụng câu lệnh như

srand( time( NULL ) );

Điều này làm cho máy tính đọc đồng hồ của nó để lấy giá trị cho hạt giống một cách tự động. Hàm time trả về số giây đã trôi qua kể từ nửa đêm ngày 1 tháng 1 1, 1970. Giá trị này được chuyển đổi thành một số nguyên không dấu và được sử dụng làm khởi đầu cho biến ngẫu nhiên trình tạo số. Nguyên mẫu hàm cho thời gian nằm trong <time.h>. Chúng tôi sẽ nói thêm về NULL trong Chương 7.

Chia tỷ lệ tổng quát và dịch chuyển số ngẫu nhiên

Các giá trị được tạo trực tiếp bởi rand luôn nằm trong phạm vi:

0 ≤ rand() ≤ RAND\_MAX

Như bạn đã biết, câu lệnh sau đây mô phỏng việc tung một con súc sắc sáu mặt:

face = 1 + rand() % 6;

Câu lệnh này luôn gán một giá trị nguyên (ngẫu nhiên) cho biến face trong phạm vi 1 ≤ mặt ≤ 6. Độ rộng của phạm vi này (nghĩa là số lượng các số nguyên liên tiếp trong phạm vi) là 6 và số bắt đầu trong phạm vi là 1. Đề cập đến trạng thái trước- lưu ý, chúng ta thấy rằng chiều rộng của phạm vi được xác định bởi số được sử dụng để chia tỷ lệ rand với toán tử còn lại (nghĩa là 6) và số bắt đầu của phạm vi bằng với số (tức là 1) được thêm vào rand % 6. Chúng ta có thể khái quát hóa kết quả này như sau:

n = a + rand() % b;

trong đó a là giá trị dịch chuyển (bằng với số đầu tiên trong phạm vi mong muốn của số nguyên liên tiếp) và b là hệ số tỷ lệ (bằng với chiều rộng của ô mong muốn dãy số nguyên liên tiếp). Trong các bài tập, chúng ta sẽ thấy rằng có thể chọn số nguyên ngẫu nhiên từ các tập giá trị khác với dãy các số nguyên liên tiếp.

Lỗi lập trình thường gặp 5.7

Sử dụng srand thay cho rand để tạo số ngẫu nhiên.

**5.11 Ví dụ: Trò chơi may rủi**

Một trong những trò chơi may rủi phổ biến nhất là trò chơi xúc xắc được gọi là “craps”, được chơi trong các sòng bạc và ngõ hẻm trên khắp thế giới. Các quy tắc của trò chơi rất đơn giản:

Một người chơi tung hai con xúc xắc. Mỗi con súc sắc có sáu mặt. Các mặt này chứa 1, 2, 3, 4, 5 và 6 điểm. Sau khi con súc sắc đứng yên thì tổng số chấm ở hai mặt ngửa là tính toán. Nếu tổng là 7 hoặc 11 trong lần ném đầu tiên, người chơi sẽ thắng. Nếu tổng là 2, 3, hoặc 12 trong lần ném đầu tiên (được gọi là "craps"), người chơi thua (tức là "nhà cái" thắng). Nếu tổng là 4, 5, 6, 8, 9 hoặc 10 trong lần ném đầu tiên, thì tổng đó sẽ trở thành của người chơi "điểm." Để giành chiến thắng, bạn phải tiếp tục tung xúc xắc cho đến khi bạn “làm rõ quan điểm của mình”. Các người chơi thua bằng cách lăn số 7 trước khi ghi điểm.

Trong luật chơi, lưu ý rằng người chơi phải tung hai viên xúc xắc trong lần tung đầu tiên, và phải làm như vậy sau này trên tất cả các cuộn tiếp theo. Chúng tôi xác định một hàm rollDice để tung xúc xắc và tính toán và in ra tổng của chúng. Chức năng rollDice được xác định một lần, nhưng nó được gọi từ hai vị trí trong chương trình (dòng 23 và 51). Thật thú vị, rollDice không có đối số, vì vậy chúng tôi đã chỉ ra khoảng trống trong danh sách tham số (dòng 74). Hàm rollDice không trả về tổng của hai con xúc xắc, do đó, kiểu trả về của int được chỉ định trong tiêu đề hàm và trong nguyên mẫu chức năng.

phép liệt kê

Trò chơi được tham gia hợp lý. Người chơi có thể thắng hoặc thua trong lần tung đầu tiên, hoặc có thể thắng hoặc thua trong bất kỳ lần tung tiếp theo nào. Biến gameStatus, được định nghĩa là một kiểu mới—enum Status—lưu trữ trạng thái hiện tại. Dòng 8 tạo một loại do lập trình viên định nghĩa được gọi là liệt kê. Một phép liệt kê, được giới thiệu bởi từ khóa enum, là một tập hợp các số nguyên stants đại diện bởi định danh. Hằng số liệt kê đôi khi được gọi là tượng trưng hằng số. Các giá trị trong một enum bắt đầu bằng 0 và được tăng thêm 1. Ở dòng 8, hằng số TIẾP TỤC có giá trị 0, WON có giá trị 1 và LOST có giá trị 2. Cũng có thể gán một giá trị số nguyên cho mỗi mã định danh trong một enum (xem Chương 10). Các định danh trong một liệt kê phải là duy nhất, nhưng các giá trị có thể bị trùng lặp.

Lỗi lập trình thường gặp 5.8

Gán một giá trị cho một hằng số liệt kê sau khi nó đã được xác định là một lỗi cú pháp.

Thực hành lập trình tốt 5.6

Chỉ sử dụng các chữ cái viết hoa trong tên của các hằng số liệt kê để tạo các hằng số này nổi bật trong một chương trình và để chỉ ra rằng hằng số liệt kê không phải là biến. Khi trò chơi thắng, ở lượt đầu tiên hoặc ở lượt tiếp theo, gameStatus là đặt thành THẮNG. Khi trò chơi bị thua, ở lần tung đầu tiên hoặc ở lần tung tiếp theo, trò chơi. Trạng thái được đặt thành LOST. Nếu không, gameStatus được đặt thành TIẾP TỤC và trò chơi tiếp tục. Trò chơi kết thúc trên cuộn đầu tiên Sau lần tung đầu tiên, nếu trò chơi kết thúc, câu lệnh while (dòng 50–62) sẽ bị bỏ qua vì gameStatus không TIẾP TỤC. Chương trình chuyển sang câu lệnh if...else tại các dòng 65–70, hiển thị "Người chơi thắng" nếu gameStatus là THẮNG và "Người chơi thua" nếu không. Trò chơi kết thúc ở một cuộn tiếp theo Sau lần quay đầu tiên, nếu trò chơi chưa kết thúc, thì tổng sẽ được lưu vào myPoint. tiền thu được với câu lệnh while vì gameStatus là TIẾP TỤC. Mỗi lần qua một lúc, rollDice được gọi để tạo ra một tổng mới. Nếu tổng khớp với myPoint, gameStatus được đặt thành WON để chỉ ra rằng người chơi đã thắng, kiểm tra trong khi không thành công, câu lệnh if...else in "Play- er thắng" và quá trình thực thi kết thúc. Nếu tổng bằng 7 (dòng 58), gameStatus được đặt thành LOST để chỉ ra rằng người chơi bị thua, kiểm tra trong khi không thành công, câu lệnh if...else in ra "Người chơi thua" và thực hiện chấm dứt.

Kiến trúc điều khiển

Lưu ý kiến ​​trúc điều khiển thú vị của chương trình. Chúng tôi đã sử dụng hai hàm—chính và rollDice—và switch, while, if...else lồng nhau và các câu lệnh if lồng nhau. Trong ex- bài tập, chúng tôi sẽ điều tra các đặc điểm thú vị khác nhau của trò chơi xúc xắc.

**5.12 Các lớp lưu trữ**

Trong các Chương 2–4, chúng tôi đã sử dụng các mã định danh cho các tên biến. Các thuộc tính của biến bao gồm tên, loại, kích thước và giá trị. Trong chương này, chúng tôi cũng sử dụng định danh làm tên cho người dùng định nghĩa chức năng. Trên thực tế, mỗi định danh trong một chương trình có các thuộc tính khác, bao gồm lưu trữ lớp, thời lượng lưu trữ, phạm vi và liên kết.

C cung cấp các chỉ định lớp lưu trữ tự động, register1, bên ngoài và tĩnh. 2 một lớp lưu trữ của định danh xác định thời lượng lưu trữ, phạm vi và liên kết của nó. Một định danh thời lượng lưu trữ là khoảng thời gian mà mã định danh tồn tại trong bộ nhớ. một số tồn tại một thời gian ngắn, một số được tạo và hủy lặp đi lặp lại, và một số khác tồn tại trong toàn bộ chương trình chấp hành. Phạm vi của mã định danh là nơi mã định danh có thể được tham chiếu trong một chương trình. chấp hành. Phạm vi của mã định danh là nơi mã định danh có thể được tham chiếu trong một chương trình. Liên kết của một mã định danh xác định cho chương trình nhiều tệp nguồn liệu mã định danh có fier chỉ được biết đến trong tệp nguồn hiện tại hoặc trong bất kỳ tệp nguồn nào có khai báo phù hợp.

Phần này thảo luận về các lớp lưu trữ và thời lượng lưu trữ. Mục 5.13 thảo luận về phạm vi. Chương 14 thảo luận về liên kết định danh và lập trình với nhiều tệp nguồn. Các chỉ định lớp lưu trữ có thể được phân chia thời lượng lưu trữ tự động và lưu trữ tĩnh khoảng thời gian. Từ khóa auto dùng để khai báo biến thời lượng lưu trữ tự động. Biến với thời lượng lưu trữ tự động được tạo khi khối mà chúng được xác định là

với thời lượng lưu trữ tự động được tạo khi khối mà chúng được xác định là đã vào; chúng tồn tại trong khi khối đang hoạt động và chúng bị hủy khi khối được thoát.

Biến cục bộ

Chỉ các biến mới có thể có thời lượng lưu trữ tự động. Các biến cục bộ của hàm (những biến khai báo trong danh sách tham số hoặc thân hàm) thường có thời lượng lưu trữ tự động.

Từ khóa auto khai báo rõ ràng các biến thời lượng lưu trữ tự động. Biến cục bộ mặc định có thời lượng lưu trữ tự động nên từ khóa auto ít được sử dụng. Đối với phần còn lại- Ngoài văn bản, chúng tôi sẽ đề cập đến các biến có thời lượng lưu trữ tự động đơn giản là tự động biến.

Mẹo hiệu suất 5.1

Lưu trữ tự động là một phương tiện bảo tồn bộ nhớ, bởi vì các biến tự động tồn tại chỉ khi chúng cần thiết. Chúng được tạo khi một chức năng được nhập và bị hủy khi chức năng được thoát ra. Các từ khóa bên ngoài và tĩnh được sử dụng trong khai báo các định danh cho các biến và chức năng của thời lượng lưu trữ tĩnh. Mã định danh thời lượng lưu trữ tĩnh tồn tại từ thời điểm tại đó chương trình bắt đầu thực hiện cho đến khi chương trình kết thúc. Đối với các biến tĩnh, lưu trữ được phân bổ và khởi tạo chỉ một lần, trước khi chương trình bắt đầu thực hiện. Đối với chức năng- tions, tên của chức năng tồn tại khi chương trình bắt đầu thực hiện. Tuy nhiên, ngay cả mặc dù các biến và tên hàm tồn tại từ khi bắt đầu thực hiện chương trình, điều này không có nghĩa là những mã định danh này có thể được truy cập trong suốt chương trình. Lưu trữ du- tỷ lệ và phạm vi (nơi có thể sử dụng tên) là những vấn đề riêng biệt, như chúng ta sẽ thấy trong Phần 5.13.

Có một số loại mã định danh có thời lượng lưu trữ tĩnh: mã định danh bên ngoài (chẳng hạn như biến toàn cục và tên hàm) và biến cục bộ được khai báo với bộ lưu trữ- trình xác định lớp tĩnh. Các biến toàn cục và tên hàm thuộc lớp lưu trữ bên ngoài bởi mặc định. Biến toàn cục được tạo bằng cách đặt các khai báo biến bên ngoài bất kỳ hàm nào định nghĩa và chúng giữ lại các giá trị của chúng trong suốt quá trình thực thi chương trình. Toàn cầu các biến và hàm có thể được tham chiếu bởi bất kỳ hàm nào tuân theo khai báo của chúng hoặc định nghĩa trong tệp. Đây là một lý do để sử dụng nguyên mẫu hàm—khi chúng ta bao gồm stdio.h trong một chương trình gọi printf, nguyên mẫu hàm được đặt ở bắt đầu tệp của chúng tôi để làm cho tên printf được biết đến với phần còn lại của tệp.

Quan sát kỹ thuật phần mềm 5.10

Xác định một biến là toàn cầu thay vì cục bộ cho phép xảy ra các tác dụng phụ ngoài ý muốn khi một chức năng không cần truy cập vào biến vô tình hoặc cố ý sửa đổi nó. Nói chung, nên tránh các biến toàn cục ngoại trừ trong một số trường hợp nhất định với các yêu cầu hiệu suất duy nhất (như đã thảo luận trong Chương 14).

Quan sát kỹ thuật phần mềm 5.11

Các biến chỉ được sử dụng trong một chức năng cụ thể nên được định nghĩa là biến cục bộ trong đó chức năng chứ không phải là biến bên ngoài.

Biến cục bộ được khai báo với từ khóa tĩnh vẫn chỉ được biết trong hàm trong đó chúng được xác định, nhưng không giống như các biến tự động, các biến cục bộ tĩnh giữ nguyên giá trị khi thoát chức năng. Lần tiếp theo hàm được gọi, cục bộ tĩnh biến chứa giá trị mà nó có khi hàm thoát lần cuối. Tuyên bố sau đây tuyên bố số lượng biến cục bộ là tĩnh và khởi tạo nó thành 1.

static int count = 1;

Tất cả các biến số của thời lượng lưu trữ tĩnh được khởi tạo bằng 0 theo mặc định nếu bạn thực hiện không khởi tạo chúng một cách rõ ràng.

Từ khóa bên ngoài và tĩnh có ý nghĩa đặc biệt khi được áp dụng rõ ràng cho định danh bên ngoài. Trong Chương 14, chúng ta thảo luận về việc sử dụng rõ ràng extern và static với mã định danh bên ngoài và chương trình nhiều tệp nguồn.

**5.13 Quy tắc phạm vi**

Phạm vi của một mã định danh là một phần của chương trình trong đó mã định danh có thể được giới thiệu phát sinh. Ví dụ, khi chúng ta định nghĩa một biến cục bộ trong một khối, nó chỉ có thể được tham chiếu theo định nghĩa của nó trong khối đó hoặc trong các khối được lồng trong khối đó. Bốn iden- phạm vi trình xác định là phạm vi chức năng, phạm vi tệp, phạm vi khối và phạm vi nguyên mẫu chức năng.

Nhãn (số nhận dạng theo sau bởi dấu hai chấm chẳng hạn như bắt đầu:) là số nhận dạng duy nhất có chức năng- phạm vi. Nhãn có thể được sử dụng ở bất kỳ đâu trong chức năng mà chúng xuất hiện, nhưng không thể được tham chiếu bên ngoài thân hàm. Nhãn được sử dụng trong các câu lệnh chuyển đổi (như trường hợp la- bels) và trong câu lệnh goto (xem Chương 14). Nhãn là chi tiết triển khai mà chức năng ẩn với nhau. Việc che giấu này—chính thức hơn được gọi là giấu thông tin- ing—là một phương tiện để thực hiện nguyên tắc đặc quyền tối thiểu—một nguyên tắc cơ bản ple của kỹ thuật phần mềm tốt. Trong bối cảnh của một ứng dụng, nguyên tắc nói rằng mã chỉ nên được cấp số lượng đặc quyền và quyền truy cập mà nó cần để hoàn thành nhiệm vụ được chỉ định của nó, nhưng không còn nữa.

Một mã định danh được khai báo bên ngoài bất kỳ chức năng nào có phạm vi tệp. Một định danh như vậy là “đã biết” (nghĩa là có thể truy cập) trong tất cả các chức năng từ thời điểm mà mã định danh được khai báo cho đến khi kết thúc tập tin. Biến toàn cục, định nghĩa hàm và nguyên mẫu hàm được đặt bên ngoài một chức năng đều có phạm vi tệp.

Định danh được xác định bên trong một khối có phạm vi khối. Phạm vi khối kết thúc tại termi- dấu ngoặc nhọn phải (}) của khối. Các biến cục bộ được xác định ở đầu hàm có phạm vi khối, cũng như các tham số chức năng, được coi là biến cục bộ bởi chức năng. Bất kỳ khối nào cũng có thể chứa các định nghĩa biến. Khi các khối được lồng vào nhau và một giá trị nhận dạng định danh trong khối bên ngoài có cùng tên với định danh trong khối bên trong, định danh trong khối bên ngoài được ẩn cho đến khi khối bên trong kết thúc. Điều này có nghĩa là trong khi exe- cắt vào khối bên trong, khối bên trong nhìn thấy giá trị của mã định danh cục bộ của chính nó chứ không phải giá trị của mã định danh được đặt tên giống hệt nhau trong khối kèm theo. Biến cục bộ khai báo tĩnh vẫn có phạm vi khối, mặc dù chúng tồn tại từ trước chương trình khởi động. Do đó, thời lượng lưu trữ không ảnh hưởng đến phạm vi của mã định danh.

Các mã định danh duy nhất có phạm vi nguyên mẫu chức năng là những mã được sử dụng trong tham số danh sách nguyên mẫu hàm. Như đã đề cập trước đó, nguyên mẫu hàm không yêu cầu tên trong danh sách tham số—chỉ các loại được yêu cầu. Nếu một tên được sử dụng trong danh sách tham số của nguyên mẫu hàm, trình biên dịch sẽ bỏ qua tên đó. Định danh được sử dụng trong một chức năng pro- totype có thể được sử dụng lại ở những nơi khác trong chương trình mà không có sự mơ hồ.

Lỗi lập trình thường gặp 5.9

Vô tình sử dụng cùng một tên cho một mã định danh trong một khối bên trong như được sử dụng cho một mã định danh- bộ định danh trong một khối bên ngoài, trong khi thực tế bạn muốn số nhận dạng trong khối bên ngoài hoạt động trong suốt thời gian của khối bên trong.

Mẹo ngăn ngừa lỗi 5.4

Tránh các tên biến ẩn tên trong phạm vi bên ngoài.

Hình 5.16 minh họa các vấn đề về phạm vi với các biến toàn cục, biến cục bộ tự động ables và các biến cục bộ tĩnh. Biến toàn cục x được xác định và khởi tạo thành 1 (dòng 9).

Biến toàn cục này được ẩn trong bất kỳ khối (hoặc chức năng) nào trong đó một biến có tên x là xác định. Trong main, một biến cục bộ x được xác định và khởi tạo thành 5 (dòng 14). biến này là sau đó được in để hiển thị rằng x toàn cầu được ẩn trong chính. Tiếp theo, một khối mới được xác định trong main với một biến cục bộ khác x được khởi tạo thành 7 (dòng 19). Biến này được in để hiển thị rằng nó ẩn x trong khối bên ngoài của main. Biến x có giá trị 7 tự động bị hủy khi khối được thoát và biến cục bộ x trong khối chính bên ngoài là được in lại để cho thấy rằng nó không còn bị ẩn nữa.

Chương trình định nghĩa ba hàm mà mỗi hàm không nhận đối số và không trả về gì cả. Hàm useLocal định nghĩa một biến tự động x và khởi tạo nó thành 25 (dòng 39). Khi useLocal được gọi, biến được in, tăng dần và in lại trước khi thoát chức năng. Mỗi khi hàm này được gọi, biến tự động x được khởi tạo lại thành 25. Hàm useStaticLocal định nghĩa một biến tĩnh x và khởi tạo nó thành 50 ở dòng 52 (hãy nhớ rằng bộ lưu trữ cho các biến tĩnh chỉ được cấp phát và khởi tạo một lần, trước chương trình bắt đầu thực hiện). Các biến cục bộ được khai báo là tĩnh giữ nguyên giá trị của chúng ngay cả khi chúng nằm ngoài phạm vi. Khi useStaticLocal được gọi, x được in, tăng dần và được in lại trước khi thoát khỏi chức năng. Trong lần gọi hàm tiếp theo, tĩnh cục bộ biến x sẽ chứa giá trị 51. Hàm useGlobal không định nghĩa bất kỳ biến nào. Do đó, khi đề cập đến biến x, x toàn cục (dòng 9) được sử dụng. Khi useGlobal là được gọi, biến toàn cục được in, nhân với 10 và được in lại trước khi thoát khỏi chức năng. Lần tiếp theo hàm useGlobal được gọi, biến toàn cục vẫn có mod- giá trị đã xác định, 10. Cuối cùng, chương trình in lại biến cục bộ x trong main (dòng 33) thành cho thấy rằng không có lệnh gọi hàm nào sửa đổi giá trị của x vì tất cả các hàm được gọi đến các biến trong các phạm vi khác.

**5.14 Đệ quy**

Các chương trình chúng ta đã thảo luận thường được cấu trúc như các chức năng gọi lẫn nhau một cách có kỷ luật, có thứ bậc. Đối với một số loại vấn đề, sẽ rất hữu ích nếu có chức năng tự gọi mình. Một hàm đệ quy là một hàm gọi chính nó trực tiếp hoặc gián tiếp thông qua một chức năng khác. Đệ quy là một chủ đề phức tạp được thảo luận rất lâu trong các khóa học khoa học máy tính cấp trên. Trong phần này và các phần tiếp theo, các ví dụ đơn giản về tái Cursion được trình bày. Cuốn sách này bao gồm một cách xử lý mở rộng về đệ quy, đó là trải rộng khắp các Chương 5–8 và 12 và Phụ lục F. Hình 5.21, trong Phần 5.16, tóm tắt các ví dụ đệ quy và bài tập trong cuốn sách.

Chúng tôi xem xét đệ quy về mặt khái niệm trước, sau đó kiểm tra một số chương trình có chứa các hàm đệ quy. Phương pháp giải quyết vấn đề đệ quy có một số yếu tố trong chung. Một hàm đệ quy được gọi để giải quyết vấn đề. Chức năng thực sự biết cách chỉ giải quyết (các) trường hợp đơn giản nhất, hay còn gọi là (các) trường hợp cơ bản. Nếu chức năng được gọi với trường hợp cơ bản, hàm chỉ trả về kết quả. Nếu chức năng được gọi với một com- vấn đề phức tạp, chức năng chia vấn đề thành hai phần khái niệm: một phần mà chức năng biết cách thực hiện và một phần mà nó không biết cách thực hiện. Để làm cho đệ quy khả thi, phần sau phải giống với vấn đề ban đầu, nhưng hơi phiên bản đơn giản hơn hoặc nhỏ hơn. Bởi vì vấn đề mới này trông giống như vấn đề ban đầu, chức năng khởi chạy (gọi) một bản sao mới của chính nó để giải quyết vấn đề nhỏ hơn—điều này được gọi là một cuộc gọi đệ quy hoặc bước đệ quy. Bước đệ quy cũng bao gồm trả lại từ khóa, bởi vì kết quả của nó sẽ được kết hợp với phần của vấn đề chức năng biết cách giải quyết để tạo thành một kết quả sẽ được chuyển lại cho người gọi ban đầu.

Bước đệ quy thực hiện trong khi lệnh gọi hàm ban đầu chưa kết thúc thi hành. Bước đệ quy có thể dẫn đến nhiều cuộc gọi đệ quy như vậy, vì hàm tiếp tục chia từng vấn đề mà nó được gọi thành hai phần khái niệm. Đối với đệ quy để kết thúc, mỗi lần chức năng gọi chính nó bằng một phiên bản gốc đơn giản hơn một chút vấn đề cuối cùng, chuỗi các vấn đề nhỏ hơn này cuối cùng phải hội tụ về trường hợp cơ bản.

Khi hàm nhận ra trường hợp cơ sở, nó sẽ trả về kết quả cho bản sao trước đó của chức năng và một chuỗi trả về xảy ra trong suốt dòng cho đến khi lệnh gọi ban đầu của hàm cuối cùng trả về kết quả cuối cùng cho hàm main. Tất cả những điều này nghe có vẻ khá kỳ lạ.

Sánh ngang với kiểu giải quyết vấn đề mà chúng ta đang sử dụng với các lời gọi hàm truyền thống tới điểm này. Có thể mất rất nhiều thời gian để thực hành viết các chương trình đệ quy trước khi xử lý.

Như một ví dụ về các khái niệm này tại nơi làm việc, hãy viết một quy trình đệ quy gram để thực hiện một phép tính toán phổ biến.

Tính giai thừa đệ quy

Giai thừa của số nguyên n không âm, được viết là n! (phát âm là “n giai thừa”), là sản phẩm Úc

n · (n – 1) · (n – 2) · ... · 1

với 1! bằng 1 và 0! được định nghĩa là 1. Ví dụ: 5! là sản phẩm 5\*4\*3\*2\*1, bằng 120. Có thể tính giai thừa của một số nguyên, số lớn hơn hoặc bằng 0 lặp đi lặp lại (không đệ quy) bằng cách sử dụng câu lệnh for như sau:

factorial = 1;

for ( counter = number; counter >= 1; --counter )

factorial \*= counter;

Một định nghĩa đệ quy của hàm giai thừa được đưa ra bằng cách quan sát những điều sau đây mối quan hệ:

n! = n · (n – 1)!

Ví dụ, 5! rõ ràng là bằng 5 \* 4! như được hiển thị bởi những điều sau đây:

5! = 5 · 4 · 3 · 2 · 1

5! = 5 · (4 · 3 · 2 · 1)

5! = 5 · (4!)

Việc đánh giá 5! sẽ tiến hành như trong Hình 5.17. Hình 5.17(a) cho thấy cách sự liên tiếp của các cuộc gọi đệ quy tiến hành cho đến 1! được đánh giá là 1 (nghĩa là trường hợp cơ sở), mà chấm dứt đệ quy. Hình 5.17(b) hiển thị các giá trị được trả về từ mỗi lần lặp lại sive đến người gọi của nó cho đến khi giá trị cuối cùng được tính toán và trả về.

Hình 5.18 sử dụng đệ quy để tính và in giai thừa của các số nguyên 0–10 (việc chọn kiểu unsigned long long int sẽ được giải thích trong giây lát).

Hình 5.18 | Hàm giai thừa đệ quy. (Phần 2 của 2.)

Hàm giai thừa đệ quy trước tiên kiểm tra xem điều kiện kết thúc có đúng hay không, tức là, cho dù số nhỏ hơn hoặc bằng 1. Nếu số thực sự nhỏ hơn hoặc bằng 1, giai thừa trả về 1, không cần đệ quy nữa và chương trình kết thúc. Nếu như số lớn hơn 1, câu lệnh return number \* factorial( number - 1 ); biểu thị vấn đề dưới dạng tích của số và một lệnh gọi đệ quy để đánh giá giai thừa- lấy giai thừa của số - 1. Gọi giai thừa ( số - 1 ) là một bài toán đơn giản hơn một chút lem so với giai thừa tính toán ban đầu (số).

Hàm giai thừa (dòng 20–29) nhận một số nguyên không dấu và trả về kết quả là nhập unsigned long long int. Tiêu chuẩn C chỉ định rằng một biến kiểu không dấu long long int có thể chứa một giá trị ít nhất lớn bằng 18,446,744,073,709,551,615. Quét trong Hình 5.18, các giá trị giai thừa trở nên lớn nhanh chóng. Chúng tôi đã chọn loại dữ liệu unsigned long long int để chương trình có thể tính các giá trị giai thừa lớn hơn. chuyển đổi- sion specifier %llu được sử dụng để in các giá trị int long long không dấu. thật không may hàm giai thừa tạo ra các giá trị lớn nhanh đến mức thậm chí unsigned long long int không giúp chúng tôi in rất nhiều giá trị giai thừa trước giá trị lớn nhất của một dấu biến long long int bị vượt quá.

Ngay cả khi chúng tôi sử dụng unsigned long long int, chúng tôi vẫn không thể tính giai thừa vượt quá 21! Điều này chỉ ra một điểm yếu trong C (và hầu hết các ngôn ngữ lập trình thủ tục khác)— cụ thể là ngôn ngữ không dễ dàng mở rộng để xử lý các yêu cầu duy nhất của var- ứng dụng hữu ích. Như chúng ta sẽ thấy ở phần sau của cuốn sách, C++ là một ngôn ngữ có thể mở rộng, thông qua "các lớp" cho phép chúng tôi tạo các loại dữ liệu mới, bao gồm cả những loại có thể chứa lớn tùy ý số nguyên nếu chúng ta muốn.

Lỗi lập trình thường gặp 5.10

Quên trả lại giá trị từ hàm đệ quy khi cần.

Lỗi lập trình thường gặp 5.11

Bỏ qua trường hợp cơ sở hoặc viết bước đệ quy không chính xác để nó không hội tụ về trường hợp cơ sở, sẽ gây ra đệ quy vô hạn, cuối cùng làm cạn kiệt bộ nhớ.

Điều này tương tự như vấn đề về một vòng lặp vô hạn trong một giải pháp lặp (không đệ quy). Đệ quy vô hạn cũng có thể được gây ra bằng cách cung cấp đầu vào không mong muốn.

**5.15 Ví dụ Sử dụng Đệ quy: Dãy Fibonacci**

The Fibonacci series

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

bắt đầu bằng 0 và 1 và có thuộc tính là mỗi số Fibonacci tiếp theo là tổng của hai số Fibonacci trước đó.

Chuỗi xảy ra trong tự nhiên và đặc biệt, mô tả một dạng xoắn ốc. Tỷ lệ của các số Fibonacci liên tiếp hội tụ đến một giá trị không đổi là 1,618.... Con số này cũng vậy, xảy ra nhiều lần trong tự nhiên và được gọi là tỷ lệ vàng hay giá trị trung bình vàng.

Con người có xu hướng tìm ý nghĩa vàng về mặt thẩm mỹ. Kiến trúc sư thường thiết kế win- các tòa nhà, phòng và tòa nhà có chiều dài và chiều rộng theo tỷ lệ của giá trị trung bình vàng. Bưu thiếp thường được thiết kế với tỷ lệ chiều dài/chiều rộng trung bình vàng.

Dãy Fibonacci có thể được định nghĩa đệ quy như sau:

fibonacci(0) = 0

fibonacci(1) = 1

fibonacci(n) = fibonacci(n – 1) + fibonacci(n – 2)

Hình 5.19 tính toán đệ quy số Fibonacci thứ n bằng hàm fibonacci.

Lưu ý rằng các số Fibonacci có xu hướng trở nên lớn nhanh chóng. Vì vậy, chúng tôi đã chọn các kiểu dữ liệu unsigned int cho kiểu tham số và kiểu dữ liệu unsigned long long int cho kiểu trả về trong hàm fibonacci. Trong Hình 5.19, mỗi cặp dòng đầu ra hiển thị một chạy riêng của chương trình.

Hình 5.20 | Tập hợp các lời gọi đệ quy cho fibonacci( 3 ).

Hình 5.19 | Hàm fibonacci đệ quy. (Phần 2 của 2.)

Lệnh gọi đến fibonacci từ main không phải là lệnh gọi đệ quy (dòng 18), mà là tất cả lệnh gọi tiếp theo các cuộc gọi đến fibonacci là đệ quy (dòng 32). Mỗi lần gọi hàm fibonacci, nó ngay lập tức kiểm tra trường hợp cơ bản—n bằng 0 hoặc 1. Nếu điều này đúng, n được trả về. Thật thú vị, nếu n lớn hơn 1, bước đệ quy tạo ra hai lời gọi đệ quy, mỗi lời gọi đơn giản hơn một chút vấn đề hơn cuộc gọi ban đầu đến fibonacci. Hình 5.20 cho thấy cách hoạt động của hàm fibonacci sẽ đánh giá fibonacci(3).

Thứ tự đánh giá toán hạng

Con số này đặt ra một số vấn đề thú vị về thứ tự mà trình biên dịch C sẽ đánh giá toán hạng của toán tử. Đây là một vấn đề khác với thứ tự mà các nhà khai thác được áp dụng cho các toán hạng của chúng, cụ thể là thứ tự được quy định bởi các quy tắc về mức độ ưu tiên của toán tử.

Hình 5.20 cho thấy rằng trong khi đánh giá fibonacci(3), hai lệnh gọi đệ quy sẽ được thực hiện, cụ thể là fibonacci(2) và fibonacci(1). Nhưng những cuộc gọi này sẽ được thực hiện theo thứ tự nào? Bạn có thể đơn giản giả định rằng các toán hạng sẽ được đánh giá từ trái sang phải. Vì lý do tối ưu hóa, C không chỉ định thứ tự các toán hạng của hầu hết các toán tử (bao gồm cả +) được đánh giá. Vì vậy, bạn không nên đưa ra giả định về thứ tự mà các cuộc gọi sẽ thực hiện. Trên thực tế, các cuộc gọi có thể thực hiện fibonacci(2) trước và sau đó là fibonac- ci(1) hoặc lệnh gọi có thể thực hiện theo thứ tự ngược lại, fibonacci(1) rồi đến fibonacci(2).

Trong chương trình này và hầu hết các chương trình khác, kết quả cuối cùng sẽ giống nhau. Nhưng trong một số chương trình việc đánh giá một toán hạng có thể có tác dụng phụ có thể ảnh hưởng đến kết quả cuối cùng của sự biểu lộ. C chỉ định thứ tự đánh giá toán hạng của chỉ bốn toán tử— cụ thể là &&, ||, toán tử dấu phẩy (,) và ?:. Ba cái đầu tiên trong số này là toán tử nhị phân có toán hạng được đảm bảo để được đánh giá từ trái sang phải. [Lưu ý: Dấu phẩy được sử dụng để tách các đối số trong một lời gọi hàm không phải là toán tử dấu phẩy.] Toán tử cuối cùng là Toán tử bậc ba duy nhất của C. Toán hạng ngoài cùng bên trái của nó luôn được đánh giá đầu tiên; nếu op ngoài cùng bên trái erand đánh giá khác không, toán hạng ở giữa được đánh giá tiếp theo và toán hạng cuối cùng là làm ngơ; nếu toán hạng ngoài cùng bên trái đánh giá bằng 0, thì toán hạng thứ ba được đánh giá tiếp theo và toán hạng ở giữa bị bỏ qua.

Lỗi lập trình thường gặp 5.12

Viết các chương trình phụ thuộc vào thứ tự đánh giá toán hạng của các toán tử khác- hơn &&, ||, ?:, và toán tử dấu phẩy (,) có thể dẫn đến lỗi vì trình biên dịch có thể không nhất thiết phải đánh giá các toán hạng theo thứ tự bạn mong đợi.

Mẹo về tính di động 5.2

Các chương trình phụ thuộc vào thứ tự đánh giá toán hạng của các toán tử khác với &&, ||, ?:, và toán tử dấu phẩy (,) có thể hoạt động khác nhau trên các trình biên dịch khác nhau.

Độ phức tạp theo cấp số nhân

Một lời cảnh báo là về các chương trình đệ quy giống như chương trình chúng tôi sử dụng ở đây để tạo dãy số Fibonacci. Mỗi mức đệ quy trong hàm fibonacci có tác dụng nhân đôi trên số lượng cuộc gọi—số lượng cuộc gọi đệ quy sẽ được thực hiện để tính toán Số Fibonacci thứ n có thứ tự là 2n. Điều này nhanh chóng vượt khỏi tầm kiểm soát. chỉ tính toán số Fibonacci thứ 20 sẽ yêu cầu theo thứ tự 220 hoặc khoảng một triệu cuộc gọi, cal- tính toán số Fibonacci thứ 30 sẽ yêu cầu theo thứ tự 230 hoặc khoảng một tỷ cuộc gọi, v.v. Các nhà khoa học máy tính gọi đây là độ phức tạp theo cấp số nhân. Vấn đề này thiên nhiên khiêm tốn ngay cả những máy tính mạnh nhất thế giới! Các vấn đề phức tạp nói chung, và độ phức tạp hàm mũ nói riêng, được thảo luận chi tiết trong phần tổng hợp cấp trên. Khóa học khoa học máy tính thường được gọi là "Thuật toán".

Ví dụ chúng tôi trình bày trong phần này đã sử dụng một giải pháp trực quan hấp dẫn để tính toán sắp xếp các số Fibonacci, nhưng có nhiều cách tiếp cận tốt hơn. Bài tập 5.48 yêu cầu bạn đầu tư- tiến hành đệ quy sâu hơn và đề xuất các phương pháp thay thế để thực hiện thuật toán Fibonacci đệ quy.

**5.16 Đệ quy so với Lặp lại**

Trong các phần trước, chúng tôi đã nghiên cứu hai chức năng có thể dễ dàng thực hiện hoặc đệ quy hoặc lặp đi lặp lại. Trong phần này, chúng tôi so sánh hai cách tiếp cận và thảo luận tại sao bạn có thể chọn cách tiếp cận này thay vì cách tiếp cận khác trong một tình huống cụ thể.

• Cả lặp và đệ quy đều dựa trên cấu trúc điều khiển: Phép lặp sử dụng một cấu trúc lặp lại; đệ quy sử dụng một cấu trúc lựa chọn.

• Cả phép lặp và phép đệ quy đều liên quan đến phép lặp: Phép lặp rõ ràng sử dụng phép lặp tuyên bố tion; đệ quy đạt được sự lặp lại thông qua các lệnh gọi hàm lặp lại.

• Phép lặp và phép đệ quy đều liên quan đến phép thử kết thúc: Phép lặp kết thúc khi điều kiện tiếp tục vòng lặp không thành công; đệ quy khi một trường hợp cơ sở được công nhận.

• Phép lặp với phép lặp có kiểm soát ngược và phép đệ quy từng bước được áp dụng chấm dứt proach: Lặp lại tiếp tục sửa đổi bộ đếm cho đến khi bộ đếm như- tính tổng một giá trị làm cho điều kiện tiếp tục vòng lặp không thành công; đệ quy giữ tạo ra các phiên bản đơn giản hơn của vấn đề ban đầu cho đến khi đạt được trường hợp cơ sở.

• Cả lặp và đệ quy đều có thể xảy ra vô hạn: Một vòng lặp vô hạn xảy ra với nó- lặp lại nếu kiểm tra tiếp tục vòng lặp không bao giờ trở thành sai; đệ quy vô hạn xảy ra nếu bước đệ quy không làm giảm vấn đề mỗi lần theo cách các bờ vực trên trường hợp cơ sở.

Đệ quy có nhiều tiêu cực. Nó liên tục gọi cơ chế, và do đó chi phí hoạt động của các lời gọi hàm. Điều này có thể tốn kém cả về thời gian và bộ nhớ của bộ xử lý không gian. Mỗi lời gọi đệ quy gây ra một bản sao khác của hàm (thực ra chỉ có bản sao của hàm các biến) được tạo; điều này có thể tiêu tốn bộ nhớ đáng kể. Lặp lại thường xảy ra trong một chức năng, do đó, tổng chi phí của các cuộc gọi chức năng lặp lại và gán bộ nhớ bổ sung được bỏ qua. Vậy tại sao lại chọn đệ quy?

Quan sát kỹ thuật phần mềm 5.12

Bất kỳ vấn đề nào có thể được giải quyết theo cách đệ quy cũng có thể được giải quyết theo cách lặp lại (không đệ quy). Cách tiếp cận đệ quy thường được chọn hơn là cách tiếp cận lặp khi cách tiếp cận đệ quy phản ánh vấn đề một cách tự nhiên hơn và dẫn đến một chương trình dễ hiểu và gỡ lỗi hơn. Một lý do khác để chọn một giải pháp đệ quy là một giải pháp lặp lại có thể không rõ ràng. Hầu hết các sách giáo khoa lập trình giới thiệu đệ quy muộn hơn nhiều so với những gì chúng ta đã làm ở đây. Chúng tôi cảm thấy rằng đệ quy là một chủ đề đủ phong phú và phức tạp để giới thiệu nó sớm hơn và lan truyền các ví dụ trên phần còn lại của văn bản. Hình 5.21 tóm tắt theo chương 31 ví dụ đệ quy và bài tập trong văn bản.

Chương Đệ quy ví dụ và bài tập

Chương 5 Hàm giai thừa

Hàm Fibonacci

Ước chung lớn nhất

Tổng của hai số nguyên

Nhân hai số nguyên

Nâng một số nguyên lên một số nguyên

Tháp Hà Nội

đệ quy chính

In đảo ngược đầu vào bàn phím

Hình dung đệ quy

Hình 5.21 | Các ví dụ và bài tập về đệ quy trong văn bản. (Phần 1/2.)

Hãy kết thúc chương này với một số quan sát mà chúng tôi lặp đi lặp lại trong suốt quá trình sách. Công nghệ phần mềm tốt là quan trọng. Hiệu suất cao là quan trọng. Unfor- may mắn thay, những mục tiêu này thường mâu thuẫn với nhau. Công nghệ phần mềm tốt là chìa khóa để làm cho nhiệm vụ phát triển hệ thống phần mềm lớn hơn và phức tạp hơn trở nên dễ quản lý hơn. Team chúng tôi cần. Hiệu suất cao là chìa khóa để hiện thực hóa các hệ thống của tương lai sẽ đặt nhu cầu điện toán lớn hơn bao giờ hết trên phần cứng. Trường hợp các chức năng phù hợp ở đây?

Mẹo hiệu suất 5.2

Chức năng hóa các chương trình thúc đẩy công nghệ phần mềm tốt. Nhưng nó có giá. Một nặng- chương trình được chức năng hóa một cách khéo léo—so với chương trình nguyên khối (nghĩa là một phần) với- out functions—thực hiện một số lượng lớn các lời gọi hàm, và những lời gọi này tiêu tốn thời gian thực hiện trên (các) bộ xử lý của máy tính. Mặc dù các chương trình nguyên khối có thể thực hiện tốt hơn, chúng khó lập trình, kiểm tra, gỡ lỗi, bảo trì và phát triển hơn.

Mẹo hiệu suất 5.3

Kiến trúc phần cứng ngày nay được điều chỉnh để thực hiện các lời gọi hàm hiệu quả và ngày nay bộ xử lý phần cứng cực kỳ nhanh. Đối với phần lớn các ứng dụng và phần mềm hệ thống bạn sẽ xây dựng, tập trung vào công nghệ phần mềm tốt sẽ quan trọng hơn là lập trình cho hiệu suất cao. Tuy nhiên, trong nhiều ứng dụng C và hệ thống chủ đề, chẳng hạn như lập trình trò chơi, hệ thống thời gian thực, hệ điều hành và hệ thống nhúng Tuy nhiên, hiệu suất là rất quan trọng, vì vậy chúng tôi đưa các mẹo về hiệu suất xuyên suốt cuốn sách.

**5.17 Lập trình C an toàn**

Số ngẫu nhiên an toàn

Trong Phần 5.10, chúng tôi đã giới thiệu hàm rand để tạo số giả ngẫu nhiên. Thư viện chuẩn C không cung cấp trình tạo số ngẫu nhiên an toàn. Dựa theo mô tả chức năng của tài liệu tiêu chuẩn C rand, “Không có gì đảm bảo về chất lượng của chuỗi ngẫu nhiên được tạo ra và một số triển khai được biết là tạo ra các chuỗi với các bit có thứ tự thấp không ngẫu nhiên một cách đáng lo ngại.” Hướng dẫn CERT MSC30-C chỉ ra rằng phải sử dụng các chức năng tạo số ngẫu nhiên dành riêng cho triển khai để đảm bảo rằng các số ngẫu nhiên được tạo ra là không thể dự đoán được—điều này cực kỳ quan trọng tant, ví dụ, trong mật mã và các ứng dụng bảo mật khác. Hướng dẫn trình bày một số trình tạo số ngẫu nhiên dành riêng cho nền tảng được coi là an toàn. Đối với người cũ rộng rãi, Microsoft Windows cung cấp chức năng CryptGenRandom và hệ thống dựa trên POSIX tems (chẳng hạn như Linux) cung cấp chức năng ngẫu nhiên tạo ra kết quả an toàn hơn. để biết thêm thông tin, hãy xem hướng dẫn MSC30-C tại https://www.securecoding.cert.org. Nếu bạn xây dựng các ứng dụng cường độ công nghiệp yêu cầu số ngẫu nhiên, bạn nên đầu tư cổng cho (các) chức năng được khuyến nghị sử dụng trên nền tảng của bạn.

Bản tóm tắt

Mục 5.1 Giới thiệu

• Cách tốt nhất để phát triển và duy trì một chương trình lớn là chia nó thành nhiều chương trình nhỏ hơn mô-đun, mỗi mô-đun dễ quản lý hơn chương trình gốc. Các mô-đun được viết dưới dạng các hàm trong C.

Mục 5.2 Các mô-đun chương trình trong C

• Một chức năng được gọi bởi một cuộc gọi chức năng. Cuộc gọi hàm đề cập đến chức năng theo tên và

cung cấp thông tin (dưới dạng đối số) mà hàm được gọi cần để thực hiện nhiệm vụ của nó.

• Mục đích của việc che giấu thông tin là chỉ cung cấp cho các chức năng quyền truy cập vào thông tin họ cần để hoàn thành nhiệm vụ của họ. Đây là một phương tiện để thực hiện nguyên tắc đặc quyền tối thiểu, một trong những các nguyên tắc quan trọng nhất của công nghệ phần mềm tốt.

Mục 5.3 Hàm thư viện toán học

• Hàm os thường được gọi trong chương trình bằng cách viết tên hàm theo sau là dấu trái dấu ngoặc đơn theo sau là đối số (hoặc danh sách đối số được phân tách bằng dấu phẩy) của hàm theo sau bởi một dấu ngoặc đơn bên phải.

• Kiểu dữ liệu double là kiểu dấu phẩy động giống như float. Một biến kiểu double có thể lưu trữ một giá trị có cường độ và độ chính xác lớn hơn nhiều so với float có thể lưu trữ.

• Mỗi đối số của hàm có thể là một hằng, một biến hoặc một biểu thức.

Mục 5.4 Chức năng

• Biến cục bộ chỉ được biết trong định nghĩa hàm. Các chức năng khác không được phép biết tên của các biến cục bộ của hàm, cũng như không có hàm nào được phép biết cách triển khai chi tiết của bất kỳ chức năng nào khác.

Mục 5.5 Định nghĩa hàm

• Định dạng chung cho một định nghĩa hàm là tên hàm kiểu giá trị trả về ( danh sách tham số )

{

các định nghĩa

các câu lệnh

}

Loại giá trị trả về cho biết loại giá trị được trả về cho hàm gọi. Nếu một chức năng làm không trả về giá trị, kiểu giá trị trả về được khai báo là void. Tên hàm là bất kỳ định danh hợp lệ nào lửa hơn. Danh sách tham số là một danh sách được phân tách bằng dấu phẩy chứa các định nghĩa của các biến sẽ được chuyển đến chức năng. Nếu một hàm không nhận bất kỳ giá trị nào, danh sách tham số được khai báo là void. Thân hàm là tập hợp các định nghĩa và câu lệnh cấu thành hàm.

• Các đối số truyền vào hàm phải khớp về số, loại và thứ tự với các tham số trong định nghĩa hàm.

• Khi chương trình gặp lệnh gọi hàm, quyền điều khiển sẽ được chuyển từ điểm gọi hàm đến hàm được gọi, các câu lệnh của hàm được gọi được thực thi và điều khiển quay trở lại người gọi.

• Hàm được gọi có thể trả lại quyền điều khiển cho người gọi theo một trong ba cách. Nếu chức năng không trả về một giá trị, điều khiển được trả về khi đạt đến dấu ngoặc nhọn bên phải kết thúc chức năng hoặc bằng cách thực thi- đưa ra tuyên bố trở lại; Nếu hàm trả về một giá trị, câu lệnh trả về biểu thức; trả về giá trị của biểu thức.

Phần 5.6 Nguyên mẫu chức năng:

• Nguyên mẫu hàm khai báo kiểu trả về của hàm và khai báo số, loại và hoặc- der của các tham số mà chức năng mong muốn nhận được.

• Nguyên mẫu hàm cho phép trình biên dịch xác minh rằng các hàm được gọi chính xác.

• Trình biên dịch bỏ qua các tên biến được đề cập trong nguyên mẫu hàm.

• Các đối số trong biểu thức kiểu hỗn hợp được chuyển đổi thành cùng kiểu thông qua tiêu chuẩn C thông thường quy tắc chuyển đổi số học.

Mục 5.7 Gọi hàm Stack và Stack Frame

• Ngăn xếp được gọi là cấu trúc dữ liệu nhập trước, xuất trước (LIFO)—mục cuối cùng được đẩy (chèn) vào ngăn xếp là mục đầu tiên được bật (xóa) khỏi ngăn xếp.

• Hàm được gọi phải biết cách quay lại trình gọi của nó, vì vậy địa chỉ trả về của hàm gọi- được đẩy vào ngăn xếp thực thi chương trình khi hàm được gọi. Nếu một loạt chức năng- cuộc gọi xảy ra, các địa chỉ trả về liên tiếp được đẩy vào ngăn xếp theo kiểu nhập sau, xuất trước để chức năng cuối cùng được thực thi sẽ là chức năng đầu tiên quay trở lại trình gọi của nó.

• Ngăn xếp thực thi chương trình chứa bộ nhớ cho các biến cục bộ được sử dụng trong mỗi lần gọi của một chức năng trong quá trình thực hiện chương trình. Dữ liệu này được gọi là khung ngăn xếp của hàm gọi. Khi một lời gọi hàm được thực hiện, khung ngăn xếp cho lời gọi hàm đó được đẩy lên pro- ngăn xếp thực thi gram. Khi chức năng trở lại trình gọi của nó, khung ngăn xếp cho chức năng này cuộc gọi được bật ra khỏi ngăn xếp và các biến cục bộ đó không còn được biết đến trong chương trình.

• Dung lượng bộ nhớ trong máy tính là hữu hạn nên chỉ sử dụng được một lượng bộ nhớ nhất định để lưu trữ các khung ngăn xếp trên ngăn xếp thực thi chương trình. Nếu có nhiều cuộc gọi chức năng hơn có thể có các khung ngăn xếp của chúng được lưu trữ trên ngăn xếp thực thi chương trình, một lỗi được gọi là tràn ngăn xếp xảy ra. Ứng dụng sẽ biên dịch chính xác, nhưng quá trình thực thi của nó gây ra lỗi tràn ngăn xếp.

Mục 5.8 Tiêu đề

• Mỗi thư viện chuẩn có một tiêu đề tương ứng chứa các nguyên mẫu hàm cho tất cả các các hàm trong thư viện đó, cũng như các định nghĩa về các hằng ký hiệu khác nhau cần thiết bởi những chức năng.

• Bạn có thể tạo và bao gồm các tiêu đề của riêng bạn.

Mục 5.9 Truyền đối số theo giá trị và theo tham chiếu

• Khi một đối số được truyền theo giá trị, một bản sao giá trị của biến được tạo và bản sao đó được chuyển đến chức năng được gọi. Các thay đổi đối với bản sao trong chức năng được gọi không ảnh hưởng đến bản gốc giá trị của biến.

• Tất cả các lời gọi trong C đều là lời gọi theo giá trị.

• Có thể mô phỏng gọi theo tham chiếu bằng cách sử dụng toán tử địa chỉ và toán tử gián tiếp.

Mục 5.10 Tạo số ngẫu nhiên

• Hàm rand tạo ra một số nguyên từ 0 đến RAND\_MAX được xác định theo tiêu chuẩn C ít nhất là 32767.

• Nguyên mẫu hàm cho rand và srand được chứa trong <stdlib.h>.

• Các giá trị do rand tạo ra có thể được thu nhỏ và thay đổi để tạo ra các giá trị trong một phạm vi cụ thể.

• Để ngẫu nhiên hóa một chương trình, hãy sử dụng hàm thư viện chuẩn C srand.

• Lệnh gọi hàm srand thường chỉ được chèn vào chương trình sau khi nó đã được loại bỏ hoàn toàn. Trong khi gỡ lỗi, tốt hơn là bỏ qua srand. Điều này đảm bảo tính lặp lại, điều cần thiết để chứng minh rằng các hiệu chỉnh đối với chương trình tạo số ngẫu nhiên hoạt động bình thường.

• Để chọn ngẫu nhiên mà không cần nhập seed mỗi lần, chúng ta sử dụng srand(time(NULL)).

• Phương trình tổng quát để chia tỷ lệ và dịch chuyển một số ngẫu nhiên là trong đó a là giá trị dịch chuyển (nghĩa là số đầu tiên trong dãy số nguyên liên tiếp mong muốn) và b là hệ số tỷ lệ (nghĩa là chiều rộng của phạm vi số nguyên liên tiếp mong muốn).

n = a + rand() % b;

Mục 5.11 Ví dụ: Trò chơi may rủi

• Một phép liệt kê, được giới thiệu bởi từ khóa enum, là một tập hợp các hằng số nguyên được biểu diễn bởi định danh. Các giá trị trong một enum bắt đầu bằng 0 và được tăng thêm 1. Cũng có thể gán một giá trị số nguyên cho mỗi mã định danh trong một enum. Các định danh trong một bảng liệt kê phải là duy nhất, nhưng các giá trị có thể được nhân đôi.

Mục 5.12 Các lớp lưu trữ

• Mỗi mã định danh trong chương trình có thuộc tính lớp lưu trữ, thời lượng lưu trữ, phạm vi và liên kết.

• C cung cấp bốn lớp lưu trữ được chỉ định bởi các chỉ định lớp lưu trữ: auto, register, extern và tĩnh.

• Thời lượng lưu trữ của mã định danh là khi mã định danh đó tồn tại trong bộ nhớ.

Mục 5.13 Quy tắc phạm vi

• Phạm vi của mã định danh là nơi mã định danh có thể được tham chiếu trong một chương trình.

• Liên kết của mã định danh xác định đối với chương trình nhiều tệp nguồn liệu mã định danh có phải là chỉ được biết trong tệp nguồn hiện tại hoặc trong bất kỳ tệp nguồn nào có khai báo phù hợp.

• Các biến có thời lượng lưu trữ tự động được tạo khi khối mà chúng được xác định là được nhập, tồn tại trong khi khối đang hoạt động và bị hủy khi khối được thoát. một chức năng các biến cục bộ thường có thời lượng lưu trữ tự động.

• Từ khóa extern và static dùng để khai báo định danh cho các biến và hàm của static thời hạn lưu trữ.

• Các biến thời lượng lưu trữ tĩnh được phân bổ và khởi tạo một lần, trước khi chương trình bắt đầu ex-

hành quyết.

• Có hai loại mã định danh có thời lượng lưu trữ tĩnh: mã định danh bên ngoài (chẳng hạn như toàn cầu

biến và tên hàm) và các biến cục bộ được khai báo với tĩnh chỉ định lớp lưu trữ.

• Biến toàn cục được tạo bằng cách đặt các định nghĩa biến bên ngoài bất kỳ định nghĩa hàm nào. Quả cầu- biến al giữ nguyên giá trị của chúng trong suốt quá trình thực hiện chương trình.

• Các biến tĩnh cục bộ giữ lại giá trị của chúng giữa các lần gọi đến hàm mà chúng được xác định.

• Tất cả các biến số của thời lượng lưu trữ tĩnh được khởi tạo bằng 0 nếu bạn không khởi tạo rõ ràng.

hóa chúng.

• Một mã định danh có thể có phạm vi chức năng, phạm vi tệp, phạm vi khối hoặc phạm vi nguyên mẫu chức năng.

• Nhãn là định danh duy nhất có phạm vi chức năng. Nhãn có thể được sử dụng ở bất cứ đâu trong chức năng trong đó chúng xuất hiện nhưng không thể được tham chiếu bên ngoài thân hàm.

• Một mã định danh được khai báo bên ngoài bất kỳ chức năng nào có phạm vi tệp. Một định danh như vậy là "đã biết" trong tất cả chức năng từ điểm mà nó được khai báo cho đến khi kết thúc tệp.

• Định danh được xác định bên trong một khối có phạm vi khối. Phạm vi khối kết thúc ở bên phải kết thúc

dấu ngoặc nhọn (}) của khối.

• Các biến cục bộ được xác định ở đầu hàm có phạm vi khối, cũng như tham số hàm, được hàm coi là biến cục bộ.

• Bất kỳ khối nào cũng có thể chứa các định nghĩa biến. Khi các khối được lồng vào nhau và một mã định danh ở bên ngoài khối có cùng tên với mã định danh ở khối bên trong, mã định danh ở khối bên ngoài là "ẩn" cho đến khi khối bên trong kết thúc.

• Các định danh duy nhất có phạm vi chức năng-nguyên mẫu là những định danh được sử dụng trong danh sách tham số của chức năng- nguyên mẫu. Các mã định danh được sử dụng trong nguyên mẫu hàm có thể được sử dụng lại ở nơi khác trong chương trình không mơ hồ.

Mục 5.14 Đệ quy

• Hàm đệ quy là hàm gọi chính nó một cách trực tiếp hoặc gián tiếp.

• Nếu một hàm đệ quy được gọi với trường hợp cơ bản, thì hàm này chỉ trả về một kết quả. Nếu nó được gọi là với một vấn đề phức tạp hơn, chức năng chia vấn đề thành hai phần khái niệm: phần mà hàm biết cách thực hiện và một phiên bản nhỏ hơn một chút của vấn đề ban đầu.

Bởi vì vấn đề mới này trông giống như vấn đề ban đầu, hàm sẽ khởi chạy một cuộc gọi đệ quy để giải bài toán nhỏ hơn.

• Để đệ quy kết thúc, mỗi lần hàm đệ quy gọi chính nó với một phiên bản đơn giản hơn một chút của bài toán ban đầu, chuỗi các bài toán nhỏ hơn và nhỏ hơn phải hội tụ về trường hợp cơ sở. Khi chức năng nhận dạng trường hợp cơ sở, kết quả được trả về chức năng trước đó- lệnh gọi, và một chuỗi trả về xảy ra sau đó cho đến khi lệnh gọi ban đầu của chức năng cuối cùng trả về kết quả cuối cùng.

• Tiêu chuẩn C không chỉ định thứ tự các toán hạng của hầu hết các toán tử (bao gồm +) để được đánh giá. Trong số nhiều toán tử của C, tiêu chuẩn chỉ định thứ tự đánh giá của op- số lượng chỉ các toán tử &&, ||, toán tử dấu phẩy (,) và ?:. Ba cái đầu tiên trong số này là toán tử nhị phân có hai toán hạng được đánh giá từ trái sang phải. Toán tử cuối cùng là thuật ngữ duy nhất của C- nhà điều hành nary. Toán hạng ngoài cùng bên trái của nó được đánh giá đầu tiên; nếu nó đánh giá khác không, toán tử ở giữa và được đánh giá tiếp theo và toán hạng cuối cùng bị bỏ qua; nếu toán hạng ngoài cùng bên trái đánh giá bằng 0, toán hạng thứ ba được đánh giá tiếp theo và toán hạng ở giữa bị bỏ qua.

Mục 5.16 Đệ quy so với Lặp lại

• Cả lặp và đệ quy đều dựa trên cấu trúc điều khiển: Phép lặp sử dụng cấu trúc lặp; đệ quy sử dụng một cấu trúc lựa chọn.

• Cả phép lặp và đệ quy đều liên quan đến phép lặp: Phép lặp rõ ràng sử dụng cấu trúc lặp; nốt Rê- Cursion đạt được sự lặp lại thông qua các cuộc gọi chức năng lặp đi lặp lại.

• Mỗi phép lặp và đệ quy đều liên quan đến một bài kiểm tra kết thúc: Phép lặp kết thúc khi vòng lặp-con-

điều kiện gia hạn không thành công; đệ quy chấm dứt khi một trường hợp cơ sở được công nhận.

• Phép lặp và phép đệ quy có thể xảy ra vô tận: Một vòng lặp vô hạn xảy ra với phép lặp nếu vòng lặp- kiểm tra tiếp tục không bao giờ trở thành sai; đệ quy vô hạn xảy ra nếu bước đệ quy không lặp lại giải quyết vấn đề theo cách hội tụ trên trường hợp cơ sở.

• Đệ quy liên tục gọi cơ chế, và do đó, chi phí hoạt động của các lời gọi hàm. Điều này có thể tốn kém cả về thời gian xử lý và không gian bộ nhớ.

Bài Tập Tự Ôn Tập

5.1 Trả lời từng câu hỏi sau:

a) Mô-đun chương trình trong C được gọi là a(n) .

b) Một chức năng được gọi với a(n) .

c) Một biến chỉ được biết trong hàm mà nó được xác định được gọi là a(n)

d) Câu lệnh trong hàm được gọi dùng để truyền giá trị của biểu thức trở lại chức năng gọi.

e) Từ khóa được sử dụng trong tiêu đề chức năng để chỉ ra rằng một chức năng không biến một giá trị hoặc để chỉ ra rằng một chức năng không chứa tham số.

f) Phần của mã định danh là phần của chương trình mà mã định danh có thể được dùng.

g) Ba cách để trả lại quyền điều khiển từ hàm

h) A(n) cho phép trình biên dịch kiểm tra số lượng, kiểu và thứ tự của đối số ments được truyền cho một chức năng.

i) Hàm được sử dụng để tạo số ngẫu nhiên.

j) Hàm này được sử dụng để đặt hạt giống số ngẫu nhiên để tạo ngẫu nhiên một chương trình.

k) Các chỉ định lớp lưu trữ là , và .

l) Các biến được khai báo trong khối hoặc trong danh sách tham số của hàm được giả định là của lớp lưu trữ trừ khi có quy định khác.

m) Một biến không tĩnh được xác định bên ngoài bất kỳ khối hoặc chức năng nào là một (n) biến.

n) Để một biến cục bộ trong một hàm giữ lại giá trị của nó giữa các lần gọi hàm, nó phải được khai báo với bộ xác định lớp lưu trữ.

o) Bốn phạm vi có thể có của một mã định danh.

p) Một hàm gọi chính nó trực tiếp hoặc gián tiếp là hàm (n).

q) Một hàm đệ quy thường có hai thành phần: một thành phần cung cấp phương tiện cho đệ quy chấm dứt bằng cách kiểm tra một (n) trường hợp và một trường hợp thể hiện vấn đề dưới dạng một cuộc gọi đệ quy cho một vấn đề đơn giản hơn một chút so với cuộc gọi ban đầu.

5.2 Đối với chương trình sau, hãy nêu phạm vi (phạm vi chức năng, phạm vi tệp, phạm vi khối hoặc phạm vi chức năng-nguyên mẫu) của từng thành phần sau.

a) Biến x trong hàm main.

b) Biến y lập phương.

c) Hàm lập phương.

d) Chức năng chính.

e) Nguyên mẫu hàm cho khối lập phương.

f) Định danh y trong nguyên mẫu hàm cho khối lập phương.

5.3 Viết chương trình kiểm tra xem các ví dụ về lệnh gọi hàm thư viện toán học có hiển thị không trong Hình 5.2 thực sự tạo ra các kết quả được chỉ ra.

5.4 Đặt tiêu đề chức năng cho mỗi chức năng sau.

a) Cạnh huyền của hàm nhận hai đối số dấu phẩy động có độ chính xác kép, side1 và side2, đồng thời trả về kết quả dấu phẩy động có độ chính xác kép.

b) Hàm nhỏ nhất nhận vào ba số nguyên x, y, z và trả về một số nguyên.

c) Các hướng dẫn hàm không nhận bất kỳ đối số nào và không trả về giá trị- uê. [Lưu ý: Các chức năng như vậy thường được sử dụng để hiển thị hướng dẫn cho người dùng.]

d) Hàm intToFloat nhận vào một đối số nguyên, số và trả về một dấu phẩy động kết quả điểm.

5.5 Cho nguyên mẫu hàm cho mỗi trường hợp sau:

a) Hàm được mô tả trong Bài tập 5.4(a).

b) Hàm được mô tả trong Bài tập 5.4(b).

c) Hàm mô tả trong Bài tập 5.4(c).

d) Hàm được mô tả trong Bài tập 5.4(d).

5.6 Viết khai báo cho biến số thực lastVal để giữ lại giá trị của nó giữa các lần gọi đến chức năng mà nó được xác định.

5.7 Tìm lỗi trong mỗi đoạn chương trình sau và giải thích lỗi có thể xảy ra như thế nào đã sửa (xem thêm

Đáp Án Bài Tập Tự Ôn Tập

5.1 a) chức năng. b) lời gọi hàm. c) biến cục bộ. đ) trở về. đ) vô hiệu. f) Phạm vi. g) trả lại; hoặc

trả về biểu thức; hoặc gặp phải dấu ngoặc nhọn bên phải của hàm. h) nguyên mẫu chức năng.

i) rand. j) srand. k) tự động, đăng ký, bên ngoài, tĩnh. l) ô tô. m) bên ngoài, toàn cầu. n) tĩnh.

o) phạm vi chức năng, phạm vi tệp, phạm vi khối, phạm vi nguyên mẫu chức năng. p) đệ quy. q) căn.

5.2 a) Phạm vi khối. b) Phạm vi khối. c) Phạm vi tệp. d) Phạm vi tệp. e) Phạm vi tệp. f) Chức năng-pro- phạm vi totype.

5.3 Xem bên dưới. [Lưu ý: Trên hầu hết các hệ thống Linux, bạn phải sử dụng tùy chọn -lm khi biên dịch chương trình này.]

5.4 a) double hypotenuse( double side1, double side2 )

b) int smallest( int x, int y, int z )

c) void instructions( void )

d) float intToFloat( int number )

5.5 a) double hypotenuse( double side1, double side2 );

b) int smallest( int x, int y, int z );

c) void instructions( void );

d) float intToFloat( int number );

5.6 static float lastVal;

5.7 a) Lỗi: Hàm h được định nghĩa trong hàm g.

Sửa lại: Chuyển định nghĩa của h ra khỏi định nghĩa của g.

b) Lỗi: Phần thân của hàm được cho là trả về một số nguyên, nhưng không.

Sửa lỗi: Xóa biến result và đặt câu lệnh sau vào hàm:

trả lại x + y;

c) Lỗi: Dấu chấm phẩy sau dấu ngoặc đơn bên phải bao quanh danh sách tham số và lại xác định tham số a trong định nghĩa hàm.

Sửa chữa: Xóa dấu chấm phẩy sau dấu ngoặc đơn bên phải của danh sách tham số và xóa khai báo float a; trong thân hàm.

d) Lỗi: Không trả về kết quả của n + sum( n - 1 ); tổng trả về một kết quả không chính xác.

Sửa lỗi: Viết lại câu lệnh trong mệnh đề khác như trả về n + tổng( n - 1 );

e) Lỗi: Hàm trả về một giá trị khi nó không được yêu cầu. Khắc phục: Loại bỏ câu lệnh return.

bài tập

5.8 Cho biết giá trị của x sau khi thực hiện mỗi câu lệnh sau:

a) x = fabs( 7,5 );

b) x = sàn( 7,5 );

c) x = fabs( 0,0 );

d) x = trần( 0,0 );

e) x = fabs( -6,4 );

f) x = trần( -6,4 );

g) x = trần( -fabs( -8 + sàn( -5,5 ) ) );

5.9 (Phí Đậu Xe) Nhà để xe tính phí tối thiểu $2.00 để đậu xe cho tối đa ba giờ và thêm $0,50 mỗi giờ cho mỗi giờ hoặc một phần của giờ đó trong hơn ba giờ. Giá trị lớn nhất phí cho bất kỳ khoảng thời gian 24 giờ nhất định nào là $10,00. Giả sử rằng không có bãi đỗ xe nào lâu hơn 24 giờ tại một thời điểm. Viết một chương trình sẽ tính toán và in các khoản phí đỗ xe cho mỗi trong số ba khách hàng những người đã đỗ xe trong nhà để xe này ngày hôm qua. Bạn nên nhập số giờ đỗ cho mỗi khách hàng. Chương trình của bạn sẽ in kết quả ở định dạng bảng và sẽ tính toán và in tổng số tiền nhận được của ngày hôm qua. Chương trình nên sử dụng chức năng tính phí để xác định- khai thác phí cho mỗi khách hàng. Đầu ra của bạn sẽ xuất hiện ở định dạng sau:

Car Hours Charge

1 1.5 2.00

2 4.0 2.50

3 24.0 10.00

TOTAL 29.5 14.50

5.10 (Làm tròn số) Một ứng dụng của hàm sàn đang làm tròn một giá trị đến giá trị gần nhất số nguyên. tuyên bố: y = sàn( x + .5 ); sẽ làm tròn số x đến số nguyên gần nhất và gán kết quả cho y. Viết chương trình đọc một vài số và sử dụng câu lệnh trước để làm tròn từng số này đến số gần nhất số nguyên. Đối với mỗi số được xử lý, hãy in cả số ban đầu và số đã được làm tròn.

5.11 (Làm tròn số) Hàm sàn có thể được sử dụng để làm tròn một số thành một số thập phân cụ thể địa điểm. tuyên bố: y = sàn( x \* 10 + .5 ) / 10; làm tròn x đến vị trí phần mười (vị trí đầu tiên bên phải dấu thập phân). tuyên bố: y = sàn( x \* 100 + .5 ) / 100; làm tròn x đến vị trí phần trăm (vị trí thứ hai bên phải dấu thập phân). Viết một chương trình định nghĩa bốn hàm để làm tròn số x theo nhiều cách khác nhau

a) roundToInteger(số)

b) roundToTenths(số)

c) roundToHundreth (số)

d) RoundToThousandths(số)

Đối với mỗi giá trị được đọc, chương trình của bạn sẽ in ra giá trị ban đầu, số được làm tròn thành số nguyên gần nhất, số được làm tròn đến hàng chục gần nhất, số được làm tròn đến hàng trăm gần nhất dredth và số được làm tròn đến phần nghìn gần nhất.

5.12 Trả lời từng câu hỏi sau.

a) Chọn số “ngẫu nhiên” nghĩa là gì?

b) Tại sao hàm rand hữu ích để mô phỏng các trò chơi may rủi?

c) Tại sao bạn chọn ngẫu nhiên một chương trình bằng cách sử dụng srand? Trong hoàn cảnh nào vậy mong muốn không ngẫu nhiên?

d) Tại sao thường cần phải chia tỷ lệ và/hoặc thay đổi các giá trị do rand tạo ra?

5.13 Viết câu lệnh gán các số nguyên ngẫu nhiên cho biến n trong các khoảng sau:

a) 1 ≤ n ≤ 2

b) 1 ≤ n ≤ 100

c) 0 ≤ n ≤ 9

d) 1000 ≤ n ≤ 1112

e) –1 ≤ n ≤ 1

f) –3 ≤ n ≤ 11

5.14 Với mỗi tập hợp số nguyên sau, hãy viết một câu lệnh để in ra một số ngẫu nhiên từ tập hợp.

a) 2, 4, 6, 8, 10.

b) 3, 5, 7, 9, 11.

c) 6, 10, 14, 18, 22.

5.15 (Tính toán cạnh huyền) Định nghĩa một hàm gọi là cạnh huyền để tính độ dài của cạnh huyền của một tam giác vuông khi cho trước hai cạnh góc vuông. Các chức năng nên mất hai đối số kiểu double và trả về cạnh huyền là double. Kiểm tra chương trình của bạn với các giá trị bên được chỉ định trong Hình 5.22.

5.16 (Lũy thừa) Viết hàm integerPower(cơ số, số mũ) trả về giá trị của số mũ cơ sở.

Ví dụ: integerPower( 3, 4 ) = 3 \* 3 \* 3 \* 3. Giả sử rằng số mũ là một số dương, khác không số nguyên và cơ số là số nguyên. Hàm integerPower nên sử dụng để kiểm soát phép tính. Không sử dụng bất kỳ hàm thư viện toán học nào.

5.17 (Bội số) Viết một hàm bội số xác định cho một cặp số nguyên liệu sec- số nguyên ond là bội số của số nguyên đầu tiên. Hàm sẽ nhận hai đối số nguyên và trả về 1 (đúng) nếu số thứ hai là bội số của số thứ nhất và 0 (sai) nếu ngược lại. Sử dụng chức năng này trong một chương trình nhập vào một loạt các cặp số nguyên.

5.18 (Chẵn hoặc Lẻ) Viết chương trình nhập vào một dãy số nguyên và truyền lần lượt từng số nguyên để hoạt động chẵn, sử dụng toán tử phần dư để xác định xem một số nguyên có chẵn hay không. Các hàm sẽ lấy một đối số là số nguyên và trả về 1 nếu số nguyên đó là số chẵn và 0 nếu ngược lại.

5.19 (Hình vuông dấu sao) Viết hàm hiển thị một hình vuông liền các dấu hoa thị có cạnh là được chỉ định ở phía tham số nguyên. Ví dụ: nếu bên là 4, hàm sẽ hiển thị:

5.20 (Hiển thị hình vuông của bất kỳ ký tự nào) Sửa đổi hàm được tạo trong Bài tập 5.19 thành tạo thành hình vuông từ bất kỳ ký tự nào được chứa trong tham số ký tự fill Character. Như vậy nếu bên là 5 và fill Character là “#”, thì hàm này sẽ in ra:

5.21 (Dự án: Vẽ hình dạng bằng ký tự) Sử dụng các kỹ thuật tương tự như những kỹ thuật được phát triển trong Bài tập 5.19–5.20 để tạo một chương trình vẽ đồ thị cho nhiều hình dạng.

5.22 (Tách các chữ số) Viết các đoạn chương trình thực hiện từng việc sau:

a) Tính phần nguyên của thương khi chia số nguyên a cho số b.

b) Tính số dư khi chia số a cho số b.

c) Sử dụng các đoạn chương trình được phát triển ở a) và b) để viết một hàm nhập vào một số nguyên từ 1 đến 32767 và in ra dưới dạng một chuỗi các chữ số, mỗi chữ số có hai khoảng trắng.

Ví dụ: số nguyên 4562 sẽ được in dưới dạng:

5.23 (Thời gian tính bằng giây) Viết một hàm lấy thời gian làm ba đối số nguyên (đối với giờ, phút và giây) và trả về số giây kể từ lần cuối đồng hồ “đánh 12.” Sử dụng hàm này để tính lượng thời gian tính bằng giây giữa hai lần, cả hai lần nằm trong một chu kỳ 12 giờ của đồng hồ.

5.24 (Chuyển đổi nhiệt độ) Thực hiện các hàm số nguyên sau:

a) Hàm celsius trả về độ C tương đương với nhiệt độ độ F.

b) Hàm fahrenheit trả về độ F tương đương với nhiệt độ độ C.

c) Sử dụng các hàm này để viết chương trình in biểu đồ hiển thị độ tương đương Fahrenheit alents của tất cả nhiệt độ C từ 0 đến 100 độ, và tương đương với C của tất cả nhiệt độ Fahrenheit từ 32 đến 212 độ. In kết quả đầu ra trong một bảng cho-mat giúp giảm thiểu số lượng dòng đầu ra trong khi vẫn có thể đọc được.

5.25 (Tìm giá trị nhỏ nhất) Viết hàm trả về giá trị nhỏ nhất trong ba giá trị dấu phẩy động con số.

5.26 (Số hoàn hảo) Một số nguyên được gọi là số hoàn hảo nếu các thừa số của nó bao gồm 1 (nhưng không phải chính số đó), tính tổng bằng số. Ví dụ: 6 là số hoàn hảo vì 6 = 1 + 2 + 3. Viết hàm perfect xác định xem số tham số có phải là số hoàn hảo hay không. Sử dụng chức năng này trong một chương trình xác định và in tất cả các số hoàn hảo từ 1 đến 1000. In ra thừa số của mỗi số hoàn hảo để xác nhận rằng số đó thực sự là số hoàn hảo. Chal- tăng sức mạnh cho máy tính của bạn bằng cách kiểm tra các số lớn hơn 1000.

5.27 (Số nguyên tố) Một số nguyên được gọi là số nguyên tố nếu nó chỉ chia hết cho 1 và chính nó. Đối với người cũ ample, 2, 3, 5 và 7 là số nguyên tố, nhưng 4, 6, 8 và 9 thì không.

a) Viết hàm xác định một số có phải là số nguyên tố hay không.

b) Sử dụng chức năng này trong một chương trình xác định và in tất cả các số nguyên tố được- giữa 1 và 10.000. Bạn thực sự phải kiểm tra bao nhiêu trong số 10.000 số này? trước khi chắc chắn rằng bạn đã tìm thấy tất cả các số nguyên tố?

c) Ban đầu, bạn có thể nghĩ rằng n/2 là giới hạn trên mà bạn phải kiểm tra để xem cho dù một số có phải là số nguyên tố hay không, nhưng bạn chỉ cần tăng đến căn bậc hai của n. Nốt Rê- viết chương trình và chạy nó theo cả hai cách. Ước tính cải thiện hiệu suất.

5.28 (Đảo ngược chữ số) Viết hàm nhận vào một giá trị số nguyên và trả về số có chữ số của nó đảo ngược. Ví dụ: cho số 7631, hàm sẽ trả về 1367.

5.29 (Ước chung lớn nhất) Ước chung lớn nhất (GCD) của hai số nguyên là lớn số nguyên chia hết cho hai số. Viết hàm gcd trả về giá trị lớn nhất ước chung của hai số nguyên.

5.30 (Điểm Chất lượng cho Điểm của Học sinh) Viết một hàm qualityPoints để nhập một stu- trung bình của dent và trả về 4 nếu là 90–100, 3 nếu là 80–89, 2 nếu là 70–79, 1 nếu là 60–69 và 0 nếu là trung bình thấp hơn 60.

5.31 (Tung đồng xu) Viết chương trình mô phỏng việc tung đồng xu. Đối với mỗi lần tung đồng xu, chương trình nên in Heads hoặc Tails. Hãy để chương trình tung đồng xu 100 lần và đếm số ber số lần mỗi mặt của đồng xu xuất hiện. In kết quả. Chương trình nên gọi một chức năng riêng- lật ngược không có đối số và trả về 0 cho mặt sấp và 1 cho mặt ngửa. [Lưu ý: Nếu chương trình mô phỏng thực tế việc tung đồng xu, sau đó mỗi mặt của đồng xu sẽ xuất hiện khoảng một nửa thời gian cho tổng số xấp xỉ 50 mặt ngửa và 50 mặt sấp.]

5.32 (Đoán số) Viết chương trình C chơi trò chơi “đoán số” như sau mức thấp: Chương trình của bạn chọn số cần đoán bằng cách chọn ngẫu nhiên một số nguyên trong phạm vi 1 đến 1000. Sau đó chương trình gõ:

Sau đó, người chơi nhập một dự đoán đầu tiên. Chương trình trả lời với một trong những điều sau đây:

Tôi có một số từ 1 đến 1000.

Bạn có thể đoán số của tôi?

Vui lòng nhập dự đoán đầu tiên của bạn.

1. Xuất sắc! Bạn đoán số!

Bạn có muốn chơi lại không (y hay n)?

2. Quá thấp. Thử lại.

3. Quá cao. Thử lại.

Nếu dự đoán của người chơi không chính xác, chương trình của bạn sẽ lặp lại cho đến khi người chơi cuối cùng nhận được số Phải. Chương trình của bạn phải liên tục thông báo cho người chơi Quá cao hoặc Quá thấp để giúp người chơi “không tham gia” trên câu trả lời đúng. [Lưu ý: Kỹ thuật tìm kiếm được sử dụng trong vấn đề này được gọi là nhị phân tìm kiếm. Chúng tôi sẽ nói thêm về điều này trong vấn đề tiếp theo.]

5.33 (Đoán số Sửa đổi) Sửa đổi chương trình của Bài tập 5.32 để đếm số ber đoán người chơi thực hiện. Nếu số là 10 hoặc ít hơn, in ra Hoặc bạn biết bí mật hoặc bạn đã may mắn! Nếu người chơi đoán số sau 10 lần thử, thì in ra chữ Ahah! Bạn biết bí mật! Nếu người chơi đoán nhiều hơn 10 lần, thì hãy in Bạn sẽ có thể đoán tốt hơn! Tại sao nên phải mất không quá 10 lần đoán? Chà, với mỗi lần "đoán tốt" người chơi sẽ có thể loại bỏ một nửa số. Bây giờ hãy cho biết tại sao có thể đoán bất kỳ số nào từ 1 đến 1000 sau 10 lần thử hoặc ít hơn.

5.34 (Lũy thừa đệ quy) Viết hàm đệ quy lũy thừa(cơ số, số mũ) sao cho khi gọi trả lại số mũ cơ sở

Ví dụ: luỹ thừa( 3, 4 ) = 3 \* 3 \* 3 \* 3. Giả sử số mũ là một số nguyên lớn hơn hoặc bằng đến 1. Gợi ý: Bước đệ quy sẽ sử dụng mối quan hệ lũy thừa = cơ sở \* lũy thừa cơ sở–1 và điều kiện kết thúc xảy ra khi số mũ bằng 1 vì cơ sở1 = cơ sở

5.35 (Fibonacci) Dãy Fibonacci 0, 1, 1, 2, 3, 5, 8, 13, 21, ... bắt đầu bằng các số hạng 0 và 1 và có tính chất là mỗi số hạng tiếp theo là tổng của hai các điều khoản trước đó. a) Viết hàm không đệ quy fibonacci(n) tính Fibonacci thứ n con số. Sử dụng unsigned int cho tham số của hàm và unsigned long long int cho hàm trả về kiểu. b) Xác định số Fibonacci lớn nhất có thể được in trên hệ thống của bạn.

5.36 (Tháp Hà Nội) Mọi nhà khoa học máy tính mới vào nghề đều phải vật lộn với một số tác phẩm kinh điển nhất định. Vấn đề, và Tháp Hà Nội (xem Hình 5.23) là một trong những vấn đề nổi tiếng nhất trong số này. Truyền thuyết có kể rằng trong một ngôi đền ở Viễn Đông, các linh mục đang cố gắng di chuyển một chồng đĩa từ cọc này sang cọc khác. Ngăn xếp ban đầu có 64 đĩa được luồn vào một chốt và được sắp xếp từ dưới lên trên bằng kích thước giảm dần. Các linh mục đang cố gắng di chuyển ngăn xếp từ chốt này sang chốt thứ hai bên dưới các ràng buộc rằng chính xác một đĩa được di chuyển tại một thời điểm và không bao giờ có thể đặt một đĩa lớn hơn phía trên một đĩa nhỏ hơn. Một chốt thứ ba có sẵn để giữ tạm thời các đĩa. Giả sử thế giới sẽ kết thúc khi các linh mục hoàn thành nhiệm vụ của họ, vì vậy có rất ít động lực để chúng tôi tạo điều kiện cho nỗ lực của họ pháo đài. Giả sử rằng các linh mục đang cố gắng di chuyển các đĩa từ chốt 1 sang chốt 3. Chúng tôi muốn phát triển một thuật toán sẽ in trình tự chính xác của các lần chuyển chốt từ đĩa này sang đĩa khác. Nếu chúng ta tiếp cận vấn đề này bằng các phương pháp thông thường, chúng ta sẽ nhanh chóng thấy mình vô vọng thắt nút trong việc quản lý các đĩa. Thay vào đó, nếu chúng ta tấn công vấn đề bằng đệ quy trong tâm trí, nó ngay lập tức trở nên dễ điều khiển. Di chuyển n đĩa chỉ có thể được xem xét về mặt di chuyển n – 1 đĩa (và do đó là đệ quy) như sau:

a) Di chuyển n – 1 đĩa từ chốt 1 sang chốt 2, sử dụng chốt 3 làm khu vực giữ tạm thời.

b) Di chuyển đĩa cuối cùng (lớn nhất) từ chốt 1 sang chốt 3.

c) Di chuyển n – 1 đĩa từ chốt 2 sang chốt 3, sử dụng chốt 1 làm khu vực giữ tạm thời.

Quá trình kết thúc khi nhiệm vụ cuối cùng liên quan đến việc di chuyển n = 1 đĩa, tức là trường hợp cơ sở. Đây là được thực hiện bằng cách di chuyển đĩa một cách tầm thường mà không cần khu vực giữ tạm thời. Viết chương trình giải bài toán Tháp Hà Nội. Sử dụng hàm đệ quy với bốn thông số:

a) Số đĩa cần di chuyển

b) Cái chốt mà các đĩa này ban đầu được luồn vào

c) Cái chốt mà chồng đĩa này sẽ được di chuyển đến

d) Chốt được sử dụng làm khu vực giữ tạm thời

Chương trình của bạn sẽ in các hướng dẫn chính xác cần thiết để di chuyển các đĩa từ chốt bắt đầu đến chốt đích. Ví dụ: để di chuyển một chồng ba đĩa từ cọc 1 sang cọc 3, chương trình của bạn sẽ in một loạt các nước đi sau:

1 → 3 (Điều này có nghĩa là di chuyển một đĩa từ chốt 1 sang chốt 3.)

1 → 2

3 → 2

1 → 3

2 → 1

2 → 3

1 → 3

5.37 (Towers of Hanoi: Iterative Solution) Bất kỳ chương trình nào có thể thực hiện đệ quy rõ ràng ít rõ ràng hơn. Hãy thử viết một phiên bản lặp lại của Tháp Hà Nội. Nếu bạn thành công, hãy so sánh phiên bản lặp của bạn với phiên bản đệ quy mà bạn đã phát triển trong Bài tập 5.36. điều tra các vấn đề về hiệu suất, sự rõ ràng và khả năng của bạn để chứng minh tính đúng đắn của các chương trình.

5.38 (Trực quan hóa đệ quy) Thật thú vị khi xem đệ quy “hoạt động”. Sửa đổi giai thừa của Hình 5.18 để in biến cục bộ và tham số gọi đệ quy của nó. Đối với mỗi cuộc gọi đệ quy, hiển thị kết quả đầu ra trên một dòng riêng biệt và thêm mức độ thụt lề. Làm hết sức mình để làm cho đầu ra rõ ràng, thú vị và có ý nghĩa. Mục tiêu của bạn ở đây là thiết kế và triển khai đầu ra cho- mat giúp một người hiểu rõ hơn về đệ quy. Bạn có thể muốn thêm các khả năng hiển thị như vậy đến nhiều ví dụ và bài tập đệ quy khác trong suốt văn bản.

5.39 (Ước chung lớn nhất đệ quy) Ước chung lớn nhất của các số nguyên x và y là số nguyên lớn nhất chia hết cho cả x và y. Viết hàm đệ quy gcd trả về giá trị ước chung lớn nhất của x và y. Gcd của x và y được định nghĩa đệ quy như sau: Nếu y bằng nhau đến 0, thì gcd(x, y) là x; mặt khác gcd(x, y) là gcd(y, x % y), trong đó % là toán tử còn lại.

5.40 (Recursive main) Có thể gọi main theo cách đệ quy không? Viết chương trình chứa hàm chủ yếu. Bao gồm số lượng biến cục bộ tĩnh được khởi tạo thành 1. Postincrement và in giá trị của số đếm mỗi khi main

5.41 (Khoảng cách giữa các điểm) Viết hàm distance tính toán khoảng cách giữa hai điểm (x1, y1) và (x2, y2). Tất cả các số và giá trị trả về phải thuộc loại double.

5.42 Đoạn chương trình sau đây làm gì?

5.43 Đoạn chương trình sau đây làm gì?

5.44 Sau khi bạn xác định chương trình của Bài tập 5.43 làm gì, hãy sửa đổi chương trình thành chức năng-

đúng sau khi loại bỏ giới hạn của đối số thứ hai là không âm.

5.45 (Kiểm tra chức năng thư viện toán học) Viết chương trình kiểm tra chức năng thư viện toán học trong

Hình 5.2. Thực hiện từng chức năng này bằng cách yêu cầu chương trình của bạn in ra các bảng giá trị trả về cho sự đa dạng của các giá trị đối số.

5.43 Đoạn chương trình sau đây làm gì?

5.44 Sau khi bạn xác định chương trình của Bài tập 5.43 làm gì, hãy sửa đổi chương trình thành chức năng-

đúng sau khi loại bỏ giới hạn của đối số thứ hai là không âm.

5.45 (Kiểm tra chức năng thư viện toán học) Viết chương trình kiểm tra chức năng thư viện toán học trong

Hình 5.2. Thực hiện từng chức năng này bằng cách yêu cầu chương trình của bạn in ra các bảng giá trị trả về cho sự đa dạng của các giá trị đối số.

5.46 Tìm lỗi trong mỗi đoạn chương trình sau và giải thích cách sửa:

a) khối lập phương kép ( float ); //nguyên mẫu hàm

cube(số float) // định nghĩa hàm

{

trả về số \* số \* số;

}

b) int RandomNumber = srand();

c) gấp đôi y = 123,45678;

intx;

x = y;

printf( "%f\n", (gấp đôi) x );

d) hình vuông kép (số kép)

{

số kép;

trả về số \* số;

}

e) int sum( int n )

{

nếu ( 0 == n ) {

trả về 0;

}

khác {

trả về n + tổng( n );

}

}

5.47 (Sửa đổi trò chơi craps) Sửa đổi chương trình craps của Hình 5.14 để cho phép đặt cược. Đóng gói- tuổi như một chức năng của một phần chương trình chạy một trò chơi xúc xắc. Khởi tạo ngân hàng biến- Số dư đến 1000 đô la. Nhắc người chơi tham gia đặt cược. Sử dụng vòng lặp while để kiểm tra cược đó nhỏ hơn hoặc bằng số dư ngân hàng và nếu không, hãy nhắc người dùng nhập lại cược cho đến khi cược hợp lệ được nhập. Sau khi đặt cược chính xác, hãy chơi một trò chơi xúc xắc. Nếu người chơi thắng, tăng bankBalance theo cược và in bankBalance mới. Nếu người chơi thua, hãy giảm Số dư ngân hàng xuống cược, in Số dư ngân hàng mới, kiểm tra xem Số dư ngân hàng có trở thành 0 không và nếu có thì in tin nhắn, "Xin lỗi. Bạn đã phá sản!" Khi trò chơi tiếp diễn, hãy in các tin nhắn khác nhau để tạo một số “trò chuyện- ter” chẳng hạn như, "Ồ, bạn sắp phá sản hả?" hoặc "Ồ, nắm lấy cơ hội!" hoặc “Con lớn rồi. Bây giờ là lúc để kiếm tiền từ chip của bạn!"

5.48 (Dự án nghiên cứu: Cải thiện việc triển khai Fibonacci đệ quy) Trong Phần 5.15, thuật toán đệ quy mà chúng tôi đã sử dụng để tính toán các số Fibonacci rất hấp dẫn về mặt trực giác. Tuy nhiên, lại gọi rằng thuật toán đã dẫn đến sự bùng nổ theo cấp số nhân của các lệnh gọi hàm đệ quy. Nghiên cứu các triển khai Fibonacci đệ quy trực tuyến. Nghiên cứu các cách tiếp cận khác nhau, bao gồm cả phương pháp lặp trong Bài tập 5.35 và các phiên bản chỉ sử dụng cái gọi là “đệ quy đuôi”. Thảo luận về người thân công đức của mỗi người.

5.49 (Trắc nghiệm về sự nóng lên toàn cầu) Vấn đề gây tranh cãi về sự nóng lên toàn cầu đã được phổ biến rộng rãi được công chiếu bởi bộ phim An Inconvenient Truth, với sự góp mặt của cựu Phó Tổng thống Al Gore. Ông Gore và một mạng lưới các nhà khoa học của Liên hợp quốc, Hội đồng liên chính phủ về biến đổi khí hậu, đã chia sẻ báo cáo năm 2007 giải Nobel Hòa bình để ghi nhận “những nỗ lực của họ nhằm xây dựng và phổ biến kiến ​​thức về biến đổi khí hậu do con người tạo ra.” Nghiên cứu trực tuyến cả hai mặt của vấn đề nóng lên toàn cầu (bạn có thể muốn tìm kiếm các cụm từ như “những người hoài nghi về sự nóng lên toàn cầu”). Tạo một câu hỏi gồm năm câu hỏi bài kiểm tra lựa chọn về sự nóng lên toàn cầu, mỗi câu hỏi có bốn câu trả lời có thể (được đánh số 1–4). vâng lời khách quan và cố gắng trình bày một cách công bằng cả hai mặt của vấn đề. Tiếp theo, hãy viết một ứng dụng quản lý người dùng. Nếu người dùng trả lời đúng năm câu hỏi, hãy in ra “Xuất sắc”; nếu là bốn, hãy in “Rất tốt”; nếu như ba hoặc ít hơn, hãy in “Đã đến lúc tìm hiểu kiến ​​thức của bạn về sự nóng lên toàn cầu,” và bao gồm một danh sách của một số trang web nơi bạn tìm thấy sự thật của mình.

Hướng dẫn sử dụng máy tính

Khi chi phí máy tính giảm, nó trở nên khả thi đối với mọi sinh viên, bất kể hoàn cảnh kinh tế. lập trường, để có một máy tính và sử dụng nó trong trường học. Điều này tạo ra những khả năng thú vị để cải thiện kinh nghiệm giáo dục của tất cả học sinh trên toàn thế giới theo gợi ý của năm bài tập tiếp theo. [Ghi chú: Hãy tham khảo các sáng kiến ​​như Dự án Mỗi trẻ em một máy tính xách tay (www.laptop.org). Ngoài ra, nghiên cứu máy tính xách tay “xanh”—một số đặc điểm chính “chuyển sang màu xanh” của các thiết bị này là gì? Nhìn vào Công cụ Đánh giá Môi trường Sản phẩm Điện tử (www.epeat.net) có thể giúp bạn đánh giá “màu xanh” của máy tính để bàn, máy tính xách tay và màn hình để giúp bạn quyết định nên mua sản

5.50 (Hướng dẫn có sự hỗ trợ của máy tính) Việc sử dụng máy tính trong giáo dục được gọi là hướng dẫn hỗ trợ máy tính (CAI). Viết chương trình giúp học sinh tiểu học học phép nhân. Sử dụng hàm rand để tạo hai số nguyên dương có một chữ số. Chương trình sau đó nên nhắc người dùng bằng một câu hỏi, chẳng hạn như 6 nhân 7 bằng bao nhiêu? Học sinh sau đó nhập câu trả lời. Tiếp theo, chương trình kiểm tra câu trả lời của học sinh. Nếu đúng, hiển thị thông báo "Rất tốt!" và đặt một câu hỏi nhân khác. Nếu câu trả lời là sai, hiển thị thông báo "Không. Vui lòng thử lại." và để học sinh thử lặp đi lặp lại cùng một câu hỏi cho đến khi học sinh cuối cùng hiểu đúng. Nên sử dụng một chức năng riêng biệt để tạo từng câu hỏi mới- sự. Chức năng này nên được gọi một lần khi ứng dụng bắt đầu thực thi và mỗi lần người dùng trả lời đúng câu hỏi.

5.51 (Hướng dẫn có sự hỗ trợ của máy tính: Giảm sự mệt mỏi của học sinh) Một vấn đề trong môi trường CAI- ments là sinh viên mệt mỏi. Điều này có thể được giảm bớt bằng cách thay đổi phản ứng của máy tính để giữ nguyên sự chú ý của nha sĩ. Sửa đổi chương trình của Bài tập 5.50 để các nhận xét khác nhau được hiển thị cho từng câu trả lời như sau:

Phản ứng có thể cho một câu trả lời đúng:

Rất tốt!

Xuất sắc!

Công việc tốt đẹp!

Hãy tiếp tục phát huy!

Các câu trả lời có thể có đối với một câu trả lời sai:

Không. Vui lòng thử lại.

Sai. Thử lại một lần nữa.

Đừng bỏ cuộc!

Không. Tiếp tục cố gắng.

Sử dụng tạo số ngẫu nhiên để chọn một số từ 1 đến 4 sẽ được sử dụng để chọn một trong bốn câu trả lời thích hợp cho mỗi câu trả lời đúng hoặc sai. Sử dụng câu lệnh switch để đưa ra các câu trả lời.

5.52 (Hướng dẫn có sự hỗ trợ của máy tính: Giám sát hiệu suất của học sinh) Tinh vi hơn hệ thống hướng dẫn có sự hỗ trợ của máy tính theo dõi thành tích của học sinh trong một khoảng thời gian. Các quyết định bắt đầu một chủ đề mới thường dựa trên sự thành công của học sinh với các chủ đề trước đó. Biến đổi chương trình của Bài tập 5.51 để đếm số câu trả lời đúng và sai được gõ bởi học sinh. Sau khi học sinh gõ 10 câu trả lời, chương trình của bạn sẽ tính tỷ lệ phần trăm Chính xác. Nếu tỷ lệ phần trăm thấp hơn 75%, hãy hiển thị "Vui lòng nhờ giáo viên của bạn trợ giúp thêm.", hệ thống hướng dẫn có sự hỗ trợ của máy tính theo dõi thành tích của học sinh trong một khoảng thời gian. Các quyết định bắt đầu một chủ đề mới thường dựa trên sự thành công của học sinh với các chủ đề trước đó. Biến đổi chương trình của Bài tập 5.51 để đếm số câu trả lời đúng và sai được gõ bởi học sinh. Sau khi học sinh gõ 10 câu trả lời, chương trình của bạn sẽ tính tỷ lệ phần trăm Chính xác. Nếu tỷ lệ phần trăm thấp hơn 75%, hãy hiển thị "Vui lòng nhờ giáo viên của bạn trợ giúp thêm.", sau đó thiết lập lại chương trình để học sinh khác có thể thử. Nếu tỷ lệ phần trăm là 75% hoặc cao hơn, hãy hiển thị "Xin chúc mừng, bạn đã sẵn sàng để chuyển sang cấp độ tiếp theo!", sau đó thiết lập lại chương trình để khác sinh viên có thể thử nó.

5.53 (Hướng dẫn có sự hỗ trợ của máy tính: Mức độ khó) Bài tập 5.50– đến Bài tập 5.52 đã phát triển một chương trình hướng dẫn có sự trợ giúp của máy tính để giúp dạy học sinh tiểu học lời đề nghị. Sửa đổi chương trình để cho phép người dùng nhập một mức độ khó khăn. Ở mức độ khó của 1, chương trình chỉ nên sử dụng các số có một chữ số trong các bài toán; ở mức độ khó 2, num- bers lớn bằng hai chữ số, v.v.

5.54 (Hướng dẫn có sự trợ giúp của máy tính: Thay đổi các loại vấn đề) Sửa đổi chương trình của Bài tập 5.53 cho phép người dùng chọn một loại bài toán số học để nghiên cứu. Tùy chọn 1 nghĩa là chỉ các bài toán cộng, 2 chỉ các bài toán trừ, 3 chỉ các bài toán nhân và 4 có nghĩa là một hỗn hợp ngẫu nhiên của tất cả các loại này.

**Chương 8: Kí Tự Và Chuỗi**

**8.1.Giới thiệu**

Chương này giới thiệu các hàm thư viện chuẩn C giúp xử lý chuỗi và ký tự. Các hàm này cho phép các chương trình xử lý các ký tự, chuỗi, dòng văn bản và khối bộ nhớ.

**8.2. Nguyên tắc cơ bản của chuỗi và ký tự**

Mỗi chương trình được tạo thành từ một chuỗi các ký tự. Một chương trình có thể chứa các hằng ký tự. Một hằng ký tự là một giá trị int được biểu diễn dưới dạng một ký tự trong dấu nháy đơn. Giá trị của một hằng ký tự là giá trị số nguyên của ký tự đó trong bộ ký tự của máy tính. Ví dụ, 'z' biểu thị giá trị số nguyên của ký tự z, và '\n' biểu thị giá trị số nguyên của ký tự xuống dòng. Một chuỗi có thể bao gồm chữ cái, chữ số và các ký tự đặc biệt khác như +, -, \*, / và $.

Một chuỗi trong C là một mảng các ký tự kết thúc bằng ký tự null ('\0'). Giá trị của một chuỗi là địa chỉ của ký tự đầu tiên trong chuỗi. Do đó, trong C, có thể nói rằng một chuỗi là một con trỏ. Vậy, chuỗi giống như mảng. Một mảng ký tự hoặc một biến có kiểu char \* có thể được khởi tạo với một chuỗi trong một định nghĩa.

char color[] = "blue";

const char \*colorPtr = "blue";

Khi định nghĩa một mảng ký tự để chứa một chuỗi, mảng phải đủ lớn để chứa chuỗi và ký tự null kết thúc. Một chuỗi có thể được lưu trữ trong một mảng bằng cách sử dụng scanf.

scanf( "%19s", word );

**8.3. Ký tự - Xử lý thư viện**

Thư viện xử lý ký tự (<ctype.h>) bao gồm một số hàm thực hiện các kiểm tra và thao tác trên dữ liệu ký tự. Mỗi hàm nhận một tham số unsigned char (được biểu diễn dưới dạng int) hoặc EOF (end of file). Như đã nói trong Chương 4, các ký tự thường được xử lý dưới dạng số nguyên, vì một ký tự trong C là một số nguyên một byte.

**8.3.1. Hàm isdigit, isalpha, isalnum and isxdigit.**

Hàm isdigit xác định xem đối số của nó có phải là một chữ số hay không. Hàm isalpha xác định xem đối số của nó có phải là một chữ cái in hoa hoặc chữ cái thường hay không. Hàm isalnum xác định xem đối số của nó có phải là một chữ cái in hoa, chữ cái thường hoặc một chữ số hay không. Hàm isxdigit xác định xem đối số của nó có phải là một chữ số thập lục phân (A-F, a-f, 0-9) hay không.

**8.3.2. Hàm islower, isupper, tolower and toupper.**

Hàm islower xác định xem đối số của nó có phải là một chữ thường (a-z) hay không. Hàm isupper xác định xem đối số của nó có phải là một chữ in hoa (A-Z) hay không. Hàm tolower chuyển đổi một chữ in hoa thành một chữ thường và trả về chữ thường tương ứng. Hàm toupper chuyển đổi một chữ thường thành một chữ in hoa và trả về chữ in hoa tương ứng.

**8.3.3. Hàm isspace, iscntrl, ispunct, isprint and isgraph.**

Hàm isspace xác định xem một ký tự có phải là một trong các ký tự khoảng trắng sau: dấu cách (' '), ký tự trang bịch ('\f'), ký tự xuống dòng ('\n'), ký tự về đầu dòng ('\r'), tab ngang ('\t') hoặc tab dọc ('\v').

Hàm iscntrl xác định xem một ký tự có phải là một trong các ký tự điều khiển sau: tab ngang ('\t'), tab dọc ('\v'), ký tự trang bịch ('\f'), ký tự cảnh báo ('\a'), ký tự xóa lùi ('\b'), ký tự về đầu dòng ('\r') hoặc ký tự xuống dòng ('\n').

Hàm ispunct xác định xem một ký tự có phải là một ký tự in được khác với ký tự khoảng trắng, ký số hoặc chữ cái, chẳng hạn như $, #, (, ), [, ], {, }, ;, : hoặc %.

Hàm isprint xác định xem một ký tự có thể được hiển thị trên màn hình (bao gồm cả ký tự khoảng trắng).

Hàm isgraph giống như isprint, ngoại trừ ký tự khoảng trắng không được bao gồm

**8.4. Hàm chuyển đổi chuỗi**

Phần này trình bày các hàm chuyển đổi chuỗi thành giá trị số nguyên và số thực từ thư viện (<stdlib.h>).

**8.4.1. Hàm strtoul.**

Hàm strtod chuyển đổi một chuỗi ký tự đại diện cho một giá trị **dấu chấm động** (Dấu chấm động là một kiểu dữ liệu trong lập trình, được sử dụng để biểu diễn các số thực) thành double. Hàm trả về 0 nếu không thể chuyển đổi bất kỳ phần nào của đối số đầu tiên thành double. Hàm nhận hai đối số - một chuỗi (char \*) và một con trỏ đến một chuỗi (char \*\*). Đối số chuỗi chứa chuỗi ký tự cần chuyển đổi thành double - bất kỳ ký tự trắng nào ở đầu chuỗi đều bị bỏ qua. Hàm sử dụng đối số char \*\* để sửa đổi một char \* trong hàm gọi (stringPtr) sao cho nó trỏ đến vị trí của ký tự đầu tiên sau phần được chuyển đổi của chuỗi hoặc toàn bộ chuỗi nếu không có phần nào có thể chuyển đổi.

d = strtod( string, &stringPtr );

**8.4.2. Hàm strtol.**

Hàm strtol chuyển đổi một chuỗi ký tự đại diện cho một số nguyên thành kiểu long int. Hàm trả về giá trị 0 nếu không thể chuyển đổi bất kỳ phần nào của đối số đầu tiên thành kiểu long int. Hàm nhận ba đối số - một chuỗi (char \*), một con trỏ đến chuỗi và một số nguyên. Đối số chuỗi chứa chuỗi ký tự cần chuyển đổi thành số nguyên - bất kỳ ký tự trắng nào ở đầu chuỗi đều bị bỏ qua. Hàm sử dụng đối số char \*\* để sửa đổi một char \* trong hàm gọi (remainderPtr) sao cho nó trỏ đến vị trí của ký tự đầu tiên sau phần đã chuyển đổi của chuỗi hoặc toàn bộ chuỗi nếu không có phần nào có thể chuyển đổi. Số nguyên chỉ định cơ số của giá trị đang được chuyển đổi.

x = strtol( string, &remainderPtr, 0 );

Hàm atoi trong ngôn ngữ lập trình C được sử dụng để chuyển đổi một chuỗi thành giá trị kiểu số long. Tham số đầu tiên của hàm là chuỗi cần chuyển đổi thành kiểu số long. Tham số thứ hai, remainderPtr, sẽ nhận giá trị phần dư của chuỗi sau khi đã được chuyển đổi. Sử dụng NULL cho tham số thứ hai sẽ bỏ qua phần dư của chuỗi. Tham số thứ ba, 0, cho biết giá trị cần được chuyển đổi. Cơ số có thể được chỉ định là 0 hoặc bất kỳ giá trị nào từ 2 đến 36. Biểu diễn số nguyên từ cơ số 11 đến cơ số 36 sử dụng các ký tự A-Z để đại diện cho các giá trị từ 10 đến 35. Ví dụ, giá trị thập lục phân có thể bao gồm các chữ số 0-9 và các ký tự A-F. Một số nguyên cơ số 11 có thể bao gồm các chữ số 0-9 và ký tự A. Một số nguyên cơ số 24 có thể bao gồm các chữ số 0-9 và các ký tự A-N. Một số nguyên cơ số 36 có thể bao gồm các chữ số 0-9 và các ký tự A-Z. Hàm trả về giá trị 0 nếu không thể chuyển đổi bất kỳ phần nào của tham số đầu tiên thành giá trị số long.

**8.4.3.Hàm strtoul**

Hàm strtoul chuyển đổi một chuỗi ký tự đại diện cho giá trị unsigned long int thành unsigned long int. Hàm hoạt động tương tự như hàm strtol.

x = strtoul( string, &remainderPtr, 0 );

Biến x được gán giá trị unsigned long int được chuyển đổi từ chuỗi. Đối số thứ hai, & remainderPtr, được gán giá trị dư của chuỗi sau khi chuyển đổi. Đối số thứ ba, 0, cho biết giá trị cần được chuyển đổi có thể ở định dạng bát phân, thập phân hoặc thập lục phân.

**8.5.Những hàm thư viện nhập, xuất chuẩn**

**8.5.1.Hàm fgets và hàm putchar**

Hàm fgets đọc các ký tự từ đầu vào tiêu chuẩn vào đối số đầu tiên của nó - một mảng các ký tự - cho đến khi gặp dấu xuống dòng hoặc chỉ báo kết thúc tệp hoặc cho đến khi đọc tối đa số lượng ký tự. Số lượng ký tự tối đa là một ít hơn giá trị được chỉ định trong đối số thứ hai của fgets. Đối số thứ ba chỉ định luồng từ đó đọc ký tự. Hàm putchar in ra đối số ký tự của nó

**8.5.2. Hàm getchar**

Hàm getchar để đọc các ký tự từ đầu vào tiêu chuẩn thành câu mảng ký tự và hiển thị các ký tự dưới dạng chuỗi. ký tự hàm đọc một ký tự từ đầu vào tiêu chuẩn và trả về ký tự dưới dạng số nguyên.

**8.5.3. Hàm sprintf**

Hàm sprintf để in dữ liệu được định dạng vào mảng s - một mảng ký tự. Hàm này sử dụng các thông số định dạng giống như printf. Chương trình nhập vào một giá trị int và một giá trị double để định dạng và in ra mảng s. Mảng s là đối số đầu tiên của sprintf.

**8.5.4. Hàm sscanf**

Hàm sscanf để đọc dữ liệu được định dạng từ mảng ký tự s. Hàm này sử dụng các biến đổi giống như scanf. Chương trình đọc một số nguyên và một số thực từ mảng s và lưu giá trị vào x và y tương ứng. Giá trị của x và y được in ra. Mảng s là đối số đầu tiên của hàm sscanf.

**8.6. Các hàm thao tác chuỗi của thư viện xử lý chuỗi**

**8.6.1.Hàm strcpy và strncpy**

Hàm strcpy được sử dụng để sao chép toàn bộ chuỗi trong mảng x vào mảng y và hàm strncpy được sử dụng để sao chép 14 ký tự đầu tiên của mảng x vào mảng z. Một ký tự null ('\0') được thêm vào cuối mảng z vì cuộc gọi hàm strncpy trong chương trình không ghi một ký tự null kết thúc chuỗi (đối số thứ ba nhỏ hơn độ dài chuỗi trong đối số thứ hai).

**8.6.2.Hàm strcat và strncat**

Hàm strcat thêm đối số thứ hai (một chuỗi) vào đối số thứ nhất (một mảng ký tự chứa một chuỗi). Ký tự đầu tiên của đối số thứ hai thay thế ký tự null ('\0') kết thúc chuỗi trong đối số thứ nhất. Bạn phải đảm bảo rằng mảng được sử dụng để lưu trữ chuỗi đầu tiên đủ lớn để lưu trữ chuỗi đầu tiên, chuỗi thứ hai và ký tự null kết thúc được sao chép từ chuỗi thứ hai. Hàm strncat thêm một số ký tự được chỉ định từ chuỗi thứ hai vào chuỗi đầu tiên. Một ký tự null kết thúc được tự động thêm vào kết quả.

**8.7.Hàm so sánh của thư viện xử lý chuỗi**

Hàm strcmp so sánh đối số chuỗi đầu tiên với đối số chuỗi thứ hai, từng ký tự một. Hàm trả về giá trị 0 nếu các chuỗi bằng nhau, giá trị âm nếu chuỗi đầu tiên nhỏ hơn chuỗi thứ hai và giá trị dương nếu chuỗi đầu tiên lớn hơn chuỗi thứ hai. Hàm strncmp tương tự như strcmp, nhưng strncmp so sánh đến một số ký tự được chỉ định. Hàm strncmp không so sánh các ký tự sau ký tự null trong một chuỗi. Chương trình in ra giá trị số nguyên được trả về bởi mỗi lần gọi hàm.

**8.8.Hàm tìm kiếm của thư viện xử lý chuỗi**

**8.8.1.Hàm strchr**

Hàm strchr tìm kiếm sự xuất hiện đầu tiên của một ký tự trong một chuỗi. Nếu ký tự được tìm thấy, strchr trả về một con trỏ tới ký tự trong chuỗi; mặt khác, strchr trả về NULL.

**8.8.2.Hàm strcspn**

Hàm strcspn xác định độ dài của phần đầu tiên của chuỗi trong đối số đầu tiên không chứa bất kỳ ký tự nào từ chuỗi trong đối số thứ hai của nó. Hàm trả về độ dài của đoạn.

**8.8.3.Hàm strpbrk**

Hàm strpbrk tìm kiếm chuỗi đầu tiên trong đối số thứ nhất để tìm sự xuất hiện đầu tiên của bất kỳ ký tự nào trong đối số thứ hai. Nếu tìm thấy một ký tự từ đối số thứ hai, strpbrk trả về một con trỏ đến ký tự trong đối số đầu tiên; nếu không, strpbrk trả về NULL.

**8.8.4.Hàm strrchr**

Hàm strrchr tìm kiếm sự xuất hiện cuối cùng của ký tự được chỉ định trong một chuỗi. Nếu như ký tự được tìm thấy, strrchr trả về một con trỏ tới ký tự trong chuỗi; nếu không thì, strrchr trả về NULL.

**8.8.5.Hàm strspn**

Hàm strspn (Hình 8.24) xác định độ dài của phần đầu của chuỗi trong đối số đầu tiên chỉ chứa các ký tự từ chuỗi trong đối số thứ hai của nó. Hàm trả về độ dài của đoạn.

**8.8.6.Hàm strstr**

Hàm strstr tìm kiếm lần xuất hiện đầu tiên của đối số chuỗi thứ hai trong đối số chuỗi đầu tiên của nó. Nếu chuỗi thứ hai được tìm thấy trong chuỗi đầu tiên, một con trỏ tới vị trí của chuỗi trong đối số đầu tiên được trả về.

**8.8.7.Hàm strtok**

Hàm strtok được sử dụng để ngắt một chuỗi thành một loạt mã thông báo. Mã thông báo là một chuỗi các ký tự được phân tách bằng dấu phân cách (thường là dấu cách hoặc dấu chấm câu, nhưng một dấu phân cách có thể là bất kỳ ký tự nào). Ví dụ, trong một dòng văn bản, mỗi từ có thể được coi một mã thông báo và khoảng trắng và dấu chấm câu ngăn cách các từ có thể được coi là dấu phân cách.

**8.9.Các hàm bộ nhớ của thư viện xử lý chuỗi.**

**8.9.1.Hàm memcpy**

Hàm memcpy được sử dụng để sao chép một số ký tự nhất định từ đối tượng được trỏ tới bởi đối số thứ hai vào đối tượng được trỏ tới bởi đối số thứ nhất. Hàm này có thể nhận một con trỏ tới bất kỳ loại đối tượng nào. Kết quả của hàm này không được xác định nếu hai đối tượng trùng nhau trong bộ nhớ (tức là, nếu chúng là các phần của cùng một đối tượng) - trong những trường hợp như vậy, hãy sử dụng hàm memmove.

**8.9.2.Hàm memmove**

Hàm memmove, giống như memcpy, sao chép một số byte nhất định từ đối tượng được trỏ tới bởi đối số thứ hai vào đối tượng được trỏ tới bởi đối số thứ nhất. Việc sao chép được thực hiện như thể các byte được sao chép từ đối số thứ hai vào một mảng ký tự tạm thời, sau đó được sao chép từ mảng tạm thời vào đối số thứ nhất. Điều này cho phép sao chép các ký tự từ một phần của một chuỗi sang một phần khác của cùng một chuỗi.

**8.9.3.Hàm memcmp**

Hàm memcmp so sánh số ký tự được chỉ định của đối số đầu tiên với các ký tự tương ứng của đối số thứ hai. Hàm trả về một giá trị lớn hơn 0 nếu đối số đầu tiên lớn hơn đối số thứ hai, trả về 0 nếu các đối số bằng nhau và trả về một giá trị nhỏ hơn 0 nếu đối số đầu tiên nhỏ hơn đối số thứ hai.

**8.9.4.Hàm memchr**

Hàm memchr tìm kiếm lần xuất hiện đầu tiên của một byte, được biểu diễn dưới dạng unsigned char, trong số byte được chỉ định của một đối tượng. Nếu byte được tìm thấy, một con trỏ đến byte trong đối tượng sẽ được trả về; nếu không, một con trỏ NULL sẽ được trả về.

**8.9.5.Hàm memset**

Hàm memset sao chép giá trị của byte trong đối số thứ hai vào n byte đầu tiên của đối tượng được trỏ đến bởi đối số đầu tiên, trong đó n được chỉ định bởi đối số thứ ba.

**8.10.Một số hàm khác của thư viện thao tác chuỗi**

**8.10.1.Hàm strerror**

Hàm strerror nhận một số lỗi và tạo ra một chuỗi thông báo lỗi. Con trỏ tới chuỗi được trả

**8.10.2.Hàm strlen**

Hàm strlen nhận một chuỗi làm đối số và trả về số ký tự trong chuỗi đó - ký tự null kết thúc không được tính vào độ dài

**CHƯƠNG 7: CON TRỎ**

**Mục đích**

Trong chương này, bạn sẽ học:

■ Con trỏ và con trỏ người vận hành.

■ Sử dụng con trỏ để chuyển đối số cho các chức năng bằng cách thẩm quyền giải quyết.

■ Mối quan hệ thân thiết giữa các con trỏ, mảng và dây.

■ Để sử dụng con trỏ tới hàm.

■ Để xác định và sử dụng các mảng của dây.

**7.1 Giới thiệu:**

Trong chương này, chúng ta thảo luận về một trong những tính năng mạnh mẽ nhất của ngôn ngữ lập trình C, con trỏ.

Con trỏ là một trong những khả năng khó thành thạo nhất của C. con trỏ cho phép các chương trình mô phỏng chuyển qua tham chiếu, chuyển các chức năng giữa các chức năng và để tạo và thao tác cấu trúc dữ liệu động, nghĩa là cấu trúc dữ liệu có thể phát triển và thu nhỏ tại thời điểm thực hiện, chẳng hạn như danh sách được liên kết, hàng đợi, ngăn xếp và cây. Chương này giải thích rõ các khái niệm cơ bản về con trỏ.

**7.2 Định nghĩa và khởi tạo biến con trỏ:**

Con trỏ là các biến có giá trị là địa chỉ bộ nhớ. Thông thường, một biến trực tiếp chứa một giá trị cụ thể. Mặt khác, một con trỏ chứa một địa chỉ của một biến mà chứa một giá trị cụ thể.

Ngoài ra, biến đếm được xác định là một số nguyên, không phải là một con trỏ tới một số nguyên. Dấu \* chỉ áp dụng để đếmPtr trong định nghĩa. Khi \* được sử dụng theo cách này trong một định nghĩa, nó biểu thị rằng biến đang được định nghĩa là một con trỏ. Con trỏ có thể được định nghĩa để trỏ đến các đối tượng của bất kỳ loại nào. Để tránh sự nhập nhằng trong việc khai báo các biến con trỏ và không phải con trỏ trong khai báo tương tự như hình trên, bạn chỉ nên khai báo một biến cho mỗi khai báo

\***Khởi tạo và gán giá trị cho con trỏ:**

Con trỏ nên được khởi tạo khi chúng được xác định hoặc chúng có thể được gán một giá trị. Một con trỏ có thể được khởi tạo thành NULL, 0 hoặc một địa chỉ. Một con trỏ có giá trị NULL điểm không có vấn đề gì. NULL là một hằng ký hiệu được xác định trong tiêu đề <stddef.h> (và một số tiêu đề khác tiêu đề, chẳng hạn như <stdio.h>). Khởi tạo con trỏ về 0 tương đương với khởi tạo con trỏ thành NULL, nhưng NULL được ưu tiên hơn. Khi 0 được gán, đầu tiên nó được chuyển đổi thành một con trỏ của loại thích hợp. Giá trị 0 là giá trị số nguyên duy nhất có thể được gán trực tiếp cho một biến con trỏ

**7.3 Toán tử con trỏ**

Toán tử &, hoặc địa chỉ, là một toán tử đơn nguyên trả về địa chỉ của toán hạng của nó. Ví dụ, giả sử các định nghĩa tuyên bố gán địa chỉ của biến y cho biến con trỏ yPtr. Biến yPtr sau đó được gọi là “chỉ vào” y.

Toán tử gián tiếp (\*)

Toán tử \* đơn nguyên, thường được gọi là toán tử gián tiếp hoặc hội nghị

toán tử, trả về giá trị của đối tượng mà toán hạng của nó (nghĩa là một con trỏ) trỏ tới. Ví dụ, tuyên bố in ra giá trị của biến y, cụ thể là 5. Sử dụng \* theo cách này được gọi là hủy bỏ hội nghị một con trỏ.

Biểu diễn toán tử & và \*

**7.4 Truyền đối số cho hàm bằng cách tham chiếu**

Có hai cách để truyền đối số cho một hàm—truyền theo giá trị và truyền theo tham chiếu. Tất cả các đối số trong C đều được truyền theo giá trị. Như chúng ta đã thấy trong Chương 5, return có thể được sử dụng để trả lại một giá trị từ hàm được gọi cho người gọi (hoặc trả lại điều khiển từ hàm được gọi chức năng mà không trả lại một giá trị). Nhiều chức năng yêu cầu khả năng sửa đổi các biến trong trình gọi hoặc để chuyển một con trỏ tới một đối tượng dữ liệu lớn để tránh chi phí hoạt động truyền đối tượng theo giá trị (làm phát sinh chi phí thời gian và bộ nhớ để tạo một bản sao của đối tượng).

Trong C, bạn sử dụng con trỏ và toán tử gián tiếp để mô phỏng chuyển qua tham chiếu.

Khi gọi một hàm với các đối số cần được sửa đổi, địa chỉ của các đối số sẽ được chuyển. Điều này thường được thực hiện bằng cách áp dụng toán tử địa chỉ (&) cho biến (trong trình gọi) có giá trị sẽ được sửa đổi. Như chúng ta đã thấy trong Chương 6, mảng là không được chuyển bằng toán tử & vì C tự động chuyển vị trí bắt đầu trong bộ nhớ của mảng (tên của mảng tương đương với &arrayName[0]). Khi địa chỉ của

một biến được truyền cho một hàm, toán tử gián tiếp (\*) có thể được sử dụng trong hàm để sửa đổi giá trị tại vị trí đó trong bộ nhớ của người gọi.

**7.5 Sử dụng vòng loại hằng số với con trỏ:**

Vòng loại const cho phép bạn thông báo cho trình biên dịch rằng không nên sửa đổi giá trị của một biến cụ thể.

Trong những năm qua, một lượng lớn mã kế thừa đã được viết trong các phiên bản đầu tiên của C đã làm không sử dụng const vì nó không có sẵn. Vì lý do này, có nhiều cơ hội đáng kể để cải tiến bằng cách tái cấu trúc mã C cũ.

Có sáu khả năng sử dụng (hoặc không sử dụng) const với các tham số chức năng—hai với truyền tham số truyền theo giá trị và bốn với truyền tham số truyền theo tham chiếu.

Làm thế nào để bạn chọn một trong sáu khả năng? Hãy để nguyên tắc đặc quyền tối thiểu là của bạn

hướng dẫn. Luôn cấp cho một chức năng đủ quyền truy cập vào dữ liệu trong các tham số của nó để hoàn thành

nhiệm vụ được chỉ định của nó, nhưng hoàn toàn không còn nữa.

**7.5.1 Chuyển đổi chuỗi thành chữ hoa bằng con trỏ không cố định**

**đến dữ liệu không cố định**

Mức truy cập dữ liệu cao nhất được cấp bởi một con trỏ không cố định tới dữ liệu không cố định.

Trong trường hợp này, dữ liệu có thể được sửa đổi thông qua con trỏ hủy đăng ký và con trỏ có thể

được sửa đổi để trỏ đến các mục dữ liệu khác. Một khai báo cho một con trỏ không cố định tới dữ liệu không cố định không bao gồm const. Một con trỏ như vậy có thể được sử dụng để nhận một chuỗi dưới dạng

đối số cho một hàm xử lý (và có thể sửa đổi) từng ký tự trong chuỗi.

**7.5.2 In một chuỗi một ký tự tại một thời điểm bằng cách sử dụng một giá trị không cố định**

Con trỏ tới dữ liệu không đổi

Một con trỏ không cố định tới dữ liệu không đổi có thể được sửa đổi để trỏ tới bất kỳ mục dữ liệu nào của loại thích hợp, nhưng không thể sửa đổi dữ liệu mà nó trỏ tới. Một con trỏ như vậy có thể được sử dụng để nhận một đối số mảng cho một hàm sẽ xử lý từng phần tử mà không cần sửa đổi dữ liệu

**7.5.3 Cố gắng Sửa đổi Con trỏ Hằng thành Dữ liệu Không Hằng**

Một con trỏ hằng tới dữ liệu không hằng luôn trỏ tới cùng một vị trí bộ nhớ và

dữ liệu tại vị trí đó có thể được sửa đổi thông qua con trỏ. Đây là mặc định cho một mảng tên. Tên mảng là một con trỏ hằng đến đầu mảng. Tất cả dữ liệu trong

mảng có thể được truy cập và thay đổi bằng cách sử dụng tên mảng và chỉ số mảng. Có thể sử dụng một con trỏ hằng tới dữ liệu không hằng để nhận một mảng làm đối số cho một hàm truy cập các phần tử mảng bằng cách chỉ sử dụng ký hiệu chỉ số con của mảng. Con trỏ đó là const đã khai báo phải được khởi tạo khi chúng được xác định (nếu con trỏ là một thông số tham số của hàm, thì nó được khởi tạo với một con trỏ được truyền cho hàm).

**7.5.4 Cố gắng Sửa đổi Con trỏ Hằng thành Dữ liệu Hằng**

Đặc quyền truy cập ít nhất được cấp bởi một con trỏ không đổi tới dữ liệu không đổi. Một con trỏ như vậy luôn trỏ đến cùng một vị trí bộ nhớ và dữ liệu tại vị trí bộ nhớ đó không thể được chỉnh sửa. Đây là cách một mảng nên được chuyển đến một hàm chỉ nhìn vào mảng sử dụng ký hiệu chỉ số dưới mảng và không sửa đổi mảng

**7.6 Sắp xếp bong bóng bằng cách sử dụng Pass-by-Reference**

Hãy cải thiện chương trình sắp xếp bong bóng của Hình 6.15 để sử dụng hai chức năng—bubbleSort và tráo đổi. Hàm bubbleSort sắp xếp mảng. Nó gọi hàm hoán đổi (dòng 50) để trao đổi phần tử mảng array[j] và array[j + 1] (Hình 7.15). Hãy nhớ rằng C thực thi ẩn thông tin giữa các hàm, vì vậy hoán đổi không có quyền truy cập vào các phần tử mảng riêng lẻ trong bong bóngSắp xếp. Bởi vì bubbleSort muốn trao đổi để có quyền truy cập vào các phần tử mảng hoán đổi, bubbleSort chuyển từng phần tử này bằng cách tham chiếu đến hoán đổi—địa chỉ của mỗi phần tử mảng được thông qua một cách rõ ràng. Mặc dù toàn bộ mảng được tự động chuyển qua tham chiếu, các phần tử mảng riêng lẻ là vô hướng và thường được truyền theo giá trị. Do đó, bubbleSort sử dụng toán tử địa chỉ (&) trên mỗi phần tử mảng trong hoán đổi.

Mặc dù kích thước tham số nhận một bản sao của một giá trị trong chính và sửa đổi bản sao không thể thay đổi giá trị trong main, bubbleSort không cần thay đổi kích thước để hoàn thành nhiệm vụ. Kích thước của mảng vẫn cố định trong quá trình thực thi hàm bubbleSort.

Do đó, kích thước được khai báo const để đảm bảo rằng nó không bị sửa đổi.

Nguyên mẫu cho hàm hoán đổi (dòng 38) được bao gồm trong phần thân của hàm bong bóng bleSort vì bubbleSort là hàm duy nhất gọi hàm hoán đổi. Đặt nguyên mẫu vào bubbleSort hạn chế các cuộc gọi trao đổi thích hợp đối với những cuộc gọi được thực hiện từ bong bóngSort. Cac chưc năng khác nỗ lực gọi hoán đổi không có quyền truy cập vào một nguyên mẫu chức năng phù hợp, vì vậy trình biên dịch tự động tạo ra một cái. Điều này thường dẫn đến một nguyên mẫu không phù hợp với tiêu đề hàm (và tạo cảnh báo hoặc lỗi biên dịch) vì trình biên dịch

giả sử int cho kiểu trả về và kiểu tham số.

Trong chương trình, kích thước của mảng được truyền rõ ràng cho hàm bubbleSort. Ở đó hai lợi ích chính của cách tiếp cận này—khả năng tái sử dụng phần mềm và kỹ thuật phần mềm phù hợp. Bằng cách xác định hàm để nhận kích thước mảng làm đối số, chúng tôi kích hoạt hàm được sử dụng bởi bất kỳ chương trình nào sắp xếp các mảng số nguyên có chỉ số đơn ở bất kỳ kích thước nào.

Chúng tôi có thể đã lưu trữ kích thước của mảng trong một biến toàn cục mà toàn bộ có thể truy cập được chương trình. Điều này sẽ hiệu quả hơn, bởi vì một bản sao của kích thước không được chuyển đến chức năng. Tuy nhiên, các chương trình khác yêu cầu khả năng sắp xếp mảng số nguyên có thể không có cùng biến toàn cục nên không dùng được hàm trong các chương trình đó.

Kích thước của mảng có thể đã được lập trình trực tiếp vào hàm. Cái này hạn chế việc sử dụng hàm đối với một mảng có kích thước cụ thể và giảm đáng kể

khả năng tái sử dụng. Chỉ các chương trình xử lý mảng số nguyên đăng ký đơn có kích thước cụ thể được mã hóa vào chức năng có thể sử dụng chức năng

**7.7 Toán tử sizeof**

C cung cấp toán tử một ngôi đặc biệt sizeof để xác định kích thước theo byte của một mảng (hoặc bất kỳ kiểu dữ liệu nào khác).

**7.8 Biểu thức con trỏ và số học con trỏ**

Con trỏ là toán hạng hợp lệ trong biểu thức số học, biểu thức gán và biểu thức so sánh. Tuy nhiên, không phải tất cả các toán tử thường được sử dụng trong các biểu thức này đều hợp lệ kết hợp với các biến con trỏ. Phần này mô tả các toán tử có thể có con trỏ là toán hạng và cách sử dụng các toán tử này.

Một tập giới hạn các phép toán số học có thể được thực hiện trên con trỏ. Một con trỏ có thể được tăng (++) hoặc giảm (--), một số nguyên có thể được thêm vào một con trỏ (+ hoặc +=), một số nguyên có thể bị trừ khỏi một con trỏ (- hoặc -=) và một con trỏ có thể bị trừ khỏi khác—thao tác cuối cùng này chỉ có ý nghĩa khi cả hai con trỏ đều trỏ đến các phần tử của cùng một mảng.

**7.9 Mối quan hệ giữa con trỏ và mảng**

Mảng và con trỏ có liên quan mật thiết với nhau trong C và thường có thể được sử dụng thay thế cho nhau. Một tên mảng có thể được coi là một con trỏ không đổi. Con trỏ có thể được sử dụng để thực hiện bất kỳ thao tác nào liên quan đến đăng ký mảng.

Giả sử rằng mảng số nguyên b[5] và biến con trỏ số nguyên bPtr đã được xác định.

Vì tên mảng (không có chỉ số dưới) là con trỏ tới phần tử đầu tiên của mảng nên chúng ta có thể đặt bPtr bằng địa chỉ của phần tử đầu tiên trong mảng b với câu lệnh

**7.10 Mảng con trỏ**

Mảng có thể chứa con trỏ. Một cách sử dụng phổ biến của mảng con trỏ là tạo thành một mảng chuỗi, được gọi đơn giản là một mảng chuỗi. Mỗi mục trong mảng là một chuỗi, nhưng trong C một chuỗi thực chất là một con trỏ tới ký tự đầu tiên của nó. Vì vậy, mỗi mục trong một mảng các chuỗi thực sự là một con trỏ tới ký tự đầu tiên của một chuỗi. Xem xét định nghĩa của mảng chuỗi bộ đồ, có thể hữu ích trong việc đại diện cho một bộ bài.

Các bộ quần áo có thể đã được đặt trong một mảng hai chiều, trong đó mỗi hàng sẽ đại diện cho một bộ đồ và mỗi cột sẽ đại diện cho một chữ cái trong tên bộ đồ. như vậy cấu trúc dữ liệu sẽ phải có một số cột cố định trên mỗi hàng và số đó sẽ phải lớn bằng chuỗi lớn nhất. Do đó, bộ nhớ đáng kể có thể được lãng phí khi lưu trữ một số lượng lớn các chuỗi trong đó hầu hết ngắn hơn chuỗi dài nhất sợi dây. Chúng ta sử dụng mảng chuỗi để biểu diễn một bộ bài trong phần tiếp theo.

**7.11 Nghiên cứu điển hình: Mô phỏng xáo bài và chia bài**

Trong phần này, chúng tôi sử dụng tạo số ngẫu nhiên để phát triển xáo bài và chia bài chương trình mô phỏng. Chương trình này sau đó có thể được sử dụng để thực hiện các chương trình chơi trò chơi thẻ cụ thể. Để tiết lộ một số vấn đề tinh tế về hiệu suất, chúng tôi đã cố tình sử dụng các thuật toán xáo trộn và xử lý dưới mức tối ưu. Trong các bài tập của chương này và trong Chương 10, chúng tôi phát triển các thuật toán hiệu quả hơn.

Sử dụng cách tiếp cận sàng lọc từng bước từ trên xuống, chúng tôi phát triển một chương trình sẽ xáo một bộ bài gồm 52 quân bài và sau đó chia từng quân bài trong số 52 quân bài đó. Từ trên xuống cách tiếp cận đặc biệt hữu ích trong việc tấn công các vấn đề lớn hơn, phức tạp hơn bạn đã thấy trong các chương trước.

**7.12 Con trỏ tới hàm**

Con trỏ tới hàm chứa địa chỉ của hàm trong bộ nhớ. Trong Chương 6, chúng tôi thấy rằng một tên mảng thực sự là địa chỉ trong bộ nhớ của phần tử đầu tiên của mảng.

Tương tự như vậy, tên hàm thực sự là địa chỉ bắt đầu trong bộ nhớ của mã mà mỗi hình thức thực hiện nhiệm vụ của hàm. Con trỏ tới hàm có thể được truyền tới hàm, được trả về từ các hàm, được lưu trữ trong mảng và được gán cho các con trỏ hàm khác.

**7.13 Lập trình C an toàn**

printf\_s, scanf\_s và các chức năng bảo mật khác

Các phần Lập trình C bảo mật trước đó đã trình bày printf\_s và scanf\_s và đã đề cập các phiên bản an toàn hơn khác của các chức năng thư viện tiêu chuẩn được mô tả trong Phụ lục K của tiêu chuẩn C. Một tính năng chính của các chức năng như printf\_s và scanf\_s giúp chúng trở nên hiệu quả hơn an toàn là chúng có các ràng buộc về thời gian chạy yêu cầu các đối số con trỏ của chúng không phải là NULL.

Các chức năng kiểm tra các ràng buộc thời gian chạy này trước khi cố gắng sử dụng các con trỏ. Bất kì đối số con trỏ NULL được coi là vi phạm ràng buộc và khiến hàm thất bại và trả về một thông báo trạng thái. Trong một scanf\_s, nếu bất kỳ đối số con trỏ nào (bao gồm cả chuỗi kiểm soát định dạng) là NULL, hàm trả về EOF. Trong printf\_s, nếu chuỗi điều khiển định dạng hoặc bất kỳ đối số nào tương ứng với %s là NULL, hàm sẽ dừng xuất dữ liệu ting và trả về một số âm. Để biết chi tiết đầy đủ về các chức năng của Phụ lục K, xem tài liệu chuẩn C hoặc tài liệu thư viện của trình biên dịch của bạn.

**CHƯƠNG 12 CẤU TRÚC DỮ LIỆU**

# Mục tiêu

- Cấp phát và giải phóng bộ nhớ

động cho các đối tượng dữ liệu.

- Cấu trúc dữ liệu dạng liên kết

sử dụng con trỏ, tự tham khảo

cấu trúc và đệ quy.

- Tạo và thao tác liên kết

danh sách, hàng đợi, ngăn xếp và

cây nhị phân.

- Tìm hiểu các ứng dụng quan trọng

của các cấu trúc dữ liệu được liên kết.

Giới thiệu sơ lược các nội dung trong chương

12.1 Giới thiệu

12.2 Cấu trúc tự tham chiếu

12.3 Cấp phát bộ nhớ động

12.4 Danh sách liên kết

12.4.1 Chức năng chèn

12.4.2 Chức năng xóa

12.4.3 Hàm printList

12,5 ngăn xếp

12.5.1 Đẩy chức năng

12.5.2 Bật chức năng

12.5.3 Các ứng dụng của ngăn xếp

12.6 Hàng đợi

12.6.1 Chức năng xếp hàng

12.6.2 Hàm dequeue

12.7 Cây

12.7.1 Chèn nút chức năng

12.7.2 Traversal: Hàm theo thứ tự,

đặt hàng trước và đặt hàng sau

12.7.3 Loại bỏ trùng lặp

12.7.4 Tìm kiếm cây nhị phân

12.7.5 Các phép toán cây nhị phân khác

12.8 Lập trình C an toàn

**12.1 Giới thiệu**

-Chúng tôi đã nghiên cứu các cấu trúc dữ liệu có kích thước cố định, chẳng hạn như mảng có chỉ số đơn, mảng có chỉ số kép và cấu trúc. Chương này giới thiệu các cấu trúc dữ liệu động có thể phát triển và thu nhỏ tại thời điểm thực hiện.

+ Danh sách được liên kết là tập hợp các mục dữ liệu “xếp thành một hàng”—việc thêm và xóa được thực hiện ở bất kỳ đâu trong danh sách được liên kết.

+ Ngăn xếp rất quan trọng trong trình biên dịch và hệ điều hành—việc chèn và xóa chỉ được thực hiện ở một đầu của ngăn xếp—đỉnh của ngăn xếp.

+ Queues biểu thị các hàng đợi; phần chèn chỉ được thực hiện ở phía sau (còn được gọi là

thành phần đuôi) của hàng đợi và việc xóa chỉ được thực hiện từ phía trước (còn được gọi là

làm đầu) của một hàng đợi.

+ Cây nhị phân hỗ trợ tìm kiếm và sắp xếp dữ liệu tốc độ cao, loại bỏ hiệu quả các mục dữ liệu trùng lặp, biểu diễn các thư mục hệ thống tệp và biên dịch các biểu thức sang ngôn ngữ máy.

- Mỗi cấu trúc dữ liệu này có nhiều ứng dụng thú vị khác.

- Chúng ta sẽ thảo luận về từng loại cấu trúc dữ liệu chính và triển khai các chương trình tạo và thao tác với chúng. Trong phần tiếp theo của cuốn sách—giới thiệu về C++ và lập trình hướng đối tượng—chúng ta sẽ nghiên cứu về trừu tượng hóa dữ liệu. Kỹ thuật này sẽ cho phép chúng ta để xây dựng các cấu trúc dữ liệu này theo cách khác biệt đáng kể được thiết kế để sản xuất phần mềm dễ bảo trì và tái sử dụng hơn.

\* Dự án tùy chọn: Xây dựng trình biên dịch của riêng bạn

- Chúng tôi hy vọng rằng bạn sẽ thử thực hiện dự án lớn được mô tả trong phần đặc biệt có tựa đề Xây dựng trình biên dịch của riêng bạn. Bạn đang sử dụng một trình biên dịch để dịch các chương trình C sang ngôn ngữ máy để bạn có thể thực thi các chương trình của mình trên máy tính. Trong dự án này, bạn sẽ thực sự xây dựng trình biên dịch của riêng mình. Nó sẽ đọc một tệp các câu lệnh được viết bằng một ngôn ngữ cấp cao đơn giản nhưng mạnh mẽ tương tự như các phiên bản đầu tiên của ngôn ngữ phổ biến

\* NỀN TẢNG. Trình biên dịch của bạn sẽ dịch các câu lệnh này thành một tệp hướng dẫn Ngôn ngữ máy Simpletron (SML). SML là ngôn ngữ bạn đã học trong phần đặc biệt của Chương 7,

- Xây dựng máy tính của riêng bạn. Sau đó, chương trình Simpletron Simulator của bạn sẽ thực thi Chương trình SML do trình biên dịch của bạn tạo ra! Dự án này sẽ mang đến cho bạn cơ hội tuyệt vời để thực hiện hầu hết những gì bạn đã học được trong khóa học này. Phần đặc biệt cẩn thận đi bộ bạn thông qua các thông số kỹ thuật của ngôn ngữ cấp cao và mô tả các thuật toán mà bạn sẽ cần chuyển từng loại câu lệnh của ngôn ngữ cấp cao thành các lệnh của ngôn ngữ máy. Nếu bạn thích được thử thách, bạn có thể thử nhiều cải tiến cho cả hai. Trình biên dịch và trình mô phỏng Simpletron được đề xuất trong các bài tập.

**12.2 Cấu trúc tự tham chiếu**

- Nhớ lại rằng một cấu trúc tự tham chiếu chứa một thành viên con trỏ trỏ đến một cấu trúc cùng kiểu kết cấu. Ví dụ, định nghĩa

struct node {  
int data;  
struct node \*nextPtr;  
}; // Kết thúc node

định nghĩa một loại, nút cấu trúc. Một cấu trúc của nút kiểu cấu trúc có hai thành viên—số nguyên dữ liệu thành viên và con trỏ thành viên nextPtr. Thành viên nextPtr trỏ đến một cấu trúc của nút cấu trúc kiểu—một cấu trúc cùng loại với cấu trúc được khai báo ở đây, do đó thuật ngữ cấu trúc tự quy chiếu. Thành viên nextPtr được gọi là một liên kết—nghĩa là nó có thể được sử dụng để “buộc” một cấu trúc nút kiểu cấu trúc với một cấu trúc khác cùng kiểu. Các cấu trúc tự tham chiếu có thể được liên kết với nhau để tạo thành các cấu trúc dữ liệu hữu ích như danh sách, hàng đợi, ngăn xếp và cây cối. Hình 12.1 minh họa hai đối tượng cấu trúc tự tham chiếu được liên kết với nhau để tạo thành một danh sách. Dấu gạch chéo—đại diện cho một con trỏ NULL—được đặt trong thành viên liên kết của cấu trúc tự tham chiếu thứ hai để chỉ ra rằng liên kết không trỏ đến cấu trúc khác. [Lưu ý: Dấu gạch chéo chỉ nhằm mục đích minh họa; nó không tương ứng với ký tự gạch chéo ngược trong C.] Một con trỏ NULL thường chỉ ra phần cuối của cấu trúc dữ liệu giống như ký tự null cho biết kết thúc chuỗi

\* Chú ý Lỗi lập trình thường gặp 12.1

- Không đặt liên kết trong nút cuối cùng của danh sách thành NULL có thể dẫn đến lỗi thời gian chạy.

10

15

Hình 12.1 | Các cấu trúc tự tham chiếu được liên kết với nhau

**12.3 Cấp phát bộ nhớ động**

-Việc tạo và duy trì cấu trúc dữ liệu động yêu cầu cấp phát bộ nhớ động—khả năng chương trình có được nhiều không gian bộ nhớ hơn tại thời điểm thực thi để giữ bộ nhớ mới. C’ác nút và để giải phóng không gian không còn cần thiết.

- Các hàm malloc và free, và toán tử sizeof, rất cần thiết cho bộ nhớ động phân bổ. Hàm malloc nhận đối số là số byte được phân bổ và trả về một con trỏ kiểu void \* (con trỏ tới void) vào bộ nhớ được cấp phát. Như bạn nhớ lại, một con trỏ void \* có thể được gán cho một biến thuộc bất kỳ loại con trỏ nào. Hàm malloc thường được sử dụng với toán tử sizeof. Ví dụ, tuyên bố

*new ptr = malloc ( sizeof ( struct node ) );*

đánh giá sizeof(struct node) để xác định kích thước theo byte của cấu trúc kiểu struct nút, phân bổ một vùng mới trong bộ nhớ của số byte đó và lưu một con trỏ tới bộ nhớ được cấp phát trong biến newPtr. Bộ nhớ được phân bổ không được khởi tạo. Nếu không có bộ nhớ, malloc trả về NULL.

- Chức năng giải phóng bộ nhớ miễn phí—tức là, bộ nhớ được trả về hệ thống để nó có thể được phân bổ lại trong tương lai. Để giải phóng bộ nhớ được cấp phát động bởi phần trước cuộc gọi malloc, sử dụng câu lệnh

*Free ( new ptr );*

- C cũng cung cấp các hàm calloc và realloc để tạo và sửa đổi động mảng. Các chức năng này được thảo luận trong Phần 14.9. Các phần theo sau danh sách thảo luận, ngăn xếp, hàng đợi và cây, mỗi ngăn xếp được tạo và duy trì bằng bộ nhớ động cấu trúc phân bổ và tự tham chiếu.

\* Lưu ý khi sử dụng cấp phát động

+Mẹo về tính di động 12.1

Kích thước của một cấu trúc không nhất thiết phải là tổng kích thước của các thành viên. Điều này là như vậy bởi vì các yêu cầu căn chỉnh ranh giới phụ thuộc vào máy khác nhau (xem Chương 10).

\* Mẹo ngăn ngừa lỗi 12.1

Khi sử dụng malloc, hãy kiểm tra giá trị trả về của con trỏ NULL, giá trị này cho biết bộ nhớ chưa được cấp phát.

\* Lỗi lập trình thường gặp 12.2

Việc không trả về bộ nhớ được cấp phát động khi không còn cần thiết nữa có thể khiến hệ thống hết bộ nhớ sớm. Điều này đôi khi được gọi là "rò rỉ bộ nhớ".

\* Mẹo ngăn ngừa lỗi 12.2

Khi bộ nhớ được cấp phát động không còn cần thiết, hãy sử dụng free để trả về bộ nhớ vào hệ thống ngay lập tức. Sau đó đặt con trỏ thành NULL để loại bỏ khả năng rằng chương trình có thể đề cập đến bộ nhớ đã được thu hồi và có thể đã được phân bổ cho mục đích khác.

\* Lỗi lập trình thường gặp 12.3

Giải phóng bộ nhớ không được cấp phát động với malloc là một lỗi.

\* Lỗi lập trình phổ biến 12.4

Đề cập đến bộ nhớ đã được giải phóng là một lỗi thường dẫn đến chương trình

va chạm

**12.4 Danh sách liên kết**

- Danh sách liên kết là một tập hợp tuyến tính của các cấu trúc tự tham chiếu, được gọi là các nút, được kết nối bởi liên kết con trỏ—do đó, thuật ngữ danh sách "được liên kết". Một danh sách liên kết được truy cập thông qua một con trỏ tới nút đầu tiên của danh sách. Các nút tiếp theo được truy cập thông qua thành viên con trỏ liên kết được lưu trữ trong mỗi nút. Theo quy ước, con trỏ liên kết trong nút cuối cùng của danh sách được đặt thành NULL để đánh dấu cuối danh sách. Dữ liệu được lưu trữ trong danh sách liên kết động—mỗi nút được tạo khi cần thiết. Một nút có thể chứa dữ liệu thuộc bất kỳ loại nào kể cả các cấu trúc khác. Ngăn xếp và hàng đợi cũng là các cấu trúc dữ liệu tuyến tính, và như chúng ta sẽ thấy, là các phiên bản ràng buộc của danh sách liên kết.Cây là cấu trúc dữ liệu phi tuyến tính.

- Danh sách dữ liệu có thể được lưu trữ trong mảng, nhưng danh sách được liên kết cung cấp một số lợi thế. MỘT danh sách liên kết phù hợp khi số phần tử dữ liệu được biểu diễn trong dữ liệu cấu trúc là không thể đoán trước. Danh sách được liên kết là động nên độ dài của danh sách có thể tăng hoặc giảm giảm tại thời điểm thực hiện khi cần thiết. Tuy nhiên, kích thước của một mảng được tạo tại thời điểm biên dịch không thể thay đổi được. Mảng có thể trở nên đầy đủ. Danh sách được liên kết trở nên đầy đủ chỉ khi hệ thống không đủ bộ nhớ để đáp ứng các yêu cầu cấp phát bộ nhớ động.

\* Mẹo hiệu suất 12.1

Một mảng có thể được khai báo để chứa nhiều phần tử hơn số mục dữ liệu dự kiến, nhưng điều này có thể gây lãng phí bộ nhớ. Danh sách được liên kết có thể cung cấp khả năng sử dụng bộ nhớ tốt hơn trong những tình huống này

Danh sách liên kết có thể được duy trì theo thứ tự được sắp xếp bằng cách chèn từng phần tử mới vào điểm thích hợp trong danh sách.

\* Mẹo hiệu suất 12.2

Việc chèn và xóa trong một mảng đã sắp xếp có thể tốn nhiều thời gian—tất cả các phần tử theo sau phần tử được chèn hoặc xóa phải được dịch chuyển một cách thích hợp.

\* Mẹo hiệu suất 12.3

Các phần tử của một mảng được lưu trữ liên tục trong bộ nhớ. Điều này cho phép truy cập ngay lập tức đến bất kỳ phần tử mảng nào vì địa chỉ của bất kỳ phần tử nào có thể được tính trực tiếp dựa trên vào vị trí của nó so với phần đầu của mảng. Các danh sách được liên kết không đủ khả năng truy cập ngay lập tức vào các phần tử của chúng.

- Các nút danh sách liên kết thường không được lưu trữ liên tục trong bộ nhớ. Tuy nhiên, về mặt logic, các nút của một danh sách được liên kết dường như liền kề nhau. Hình 12.2 minh họa một danh sách liên kết với một số nút.

\* Mẹo hiệu suất 12.4

Sử dụng cấp phát bộ nhớ động (thay vì mảng) cho cấu trúc dữ liệu phát triển và thu nhỏ tại thời điểm thực hiện có thể tiết kiệm bộ nhớ. Tuy nhiên, hãy ghi nhớ rằng các con trỏ lấy hết dung lượng và việc cấp phát bộ nhớ động đó phát sinh chi phí chung cho các lệnh gọi hàm.

Star Ptr

93

15

10

Hình 12.2 biểu diễn đồ họa danh sách liên kết.

Hình 12.3 (đầu ra được hiển thị trong Hình 12.4) thao tác với một danh sách các ký tự. Bạn có thể chèn một ký tự vào danh sách theo thứ tự bảng chữ cái (hàm chèn) hoặc xóa một ký tự khỏi danh sách (chức năng xóa). Sau đây là phần thảo luận chi tiết về chương trìnhchèn và xoá các nút tring danh sách

1 // Fig. 12.3: fig12\_03.c  
2 // Inserting and deleting nodes in a list  
3 #include <stdio.h>  
4 #include <stdlib.h>  
5

6// self-referential structure  
7 struct listNode {  
8 char data; // each listNode contains a character  
9 struct listNode \*nextPtr; // pointer to next node  
10 }; // end structure listNode

11  
12 typedef struct listNode ListNode; // synonym for struct listNode  
13 typedef ListNode \*ListNodePtr; // synonym for ListNode\*  
14  
15 // prototypes  
16 void insert( ListNodePtr \*sPtr, char value );  
17 char delete( ListNodePtr \*sPtr, char value );  
18 int isEmpty( ListNodePtr sPtr );  
19 void printList( ListNodePtr currentPtr );  
20 void instructions( void );  
21  
22 int main( void )  
23 {  
24 ListNodePtr startPtr = NULL; // initially there are no nodes  
25 unsigned int choice; // user's choice  
26 char item; // char entered by user  
27  
28 instructions(); // display the menu  
29 printf( "%s", "? " );  
30 scanf( "%u", &choice );  
31  
32 // loop while user does not choose 3  
33 while ( choice != 3 ) {  
34  
35 switch ( choice ) {  
36 case 1:  
37 printf( "%s", "Enter a character: " );  
38 scanf( "\n%c", &item );  
39  
40 printList( startPtr );  
41 break;  
42 case 2: // delete an element  
43 // if list is not empty  
44 if ( !isEmpty( startPtr ) ) {  
45 printf( "%s", "Enter character to be deleted: " );  
46 scanf( "\n%c", &item );

47

48 // if character is found, remove it  
49  
50 printf( "%c deleted.\n", item );  
51 printList( startPtr );  
52 } // end if  
53 else {  
54 printf( "%c not found.\n\n", item );  
55 } // end else  
56 } // end if  
57 else {  
58 puts( "List is empty.\n" );  
59 } // end else  
60  
61 break;  
62 default:  
63 puts( "Invalid choice.\n" );  
64 instructions();  
65 break;  
66 } // end switch  
67  
68 printf( "%s", "? " );  
69 scanf( "%u", &choice );  
70 } // end while  
71  
72 puts( "End of run." );  
73 } // end main  
74  
75 // display program instructions to user

76 void instructions( void )  
77 {  
78 puts( "Enter your choice:\n"  
79 " 1 to insert an element into the list.\n"  
80 " 2 to delete an element from the list.\n"  
81 " 3 to end." );  
82 } // end function instructions  
83  
84 // insert a new value into the list in sorted order  
85 void insert( ListNodePtr \*sPtr, char value )  
86 {  
87 ListNodePtr newPtr; // pointer to new node  
88 ListNodePtr previousPtr; // pointer to previous node in list  
89 ListNodePtr currentPtr; // pointer to current node in list  
90  
91newPtr = malloc( sizeof( ListNode ) ); // create node  
92  
93 if ( newPtr != NULL ) { // is space available  
94 newPtr->data = value; // place value in node  
95 newPtr->nextPtr = NULL; // node does not link to another node  
96  
97 previousPtr = NULL;  
98 currentPtr = \*sPtr;  
99

100// loop to find the correct location in the list  
101 while ( currentPtr != NULL && value > currentPtr->data ) {  
102previousPtr = currentPtr; // walk to ...  
103currentPtr = currentPtr->nextPtr; // ... next node  
104} // end while  
105  
106 // insert new node at beginning of list  
107 if ( previousPtr == NULL ) {  
108 newPtr->nextPtr = \*sPtr;  
109 \*sPtr = newPtr;  
110 } // end if  
111 else { // insert new node between previousPtr and currentPtr  
112 previousPtr->nextPtr = newPtr;  
113 newPtr->nextPtr = currentPtr;  
114 } // end else  
115 } // end if  
116 else {  
117 printf( "%c not inserted. No memory available.\n", value );  
118 } // end else  
119 } // end function insert  
120  
121 // delete a list element  
122 char delete( ListNodePtr \*sPtr, char value )  
123 {  
124 ListNodePtr previousPtr; // pointer to previous node in list  
125 ListNodePtr currentPtr; // pointer to current node in list  
126 ListNodePtr tempPtr; // temporary node pointer  
127  
128 // delete first node  
129 if ( value == ( \*sPtr )->data ) {  
130 tempPtr = \*sPtr; // hold onto node being removed  
131 \*sPtr = ( \*sPtr )->nextPtr; // de-thread the node  
132 free( tempPtr ); // free the de-threaded node  
133 return value;  
134 } // end if  
135 else {  
136 previousPtr = \*sPtr;  
137 currentPtr = ( \*sPtr )->nextPtr;  
138  
139 // loop to find the correct location in the list  
140 while ( currentPtr != NULL && currentPtr->data != value ) {  
141 previousPtr = currentPtr; // walk to ...  
142 currentPtr = currentPtr->nextPtr; // ... next node  
143 } // end while  
144  
145 // delete node at currentPtr  
146 if ( currentPtr != NULL ) {  
147 tempPtr = currentPtr;  
148 previousPtr->nextPtr = currentPtr->nextPtr;  
149  
150 return value;  
151 } // end if  
152 } // end else

153  
154 return '\0';  
155 } // end function delete  
156  
157 // return 1 if the list is empty, 0 otherwise  
158 int isEmpty( ListNodePtr sPtr )  
159 {  
160 return sPtr == NULL;  
161 } // end function isEmpty  
162  
163 // print the list  
164 void printList( ListNodePtr currentPtr )  
165 {  
166 // if list is empty  
167 if ( isEmpty( currentPtr ) ) {  
168 puts( "List is empty.\n" );  
169 } // end if  
170 else {  
171 puts( "The list is:" );  
172  
173 // while not the end of the list  
174 while ( currentPtr != NULL ) {  
175 printf( "%c --> ", currentPtr->data );  
176 currentPtr = currentPtr->nextPtr;  
177 } // end while  
178  
179 puts( "NULL\n" );  
180 } // end else  
181 } // end function printList

Hình 12.3 Chèn và xóa các nút trong danh sách.

Nhập lựa chọn của bạn:

1 để chèn một phần tử vào danh sách.

2 để xóa một phần tử khỏi danh sách.

3 để kết thúc.

? 1

Nhập một ký tự: B

Danh sách là:

B -> KHÔNG

? 1

Nhập một ký tự: A

Danh sách là:

A -> B -> NULL

? 1

Nhập một ký tự: C

Danh sách là:

A -> B -> C -> NULL

? 2

Nhập ký tự cần xóa :D

D không tìm thấy.

? 2

Nhập ký tự cần xóa: B

B đã xóa.

Danh sách là:

A -> C -> NULL

? 2

Nhập ký tự cần xóa: C

Đã xóa C.

Danh sách là:

Một -> KHÔNG

? 2

Nhập ký tự cần xóa: A

A đã xóa.

Danh sách trống.

? 4

Lựa chọn không hợp lệ.

Nhập lựa chọn của bạn:

1 để chèn một phần tử vào danh sách.

2 để xóa một phần tử khỏi danh sách.

3 để kết thúc.

? 3

Kết thúc chạy.

Các chức năng chính của danh sách liên kết là chèn (dòng 85–119) và xóa (dòng 122–

155). Hàm isEmpty (dòng 158–161) được gọi là hàm vị ngữ—nó không làm thay đổi

liệt kê theo bất kỳ cách nào; thay vào đó, nó xác định xem danh sách có trống không (nghĩa là con trỏ tới nút đầu tiên

của danh sách là NULL). Nếu danh sách trống, 1 được trả về; nếu không, 0 được trả lại. [Lưu ý: Nếu bạn đang

sử dụng trình biên dịch phù hợp với tiêu chuẩn C, bạn có thể sử dụng kiểu \_Bool

Hình 12.4 Đầu ra mẫu cho chương trình của Hình 12.3

**12.4.1 Chức năng chèn**

- Các ký tự được chèn vào danh sách theo thứ tự bảng chữ cái. Hàm insert (dòng 85–119) nhận địa chỉ của danh sách và ký tự được chèn. Địa chỉ của danh sách là cần thiết khi một giá trị được chèn vào đầu danh sách. Cung cấp địa chỉ cho phép danh sách (tức là, con trỏ tới nút đầu tiên của danh sách) sẽ được sửa đổi thông qua một cuộc gọi theo tham chiếu. Bởi vì bản thân danh sách là một con trỏ (đến phần tử đầu tiên của nó), việc chuyển địa chỉ của nó sẽ tạo ra một con trỏ tới một con trỏ (tức là, chuyển hướng kép). Đây là một khái niệm phức tạp và đòi hỏi phải lập trình cẩn thận. Các bước để chèn một ký tự vào danh sách như sau (xem Hình 12.5):

1. Tạo một nút bằng cách gọi malloc, gán cho newPtr địa chỉ của nút được cấp phát memory (dòng 91), gán ký tự được chèn vào newPtr->data (dòng 94) và gán NULL cho newPtr->nextPtr (dòng 95).

2. Khởi tạo previousPtr thành NULL (dòng 97) và currentPtr thành \*sPtr (dòng 98)—the con trỏ đến đầu danh sách. Con trỏ previousPtr và currentPtr lưu trữ vị trí của nút trước điểm chèn và nút sau điểm chèn điểm.

3. Trong khi currentPtr không phải là NULL và giá trị được chèn lớn hơn currentPtr->data (dòng 101), hãy gán currentPtr cho previousPtr (dòng 102) và nâng cao currentPtr cho nút tiếp theo trong danh sách (dòng 103). Điều này xác định vị trí chèn điểm cho giá trị.

4. Nếu previousPtr là NULL (dòng 107), hãy chèn nút mới làm nút đầu tiên trong danh sách (dòng 108–109). Gán \*sPtr cho newPtr->nextPtr (liên kết nút mới trỏ tới nút đầu tiên cũ) và gán newPtr cho \*sPtr (\*sPtr trỏ tới nút mới). Ngược lại, nếu previousPtr không phải là NULL, thì nút mới được chèn vào vị trí (dòng 112–113). Gán newPtr cho previousPtr->nextPtr (nút trước trỏ tới nút mới) và gán currentPtr cho newPtr->nextPtr (liên kết nút mới trỏ đến nút hiện tại)

\* Mẹo ngăn ngừa lỗi 12.3

Gán NULL cho thành viên liên kết của một nút mới. Con trỏ nên được khởi tạo trước chúng được sử dụng

**12.4.2 Chức năng xóa**

- Hàm xóa (dòng 122–155) nhận địa chỉ của con trỏ chỉ đầu danh sách và một ký tự sẽ bị xóa. Các bước để xóa một ký tự khỏi danh sách như sau (xem Hình 12.6):

1. Nếu ký tự cần xóa khớp với ký tự ở nút đầu tiên của danh sách (dòng 129), gán \*sPtr cho tempPtr (tempPtr sẽ được sử dụng để giải phóng bộ nhớ không cần thiết), gán (\*sPtr)->nextPtr cho \*sPtr (\*sPtr bây giờ trỏ đến nút thứ hai trong danh sách), giải phóng bộ nhớ được chỉ bởi tempPtr, và trả lại ký tự ã bị xóa.

2. Nếu không, khởi tạo previousPtr với \*sPtr và khởi tạo currentPtr với (\*sPtr)->nextPtr (dòng 136–137) để chuyển sang nút thứ hai.

3. Trong khi currentPtr không phải là NULL và giá trị cần xóa không bằng currentPtr->data (dòng 140), gán currentPtr cho previousPtr (dòng 141) và gán currentPtr->nextPtr cho currentPtr (dòng 142). Điều này xác định vị trí nhân vật sẽ bị xóa nếu nó có trong danh sách.

4. Nếu currentPtr không phải là NULL (dòng 146), gán currentPtr cho tempPtr (dòng 147), gán currentPtr->nextPtr cho previousPtr->nextPtr (dòng 148), giải phóng nút được trỏ tới bởi tempPtr (dòng 149) và trả lại ký tự đã bị xóa khỏi danh sách (dòng 150). Nếu currentPtr là NULL, trả về ký tự null ('\0') để biểu thị rằng ký tự cần xóa không được tìm thấy trong danh sách (dòng 154)

curentptr

previousPtr

\*sPtr

a

B

E

D

A

newPtr

C

\*sPtr

previousPtr

curentptr

b

B

E

D

A

newPtr

C

Hình 12.5 Chèn một nút theo thứ tự trong một danh sách.

Hình 12.6 minh họa việc xóa một nút khỏi danh sách liên kết. Phần (a) của hình hiển thị danh sách được liên kết sau thao tác chèn trước đó. Phần (b) thể hiện sự phân công lại của phần tử liên kết của previousPtr và việc gán currentPtr cho tempPtr. con trỏ tempPtr được sử dụng để giải phóng bộ nhớ được phân bổ cho nút lưu trữ 'C'. Lưu ý rằng trong các dòng 132 và 149 chúng tôi miễn phí tempPtr. Nhớ lại rằng chúng tôi khuyên bạn nên đặt một con trỏ được giải phóng thành NULL. Chúng tôi không làm điều đó trong hai trường hợp này, vì tempPtr là biến tự động cục bộ và chức năng trả về ngay lập tức.

**12.4.3 Hàm printList**

- Hàm printList (dòng 164–181) nhận một con trỏ tới đầu danh sách làm đối số và gọi con trỏ là currentPtr. Chức năng đầu tiên xác định xem danh sách trống (dòng 167–169) và nếu có, in ra "Danh sách trống." và chấm dứt. Mặt khác, nó sẽ in dữ liệu trong danh sách (dòng 170–180). Trong khi currentPtr không phải là NULL, giá trị của dữ liệu currentPtr->được in bởi hàm và currentPtr->nextPtr được gán cho currentPtr để tiến tới nút tiếp theo. Nếu liên kết trong nút cuối cùng của danh sách không phải là NULL, thuật toán in sẽ cố gắng in qua phần cuối của danh sách và sẽ xảy ra lỗi. Các thuật toán in giống hệt nhau cho các danh sách, ngăn xếp và hàng đợi được liên kết.

+ Bài tập 12.20 yêu cầu bạn thực hiện một hàm đệ quy để in ngược danh sách.

+ Bài tập 12.21 yêu cầu người đọc cài đặt hàm đệ quy tìm kiếm danh sách liên kết cho một mục dữ liệu cụ thể.

**12.5 ngăn xếp**

- Một ngăn xếp có thể được thực hiện như một phiên bản ràng buộc của một danh sách được liên kết. Các nút mới có thể được được thêm vào ngăn xếp và chỉ bị xóa khỏi ngăn xếp ở trên cùng. Vì lý do này, một ngăn xếp được gọi thành cấu trúc dữ liệu nhập sau xuất trước (LIFO). Một ngăn xếp được tham chiếu thông qua một con trỏ tới đỉnh phần tử của ngăn xếp. Thành viên liên kết trong nút cuối cùng của ngăn xếp được đặt thành NULL để biểu thị dưới cùng của ngăn xếp.

- Hình 12.7 minh họa một ngăn xếp có nhiều nút—stackPtr trỏ tới đỉnh của ngăn xếp yếu tố. Ngăn xếp và danh sách liên kết được biểu diễn giống hệt nhau. Sự khác biệt giữa ngăn xếp và danh sách được liên kết là việc thêm và xóa có thể xảy ra ở bất kỳ đâu trong danh sách được liên kết, nhưng chỉ ở trên cùng của một ngăn xếp.

\* Lỗi lập trình phổ biến 12.5

Không đặt liên kết ở nút dưới cùng của ngăn xếp thành NULL có thể dẫn đến lỗi thời gian chạy. Các chức năng chính được sử dụng để thao tác ngăn xếp là đẩy và bật. chức năng đẩy tạo một nút mới và đặt nó lên trên cùng của ngăn xếp. Hàm pop loại bỏ một nút khỏi trên cùng của ngăn xếp, giải phóng bộ nhớ đã được cấp phát cho nút bật lên và trả về giá trị xuất hiện.

- Hình 12.8 (đầu ra được hiển thị trong Hình 12.9) thực hiện một chồng số nguyên đơn giản. Các chương trình cung cấp ba tùy chọn: 1) đẩy một giá trị vào ngăn xếp (đẩy chức năng), 2) bật một giá trị ra khỏi ngăn xếp (chức năng pop) và 3) kết thúc chương trình

1 // Fig. 12.8: fig12\_08.c  
2 // A simple stack program  
3 #include <stdio.h>  
4 #include <stdlib.h>  
5

6 // self-referential structure  
7struct stackNode {  
8int data; // define data as an int  
9struct stackNode \*nextPtr; // stackNode pointer  
10}; // end structure stackNode  
11  
12 typedef struct stackNode StackNode; // synonym for struct stackNode  
13 typedef StackNode \*StackNodePtr; // synonym for StackNode\*  
14  
15 // prototypes  
16 void push( StackNodePtr \*topPtr, int info );  
17 int pop( StackNodePtr \*topPtr );  
18 int isEmpty( StackNodePtr topPtr );  
19 void printStack( StackNodePtr currentPtr );  
20 void instructions( void );  
21  
22 // function main begins program execution  
23 int main( void )  
24 {  
25 StackNodePtr stackPtr = NULL; // points to stack top  
26 unsigned int choice; // user's menu choice  
27 int value; // int input by user  
28  
29 instructions(); // display the menu  
30 printf( "%s", "? " );  
31 scanf( "%u", &choice );  
32  
33 // while user does not enter 3  
34 while ( choice != 3 ) {  
35  
36 switch ( choice ) {  
37 // push value onto stack  
38 case 1:  
39 printf( "%s", "Enter an integer: " );  
40 scanf( "%d", &value );

41  
42 printStack( stackPtr );  
43 break;  
44 // pop value off stack  
45 case 2:  
46 // if stack is not empty  
47 if ( !isEmpty( stackPtr ) ) {

48 printf( "The popped value is %d.\n", );  
49 } // end if  
50  
51 printStack( stackPtr );  
52 break;  
53 default:  
54 puts( "Invalid choice.\n" );  
55 instructions();  
56 break;  
57 } // end switch  
58  
59 printf( "%s", "? " );  
60 scanf( "%u", &choice );  
61 } // end while  
62  
63 puts( "End of run." );  
64 } // end main  
65  
66 // display program instructions to user  
67 void instructions( void )  
68 {  
69 puts( "Enter choice:\n"  
70 "1 to push a value on the stack\n"  
71 "2 to pop a value off the stack\n"  
72 "3 to end program" );  
73 } // end function instructions  
74  
75 // insert a node at the stack top  
76 void push( StackNodePtr \*topPtr, int info )  
77 {  
78 StackNodePtr newPtr; // pointer to new node  
79  
80newPtr = malloc( sizeof( StackNode ) );  
81  
82// insert the node at stack top  
83 if ( newPtr != NULL ) {  
84 newPtr->data = info;  
85 newPtr->nextPtr = \*topPtr;  
86\*topPtr = newPtr;  
87} // end if  
88 else { // no space available  
89 printf( "%d not inserted. No memory available.\n", info );  
90 } // end else  
91 } // end function push  
92  
93 // remove a node from the stack top  
94 int pop( StackNodePtr \*topPtr )  
95 {  
96 StackNodePtr tempPtr; // temporary node pointer  
97 int popValue; // node value  
98  
99tempPtr = \*topPtr;  
100 popValue = ( \*topPtr )->data;

101 \*topPtr = ( \*topPtr )->nextPtr;  
102 free( tempPtr );  
103 return popValue;  
104 } // end function pop  
105  
106 // print the stack  
107 void printStack( StackNodePtr currentPtr )  
108 {  
109 // if stack is empty  
110 if ( currentPtr == NULL ) {  
111 puts( "The stack is empty.\n" );  
112 } // end if  
113 else {  
114 puts( "The stack is:" );  
115  
116 // while not the end of the stack  
117 while ( currentPtr != NULL ) {  
118 printf( "%d --> ", currentPtr->data );  
119 currentPtr = currentPtr->nextPtr;  
120 } // end while  
121  
122 puts( "NULL\n" );  
123 } // end else  
124 } // end function printList  
125  
126 // return 1 if the stack is empty, 0 otherwise  
127 int isEmpty( StackNodePtr topPtr )  
128 {  
129 return topPtr == NULL;  
130 } // end function isEmpty

Hình 12.8 Một chương trình ngăn xếp đơn giản

Nhập lựa chọn:

1 để đẩy một giá trị vào ngăn xếp

2 để bật một giá trị ra khỏi ngăn xếp

3 để kết thúc chương trình

? 1

Nhập một số nguyên: 5

Ngăn xếp là:

5 -> KHÔNG

? 1

Nhập một số nguyên: 6

Ngăn xếp là:

6 -> 5 -> NULL

? 1

Nhập một số nguyên: 4

Ngăn xếp là:

4 -> 6 -> 5 -> NULL

? 2

Giá trị xuất hiện là 4.

Ngăn xếp là:

6 -> 5 -> NULL

? 2

Giá trị hiện ra là 6.

Ngăn xếp là:

5 -> KHÔNG

? 2

Giá trị hiện ra là 5.

Ngăn xếp trống.

? 2

Ngăn xếp trống.

? 4

Lựa chọn không hợp lệ.

Nhập lựa chọn:

1 để đẩy một giá trị vào ngăn xếp

2 để bật một giá trị ra khỏi ngăn xếp

3 để kết thúc chương trình

? 3

Kết thúc chạy.

Hình 12.9 Đầu ra mẫu từ chương trình của Hình 12.8.

**12.5.1 Đẩy chức năng**

- Hàm push (dòng 76–91) đặt một nút mới ở trên cùng của ngăn xếp. Chức năng bao gồm ba bước:

+ 1. Tạo một nút mới bằng cách gọi malloc và gán vị trí của bộ nhớ được cấp phát cho newPtr (dòng 80).

+ 2. Gán cho newPtr->data giá trị được đặt trên ngăn xếp (dòng 84) và gán \*topPtr (con trỏ trên cùng của ngăn xếp) tới newPtr->nextPtr (dòng 85)—thành viên liên kết of newPtr hiện trỏ đến nút trên cùng trước đó.

+ 3. Gán newPtr cho \*topPtr (dòng 86)—\*topPtr bây giờ trỏ tới đỉnh ngăn xếp mới. Các thao tác liên quan đến \*topPtr thay đổi giá trị của stackPtr trong main. Hình 12.10 minh họa chức năng đẩy. Phần (a) của hình hiển thị ngăn xếp và nút mới trước thao tác đẩy. Các mũi tên chấm trong phần (b) minh họa Bước 2 và 3 của thao tác đẩy cho phép nút chứa 12 trở thành đỉnh ngăn xếp mới.

**12.5.2 Bật chức năng**

- Hàm pop (dòng 94–104) loại bỏ một nút khỏi đỉnh ngăn xếp. Chức năng chính xác định xem ngăn xếp có trống hay không trước khi gọi pop. Hoạt động pop bao gồm năm bước:

+ 1. Gán \*topPtr cho tempPtr (dòng 99); tempPtr sẽ được sử dụng để giải phóng những thứ không cần thiết ký ức.

+ 2. Gán (\*topPtr)->data cho popValue (dòng 100) để lưu giá trị vào nút trên cùng.

a

11

7

12

b

11

7

12

Hình 12.10 thao tác đẩy

+ 3. Gán (\*topPtr)->nextPtr cho \*topPtr (dòng 101) để \*topPtr chứa địa chỉ của nút trên cùng mới.

+ 4. Giải phóng bộ nhớ được chỉ bởi tempPtr (dòng 102).

+ 5. Trả lại popValue cho người gọi (dòng 103).

Hình 12.11 minh họa hàm pop. Phần (a) hiển thị ngăn xếp sau lần đẩy trước

hoạt động. Phần (b) hiển thị tempPtr trỏ đến nút đầu tiên của ngăn xếp và topPtr

trỏ đến nút thứ hai của ngăn xếp. Chức năng miễn phí được sử dụng để giải phóng bộ nhớ trỏ

đến bằng tempPtr

hh

11

7

12

hh

11

7

12

Hình 12.11 hoạt động pop.

**12.5.3 Các ứng dụng của ngăn xếp**

- Ngăn xếp có nhiều ứng dụng thú vị. Ví dụ, bất cứ khi nào một lời gọi hàm được thực hiện, hàm được gọi phải biết cách quay lại trình gọi của nó, vì vậy địa chỉ trả về được đẩy lên một ngăn xếp. Nếu một loạt lệnh gọi hàm xảy ra, các giá trị trả về liên tiếp sẽ được đẩy lên ngăn xếp theo thứ tự vào sau, ra trước để mỗi chức năng có thể quay lại trình gọi của nó. Các ngăn xếp hỗ trợ các lệnh gọi hàm đệ quy giống như các lệnh gọi không đệ quy thông thường.

- Ngăn xếp chứa không gian được tạo cho các biến tự động trên mỗi lần gọi hàm. Khi hàm quay trở lại trình gọi của nó, không gian dành cho các biến tự động của hàm đó sẽ được bật ra khỏi ngăn xếp và chương trình không còn biết các biến này nữa. Các ngăn xếp được trình biên dịch sử dụng trong quá trình đánh giá các biểu thức và tạo mã ngôn ngữ máy. Các bài tập khám phá một số ứng dụng của ngăn xếp.

**12.6 Hàng đợi**

- Một cấu trúc dữ liệu phổ biến khác là hàng đợi. Một hàng đợi tương tự như một hàng thanh toán trong một cửa hàng tạp hóa—người xếp hàng đầu tiên được phục vụ trước và những khách hàng khác vào hàng chỉ ở cuối và chờ đợi để được phục vụ. Các nút hàng đợi chỉ được xóa khỏi phần đầu của hàng đợi và chỉ được chèn vào cuối hàng đợi. Vì lý do này, một hàng đợi được giới thiệu thành cấu trúc dữ liệu nhập trước xuất trước (FIFO). Các thao tác chèn và xóa là tương ứng được gọi là enqueue và dequeue.

- Hàng đợi có nhiều ứng dụng trong hệ thống máy tính. Đối với các máy tính chỉ có một bộ xử lý đơn, chỉ có một người dùng tại một thời điểm có thể được phục vụ. Mục nhập cho những người dùng khác là đặt trong một hàng đợi. Mỗi mục dần dần tiến lên phía trước hàng đợi khi người dùng nhận được dịch vụ. Mục ở đầu hàng đợi là mục tiếp theo để nhận dịch vụ.

- Hàng đợi cũng được sử dụng để hỗ trợ in spooling. Một môi trường nhiều người dùng có thể có chỉ có một máy in duy nhất. Nhiều người dùng có thể đang tạo đầu ra để in. Nếu máy in đang bận, các đầu ra khác vẫn có thể được tạo. Chúng được lưu vào đĩa nơi chúng chờ đợi một hàng đợi cho đến khi máy in khả dụng.

- Các gói thông tin cũng chờ đợi trong hàng đợi trong mạng máy tính. Mỗi lần một gói đến một nút mạng, nó phải được định tuyến đến nút tiếp theo trên mạng dọc theo đường đến đích cuối cùng của nó. Nút định tuyến định tuyến một gói tại một thời điểm, do đó, bổ sung các gói được xử lý cho đến khi bộ định tuyến có thể định tuyến chúng. Hình 12.12 minh họa một hàng đợi với một số nút. Lưu ý các con trỏ tới phần đầu của hàng đợi và phần cuối của hàng đợi.

\* Lỗi lập trình phổ biến 12.6

Không đặt liên kết trong nút cuối cùng của hàng đợi thành NULL có thể dẫn đến lỗi thời gian chạy Hình 12.13 (đầu ra trong Hình 12.14) thực hiện thao tác hàng đợi. Chương trình cung cấp một số tùy chọn: chèn một nút vào hàng đợi (hàm enqueue), xóa một nút khỏi hàng đợi (hàm dequeue) và kết thúc chương trình.

H

Q

D

Hình 12.12 Biểu diễn đồ họa hàng đợi

1 // Fig. 12.13: fig12\_13.c  
2 // Operating and maintaining a queue  
3 #include <stdio.h>  
4 #include <stdlib.h>  
5

6// self-referential structure  
7struct queueNode {  
8char data; // define data as a char  
9struct queueNode \*nextPtr; // queueNode pointer  
10 }; // end structure queueNode  
11  
12 typedef struct queueNode QueueNode;  
13 typedef QueueNode \*QueueNodePtr;  
14  
15 // function prototypes  
16 void printQueue( QueueNodePtr currentPtr );  
17 int isEmpty( QueueNodePtr headPtr );  
18 char dequeue( QueueNodePtr \*headPtr, QueueNodePtr \*tailPtr );  
19 void enqueue( QueueNodePtr \*headPtr, QueueNodePtr \*tailPtr,  
20 char value );  
21 void instructions( void );  
22  
23 // function main begins program execution  
24 int main( void )  
25 {  
26 QueueNodePtr headPtr = NULL; // initialize headPtr  
27 QueueNodePtr tailPtr = NULL; // initialize tailPtr  
28 unsigned int choice; // user's menu choice  
29 char item; // char input by user  
30  
31 instructions(); // display the menu  
32 printf( "%s", "? " );  
33 scanf( "%u", &choice );  
34  
35 // while user does not enter 3  
36 while ( choice != 3 ) {  
37  
38 switch( choice ) {  
39 // enqueue value  
40 case 1:  
41 printf( "%s", "Enter a character: " );  
42 scanf( "\n%c", &item );  
43  
44 printQueue( headPtr );  
45 break;  
46 // dequeue value  
47 case 2:  
48 // if queue is not empty  
49 if ( !isEmpty( headPtr ) ) {  
50 item =  
51 printf( "%c has been dequeued.\n", item );  
52 } // end if

53  
54 printQueue( headPtr );  
55 break;  
56 default:  
57 puts( "Invalid choice.\n" );  
58 instructions();  
59 break;  
60 } // end switch  
61  
62 printf( "%s", "? " );  
63 scanf( "%u", &choice );  
64 } // end while  
65  
66 puts( "End of run." );  
67 } // end main  
68  
69 // display program instructions to user  
70 void instructions( void )  
71 {  
72 printf ( "Enter your choice:\n"  
73 " 1 to add an item to the queue\n"  
74 " 2 to remove an item from the queue\n"  
75 " 3 to end\n" );  
76 } // end function instructions  
77  
78 // insert a node in at queue tail  
79 void enqueue( QueueNodePtr \*headPtr, QueueNodePtr \*tailPtr,  
80 char value )  
81 {  
82 QueueNodePtr newPtr; // pointer to new node  
83  
84newPtr = malloc( sizeof( QueueNode ) );  
85  
86 if ( newPtr != NULL ) { // is space available  
87 newPtr->data = value;  
88 newPtr->nextPtr = NULL;  
89  
90 // if empty, insert node at head  
91 if ( isEmpty( \*headPtr ) ) {  
92  
93 } // end if  
94 else {  
95( \*tailPtr )->nextPtr = newPtr;  
96 } // end else  
97  
98 \*tailPtr = newPtr;  
99 } // end if  
100 else {  
101 printf( "%c not inserted. No memory available.\n", value );  
102 } // end else  
103 } // end function enqueue  
104

105 // remove node from queue head  
106 char dequeue( QueueNodePtr \*headPtr, QueueNodePtr \*tailPtr )  
107 {  
108 char value; // node value  
109 QueueNodePtr tempPtr; // temporary node pointer  
110  
111value = ( \*headPtr )->data;  
  
112 tempPtr = \*headPtr;  
113\*headPtr = ( \*headPtr )->nextPtr;  
114  
115 // if queue is empty  
116 if ( \*headPtr == NULL ) {  
117 \*tailPtr = NULL;  
118 } // end if  
119  
120  
121 return value;  
122 } // end function dequeue  
123  
124 // return 1 if the queue is empty, 0 otherwise  
125 int isEmpty( QueueNodePtr headPtr )  
126 {  
127 return headPtr == NULL;  
128 } // end function isEmpty  
129  
130 // print the queue  
131 void printQueue( QueueNodePtr currentPtr )  
132 {  
133 // if queue is empty  
134 if ( currentPtr == NULL ) {  
135 puts( "Queue is empty.\n" );  
136 } // end if  
137 else {  
138 puts( "The queue is:" );  
139  
140 // while not end of queue  
141 while ( currentPtr != NULL ) {  
142 printf( "%c --> ", currentPtr->data );  
143 currentPtr = currentPtr->nextPtr;  
144 } // end while  
145  
146 puts( "NULL\n" );  
147 } // end else  
148 } // end function printQueue

Hình 12.13 Vận hành và duy trì một hàng đợi.

Nhập lựa chọn của bạn:

1 để thêm một mục vào hàng đợi

3 đến hết

? 1

Nhập một ký tự: A

Hàng đợi là:

Một -> KHÔNG

? 1

Nhập một ký tự: B

Hàng đợi là:

A -> B -> NULL

? 1

Nhập một ký tự: C

Hàng đợi là:

A -> B -> C -> NULL

? 2

A đã được dequeued.

Hàng đợi là:

B -> C -> NULL

? 2

B đã được dequeued.

Hàng đợi là:

C -> NULL

? 2

C đã được dequeued.

Hàng đợi trống.

? 2

Hàng đợi trống.

? 4

Lựa chọn không hợp lệ.

Nhập lựa chọn của bạn:

1 để thêm một mục vào hàng đợi

2 để xóa một mục khỏi hàng đợi

3 đến hết

? 3

Kết thúc chạy.

Hình 12.14 Đầu ra mẫu từ chương trình trong Hình 12.13.

**12.6.1 Chức năng xếp hàng**

- Hàm enqueue (dòng 79–103) nhận ba đối số từ chính: địa chỉ của con trỏ tới đầu hàng đợi, địa chỉ của con trỏ tới đuôi hàng đợi và giá trị được đưa vào hàng đợi. Chức năng bao gồm ba bước:

+ 1. Để tạo một nút mới: Gọi malloc, gán vị trí bộ nhớ được cấp phát cho newPtr (dòng 84), gán giá trị được chèn vào hàng đợi cho newPtr->data (dòng 87) và gán NULL cho newPtr->nextPtr (dòng 88).

+ 2. Nếu hàng đợi trống (dòng 91), hãy gán newPtr cho \*headPtr (dòng 92), bởi vì nút mới sẽ là đầu và đuôi của hàng đợi; mặt khác, gán con trỏ newPtr thành (\*tailPtr)->nextPtr (dòng 95), vì nút mới sẽ được đặt sau nút đuôi trước đó.

+ 3. Gán newPtr cho \*tailPtr (dòng 98), vì nút mới là đuôi của hàng đợi. Hình 12.15 minh họa một hoạt động enqueue. Phần (a) hiển thị hàng đợi và cái mới nút trước khi hoạt động. Các mũi tên chấm trong phần (b) minh họa Bước 2 và 3 của chức năng enqueue cho phép thêm một nút mới vào cuối hàng đợi không trống

**12.6.2 Hàm dequeue**

- Hàm dequeue (dòng 106–122) nhận địa chỉ của con trỏ tới đầu hàng đợi và địa chỉ của con trỏ tới đuôi hàng đợi làm đối số và loại bỏ nút đầu tiên khỏi hàng đợi. Hoạt động dequeue bao gồm sáu bước:

+1. Gán (\*headPtr)->data thành giá trị để lưu dữ liệu (dòng 111).

2. Gán \*headPtr cho tempPtr (dòng 112), sẽ được sử dụng để giải phóng những thứ không cần thiết ký ức.

3. Gán (\*headPtr)->nextPtr cho \*headPtr (dòng 113) để \*headPtr ngay bây giờ trỏ đến nút đầu tiên mới trong hàng đợi.

4. Nếu \*headPtr là NULL (dòng 116), hãy gán NULL cho \*tailPtr (dòng 117) vì hàng đợi hiện đang trống.

5. Giải phóng bộ nhớ được chỉ bởi tempPtr (dòng 120).

6. Trả về giá trị cho người gọi (dòng 121).

- Hình 12.16 minh họa hàm dequeue. Phần (a) hiển thị hàng đợi sau hoạt động enqueue trước đó. Phần (b) hiển thị tempPtr trỏ đến nút bị xếp hàng và headPtr trỏ đến nút đầu tiên mới của hàng đợi. Chức năng miễn phí được sử dụng để lấy lại bộ nhớ được trỏ tới bởi tempPtr.

**12.7 Cây**

Danh sách liên kết, ngăn xếp và hàng đợi là cấu trúc dữ liệu tuyến tính. Cây là một cấu trúc dữ liệu hai chiều, phi tuyến tính với các thuộc tính đặc biệt. Các nút cây chứa hai hoặc nhiều liên kết. Cái nào phần thảo luận về cây nhị phân (Hình 12.17)—các cây mà tất cả các nút đều chứa hai liên kết (không có, một hoặc cả hai có thể là NULL). Nút gốc là nút đầu tiên trong cây. Mỗi liên kết trong nút gốc đề cập đến một đứa trẻ. Nút con bên trái là nút đầu tiên trong cây con bên trái và nút con bên phải là nút đầu tiên trong cây con bên phải. Con của một nút được gọi là anh chị em. Nút không có nút con được gọi là nút lá. Các nhà khoa học máy tính thường vẽ cây từ nút gốc trở xuống—hoàn toàn ngược lại với cây cối trong tự nhiên

- Trong phần này, một cây nhị phân đặc biệt được gọi là cây nhị phân tìm kiếm được tạo ra. nhị phân cây tìm kiếm (không có giá trị nút trùng lặp) có đặc điểm là các giá trị ở bất kỳ nút nào bên trái cây con nhỏ hơn giá trị trong nút cha của nó và các giá trị trong bất kỳ cây con bên phải nào là lớn hơn giá trị trong nút cha của nó. Hình 12.18 minh họa một cây tìm kiếm nhị phân với 12 giá trị. Hình dạng của cây tìm kiếm nhị phân tương ứng với một tập hợp dữ liệu có thể khác nhau, tùy thuộc vào thứ tự mà các giá trị được chèn vào cây

\* Lỗi lập trình phổ biến 12.7

Không đặt thành NULL các liên kết trong các nút lá của cây có thể dẫn đến lỗi thời gian chạy.

47

25 77

11 43 65

31 44 68

Hình 12.19 (đầu ra được hiển thị trong Hình 12.20) tạo một cây tìm kiếm nhị phân và duyệt qua nó ba cách—đặt hàng, đặt hàng trước và đặt hàng sau. Chương trình tạo 10 số ngẫu nhiên và chèn từng số vào cây, ngoại trừ các giá trị trùng lặp sẽ bị loại bỏ

1 // Fig. 12.19: fig12\_19.c  
2 // Creating and traversing a binary tree  
3 // preorder, inorder, and postorder  
4 #include <stdio.h>  
5 #include <stdlib.h>  
6 #include <time.h>  
7

8// self-referential structure

9 struct treeNode {  
10 struct treeNode \*leftPtr; // pointer to left subtree  
  
11 int data; // node value  
  
12 struct treeNode \*rightPtr; // pointer to right subtree  
13}; // end structure treeNode  
14  
15 typedef struct treeNode TreeNode; // synonym for struct treeNode  
16 typedef TreeNode \*TreeNodePtr; // synonym for TreeNode\*  
17  
18 // prototypes  
19 void insertNode( TreeNodePtr \*treePtr, int value );  
20 void inOrder( TreeNodePtr treePtr );  
21 void preOrder( TreeNodePtr treePtr );  
22 void postOrder( TreeNodePtr treePtr );  
23

24 // function main begins program execution  
25 int main( void )  
26 {  
27 unsigned int i; // counter to loop from 1-10  
28 int item; // variable to hold random values  
29 TreeNodePtr rootPtr = NULL; // tree initially empty  
30  
31 srand( time( NULL ) );  
32 puts( "The numbers being placed in the tree are:" );  
33  
34 // insert random values between 0 and 14 in the tree  
35 for ( i = 1; i <= 10; ++i ) {  
36 item = rand() % 15;  
37 printf( "%3d", item );  
38insertNode( &rootPtr, item );  
39 } // end for  
40  
41 // traverse the tree preOrder  
42 puts( "\n\nThe preOrder traversal is:" );  
43preOrder( rootPtr );  
44  
45 // traverse the tree inOrder  
46 puts( "\n\nThe inOrder traversal is:" );  
47inOrder( rootPtr )  
48  
49 // traverse the tree postOrder  
50 puts( "\n\nThe postOrder traversal is:" );  
51postOrder( rootPtr );  
52 } // end main  
53  
54 // insert node into tree  
55 void insertNode( TreeNodePtr \*treePtr, int value )  
56 {  
57 // if tree is empty  
58 if ( \*treePtr == NULL ) {  
59 \*treePtr = malloc( sizeof( TreeNode ) );  
60  
61 // if memory was allocated, then assign data  
62 if ( \*treePtr != NULL ) {  
63 ( \*treePtr )->data = value;  
64 ( \*treePtr )->leftPtr = NULL;  
65 ( \*treePtr )->rightPtr = NULL;  
66 } // end if  
67 else {  
68 printf( "%d not inserted. No memory available.\n", value );  
69 } // end else  
70 } // end if  
71 else { // tree is not empty  
72// data to insert is less than data in current node  
73 if ( value < ( \*treePtr )->data ) {  
74 insertNode( &( ( \*treePtr )->leftPtr ), value );  
75} // end i  
76

77// data to insert is greater than data in current node  
78 else if ( value > ( \*treePtr )->data ) {  
79 insertNode( &( ( \*treePtr )->rightPtr ), value );  
80} // end else if  
81 else { // duplicate data value ignored  
82 printf( "%s", "dup" );  
83 } // end else  
84 } // end else  
85 } // end function insertNode  
86  
87 // begin inorder traversal of tree  
88 void inOrder( TreeNodePtr treePtr )  
89 {  
90// if tree is not empty, then traverse  
91 if ( treePtr != NULL ) {  
92 inOrder( treePtr->leftPtr );  
93 printf( "%3d", treePtr->data );  
94 inOrder( treePtr->rightPtr );  
95} // end if  
96 } // end function inOrder  
97  
98 // begin preorder traversal of tree  
99 void preOrder( TreeNodePtr treePtr )  
100 {  
101// if tree is not empty, then traverse  
102 if ( treePtr != NULL ) {  
103 printf( "%3d", treePtr->data );  
104 preOrder( treePtr->leftPtr );  
105 preOrder( treePtr->rightPtr );  
106} // end if  
107 } // end function preOrder  
108  
109 // begin postorder traversal of tree  
110 void postOrder( TreeNodePtr treePtr )  
111 {  
112// if tree is not empty, then traverse  
113 if ( treePtr != NULL ) {  
114 postOrder( treePtr->leftPtr );  
115 postOrder( treePtr->rightPtr );  
116 printf( "%3d", treePtr->data );  
117} // end if  
118 } // end function postOrder

Các số được đặt trong cây là:

6 7 4 12 7dup 2 2dup 5 7dup 11

Giao dịch đặt hàng trước là:

6 4 2 5 7 12 11

Giao dịch inOrder là:

2 4 5 6 7 11 12

Quá trình truyền tải postOrder là:

2 5 4 11 12 7 6

**12.7.1 Chèn nút chức năng**

- Các hàm được sử dụng trong Hình 12.19 để tạo cây tìm kiếm nhị phân và duyệt qua nó là hàm đệ quy.Hàm insertNode (dòng 55–85) nhận địa chỉ của cây và một số nguyên cần lưu trữ trong cây dưới dạng đối số. Một nút chỉ có thể được chèn dưới dạng một nút lá trong cây tìm kiếm nhị phân.

- Các bước để chèn một nút vào cây tìm kiếm nhị phân như sau:

+ 1. Nếu \*treePtr là NULL (dòng 58), hãy tạo một nút mới (dòng 59). Gọi malloc, gán cấp phát bộ nhớ cho \*treePtr, gán cho (\*treePtr)->data số nguyên cần lưu trữ (dòng 63), gán cho (\*treePtr)->leftPtr và (\*treePtr)->rightPtr giá trị NULL (dòng 64–65 và trả lại quyền điều khiển cho người gọi (cuộc gọi chính hoặc cuộc gọi trước đó để chènNode).

+ 2. Nếu giá trị của \*treePtr không phải là NULL và giá trị được chèn nhỏ hơn (\*treePtr)->data, hàm insertNode được gọi với địa chỉ là (\*treePtr)->leftPtr (dòng 74) để chèn nút vào cây con bên trái của nút được trỏ tới bởi treePtr. Nếu giá trị được chèn lớn hơn (\*treePtr)->data, hàm insertNode được gọi với địa chỉ là (\*treePtr)->rightPtr (dòng 79) để chèn nút vào cây con bên phải của nút được trỏ bởi treePtr.

Mặt khác, các bước đệ quy tiếp tục cho đến khi tìm thấy con trỏ NULL, sau đó Bước 1 được thực thi để chèn nút mới.

**12.7.2 Traversal: Hàm inOrder, preOrder và postOrder**

- Hàm inOrder (dòng 88–96), preOrder (dòng 99–107) và postOrder (dòng 110– 118) mỗi người nhận một cây (nghĩa là con trỏ tới nút gốc của cây) và duyệt qua cây.

- Các bước để duyệt theo thứ tự là:

1. Duyệt cây con bên trái theo thứ tự.

2. Xử lý giá trị trong nút.

3. Duyệt qua cây con bên phải theo thứ tự.

Giá trị trong một nút không được xử lý cho đến khi các giá trị trong cây con bên trái của nó được xử lý. Các giao dịch theo thứ tự của cây trong Hình 12.21 là:

- Giao dịch theo thứ tự của cây tìm kiếm nhị phân in các giá trị nút theo thứ tự tăng dần.

- Quá trình tạo cây tìm kiếm nhị phân thực sự sắp xếp dữ liệu—và do đó quá trình này được gọi là sắp xếp cây nhị phân.

- Các bước để duyệt Đơn đặt hàng trước là:

1. Xử lý giá trị trong nút.

2. Di chuyển cây con bên trái PreOrder.

3. Đi qua thứ tự đặt trước của cây con bên phải.

27

13 42

6 17 33 48

Hình 12.21 Cây tìm kiếm nhị phân có bảy nút.

- Giá trị trong mỗi nút được xử lý khi nút được truy cập. Sau giá trị trong một nút nhất định được xử lý, các giá trị trong cây con bên trái được xử lý, sau đó các giá trị trong cây con bên phải được xử lý. Giao dịch theo thứ tự trước của cây trong Hình 12.21 là

27 13 6 17 42 33 48

- Các bước để duyệt postOrder là:

1. Đi qua cây con bên trái

2. Đi qua cây con bên phải

3. Xử lý giá trị trong nút.

- Giá trị trong mỗi nút không được in cho đến khi giá trị của nút con được in. Đường truyền postOrder của cây trong Hình 12.21 là:

6 17 13 33 48 42 27

**12.7.3 Loại bỏ trùng lặp**

- Cây tìm kiếm nhị phân tạo điều kiện loại bỏ trùng lặp. Khi cây đang được tạo, nỗ lực chèn một giá trị trùng lặp sẽ được nhận ra vì một giá trị trùng lặp sẽ theo cùng một giá trị quyết định “sang trái” hoặc “sang phải” trên mỗi phép so sánh như giá trị ban đầu đã làm. Như vậy, các bản sao cuối cùng sẽ được so sánh với một nút trong cây chứa cùng một giá trị.

- Giá trị trùng lặp có thể bị loại bỏ vào thời điểm này.

**12.7.4 Tìm kiếm cây nhị phân**

- Việc tìm kiếm một cây nhị phân để tìm một giá trị khớp với một giá trị khóa cũng rất nhanh. Nếu cây chặt được đóng gói, mỗi cấp độ chứa khoảng gấp đôi số phần tử so với cấp độ trước đó. Vì vậy, một nhị phân cây tìm kiếm có n phần tử sẽ có tối đa các mức log2n và do đó tối đa so sánh log2n sẽ phải được thực hiện để tìm kết quả phù hợp hoặc để xác định rằng không có trận đấu nào tồn tại. Điều này có nghĩa là, ví dụ, khi tìm kiếm cây tìm kiếm nhị phân 1000 phần tử (được đóng gói chặt chẽ), không cần thực hiện quá 10 phép so sánh vì 210 > 1000. Khi tìm kiếm cây tìm kiếm nhị phân 1.000.000 phần tử (được đóng gói chặt chẽ), không còn nữa hơn 20 phép so sánh cần được thực hiện vì 220 > 1.000.000.

**12.7.5 Các phép toán cây nhị phân khác**

Trong bài tập, các thuật toán được trình bày cho một số thao tác cây nhị phân khác như xóa một mục khỏi cây nhị phân, in cây nhị phân ở định dạng cây hai chiều và thực hiện duyệt theo thứ tự cấp độ của cây nhị phân. Trình duyệt thứ tự cấp độ của một nhị phân cây truy cập từng nút của cây bắt đầu từ cấp nút gốc. Trên mỗi cấp độ của cây, các nút được thăm từ trái sang phải. Các bài tập cây nhị phân khác bao gồm allow nhập cây tìm kiếm nhị phân để chứa các giá trị trùng lặp, chèn các giá trị chuỗi vào cây nhị phân và xác định có bao nhiêu cấp độ được chứa trong cây nhị phân.

**12.8 Lập trình C an toàn**

- Chương 8 của Tiêu chuẩn mã hóa C an toàn CERT

- Chương 8 của Tiêu chuẩn mã hóa C an toàn CERT dành riêng cho các khuyến nghị và quy tắc quản lý bộ nhớ—nhiều quy tắc áp dụng cho việc sử dụng con trỏ và phân bổ bộ nhớ động được trình bày trong chương này. Để biết thêm thông tin, hãy truy cập www.securecoding.cert.org.

+ MEM01-C/MEM30-C: Không nên bỏ khởi tạo con trỏ. Thay vào đó, họ ên được gán NULL hoặc địa chỉ của một mục hợp lệ trong bộ nhớ. Khi bạn sử dụng miễn phí để giải phóng bộ nhớ được cấp phát động, con trỏ được chuyển tới free không được gán một giá trị mới, vì vậy nó vẫn trỏ đến vị trí bộ nhớ nơi bộ nhớ được cấp phát động trước đây. Sử dụng một con trỏ như vậy có thể dẫn đến sự cố chương trình và lỗ hổng bảo mật. Khi bạn giải phóng bộ nhớ được cấp phát động, bạn nên gán ngay cho con trỏ NULL hoặc một địa chỉ hợp lệ. Chúng tôi đã chọn không làm điều này đối với các biến con trỏ cục bộ biến mất ngay lập tức phạm vi sau khi gọi đến free.

+ MEM31-C: Hành vi không xác định xảy ra khi bạn cố gắng sử dụng miễn phí để giải phóng bộ nhớ động đã được giải phóng—điều này được gọi là "kép lỗ hổng miễn phí. Để đảm bảo rằng bạn không cố gắng giải quyết cùng một bộ nhớ nhiều hơn một lần, ngay lập tức đặt một con trỏ thành NULL sau lệnh gọi tới miễn phí—cố gắng giải phóng con trỏ NULL không có tác dụng.

+ MEM32-C: Hàm malloc trả về NULL nếu không thể cấp phát yêu cầu ký ức. Bạn phải luôn đảm bảo rằng malloc không trả về NULL trước đó cố gắng sử dụng con trỏ lưu trữ giá trị trả về của malloc.

**PHẦN 2: CHƯƠNG TRÌNH**

1. Lý do chọn đề tài chương trình order các món ăn trong nhà hàng

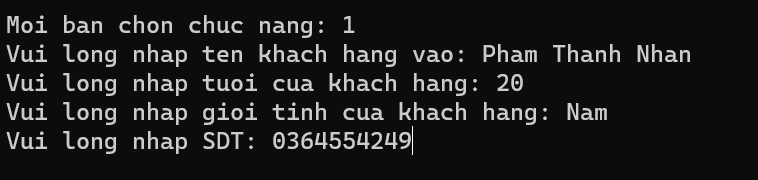
* Hiện trạng trong các quán ăn việc nhân viên đứng đợi khách order gây mất thời gian cũng như việc khách cảm thấy ngại khi có nhân viên đứng chờ.
* Hệ thống order dễ sử dụng với khách
* Có thể tính tổng bill dự kiến của khách hàng sau khi order xong
* Có thể tận dụng các dữ liệu để giúp nhà hàng quản lí được số lượng khách, KPI của quán, biết được món nào bán chạy nhất
* Có mục gợi ý món ăn giúp khách hàng dễ dàng chọn món cũng như biết được thành phần chất dinh dưỡng có trong món ăn giúp cho khách hàng kiên hay di ứng với các chất dinh dưỡng đó.

1. Mô tả bài toán

Bài toán giúp khách hàng có thể thuận tiện chọn món phù hợp với khẩu vị của mình, xem menu các món nướng, lẫu, nước một cách dễ dàng và thuận tiện. Có thể xem bill dự kiến sau khi khách hàng chọn món. Giúp nhà hàng có thể thuận tiện quản lí số lượng khách hàng của quán.

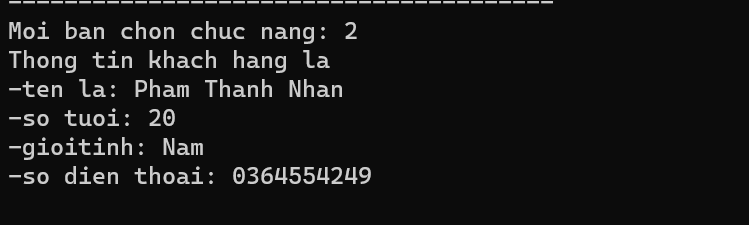
1. Giao diện chương trình

**2.1**Nhập thông tin khách hàng



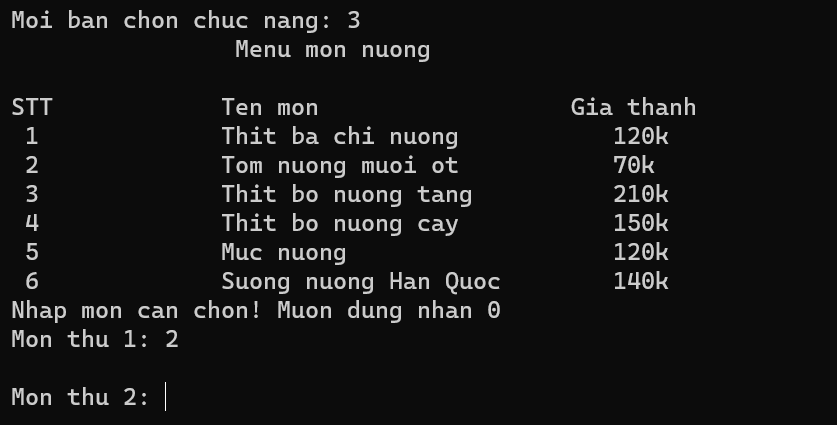
**Hình ảnh nhập thông tin của khách hàng**

**2.2**Xuất thông tin khách hàng



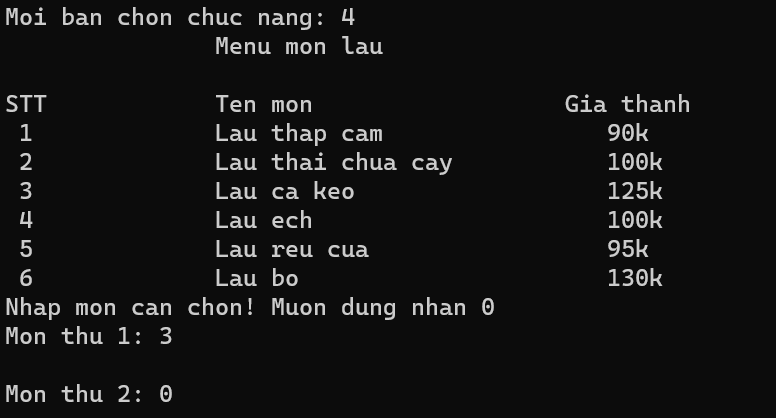
**Hình ảnh xuất thông tin khách hàng**

**2.3**Danh sách menu các món nướng cho khách hàng chọn



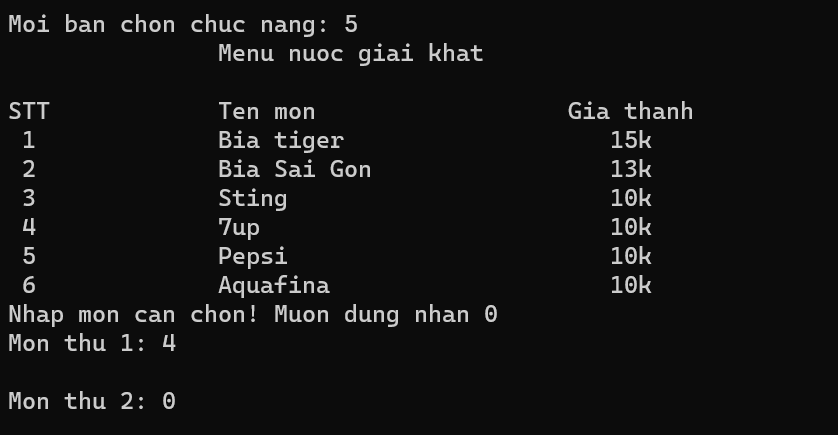
**Hình ảnh menu các món nướng và yêu cầu khách hàng chọn món**

**2.4**Danh sách menu các món lẫu cho khách hàng chọn



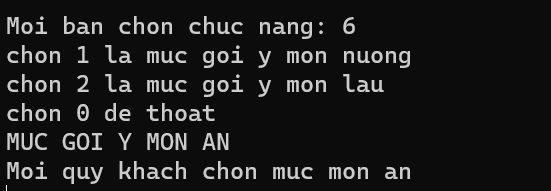
**Hình ảnh menu các món lẫu và yêu cầu khách hàng chọn món**

**2.5**Danh sách menu các món nước và yêu cầu khách hàng chọn món

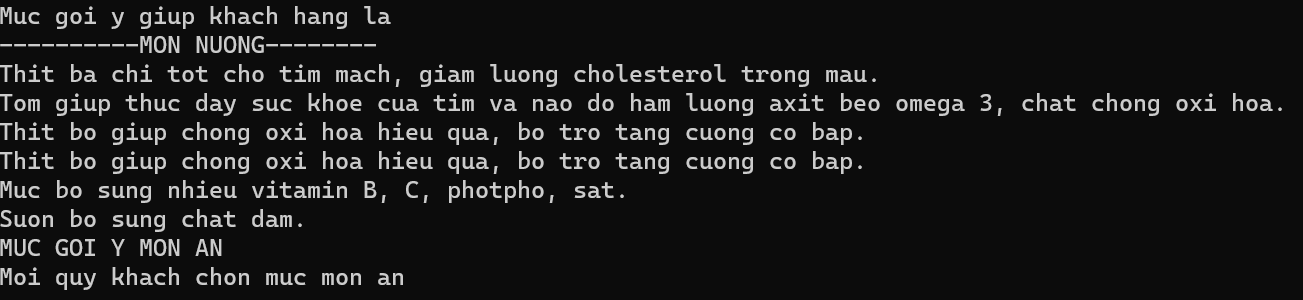


**Hình ảnh menu các món nước và yêu cầu khách hàng chọn món**

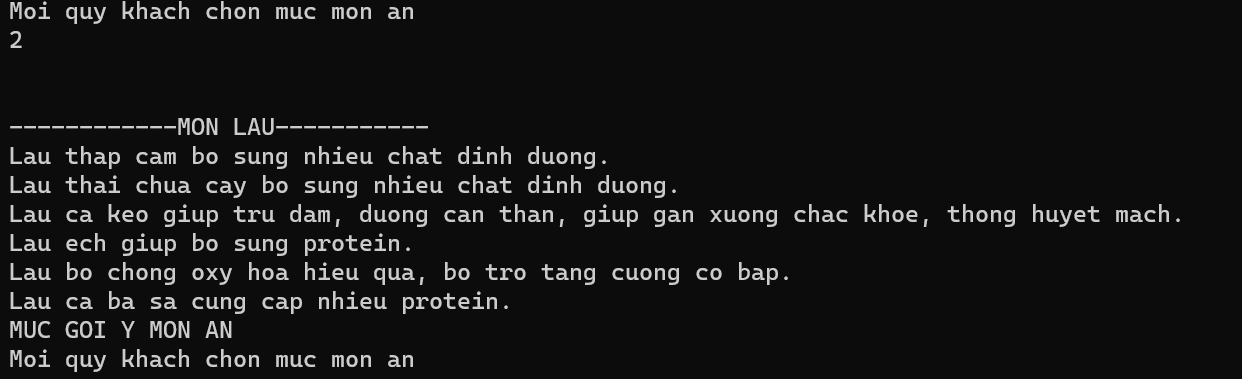
**2.6**Danh sách mục gợi ý chọn món cho khách hàng



**Hình ảnh cho khách hàng lựa chọn mục gợi ý món ăn**

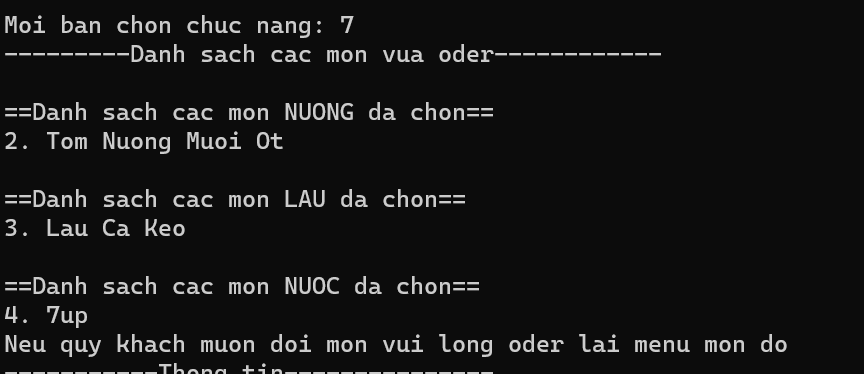


**-...Hình ảnh mục gợi ý các món nướng cho khách hàng**



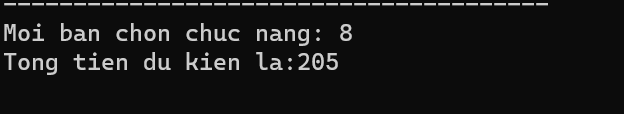
**Hình ảnh mục gợi ý các món lẫu cho khách hàng**

**2.7**Khách hàng xác nhận lại món ăn của mình chọn



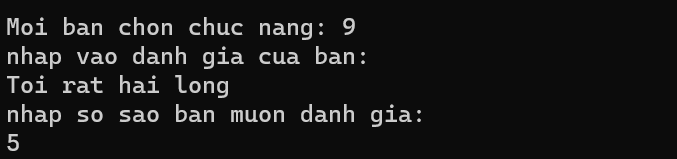
**-..Hình ảnh xác nhận lại món ăn của khách hàng**

**2.8**Tổng hóa đơn dự kiến của khách hàng

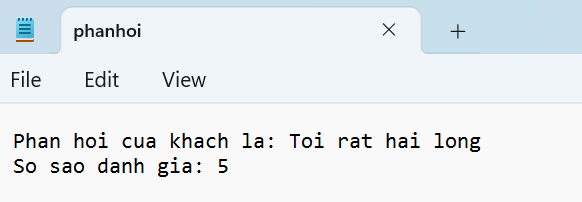


**-...Hình ảnh tổng bill dự kiến của khách hàng**

**2.9**Đánh giá của khách hàng



**-...Hình ảnh đánh giá của khách hàng**



**-..Hình ảnh đánh giá của khách hàng được lưu vào file**