**Danh sách Liên kết**

Danh sách được liên kết là một nhóm các đối tượng dữ liệu trong đó mỗi đối tượng có một con trỏ đến đối tượng tiếp theo trong nhóm. Mọi thứ bạn làm với danh sách liên kết có thể được thực hiện trong bộ nhớ hoặc như một phần của tệp đĩa.  
Đôi khi, việc sắp xếp dữ liệu từ bên ngoài vào chương trình (sử dụng chương trình DOS SORT) là không đủ. Khi người dùng đang nhập dữ liệu, không bao giờ được chấp nhận dừng chương trình, thoát khỏi chương trình, chạy sắp xếp, tạo tệp đã sắp xếp, sau đó khởi động lại chương trình.  
Chúng ta đã quen với việc máy tính thực hiện công việc cho chúng ta và đúng như vậy. Một chương trình không được yêu cầu người dùng làm bất cứ điều gì mà chương trình có thể thực hiện mà không có sự can thiệp của người dùng.Có những lựa chọn thay thế khi dữ liệu phải được sắp xếp. Tuy nhiên, nếu cơ sở dữ liệu lớn, thời gian tạm dừng có thể kéo dài đến mức bạn có thể đi ăn trưa! Ngay cả việc chèn đơn giản vào đầu danh sách cũng có thể tốn thời gian — mọi bản ghi trong cơ sở dữ liệu phải được di chuyển. Kích thước và số lượng các bản ghi này có thể là yếu tố quan trọng.  
Nói chung, việc tìm kiếm tuyến tính một khối dữ liệu hoặc sắp xếp sau khi một mục dữ liệu đã được thêm vào hoặc chỉnh sửa là quá chậm và do đó không đủ. Dữ liệu của chương trình phải được sắp xếp tốt hơn thứ tự mà nó được nhập vào.  
Khi bạn thêm một thành viên mới vào danh sách liên kết, chương trình chỉ cần theo dõi danh sách cho đến khi tìm thấy thành viên sẽ đứng trước thành viên mới và chèn thành viên mới vào thời điểm đó. Khi chương trình phải hiển thị dữ liệu được sắp xếp cho người dùng, chương trình sử dụng con trỏ danh sách liên kết để tìm dữ liệu cần thiết. Vì các liên kết đã được sắp xếp nên hiệu suất của chương trình rất nhanh.

* **Sử dụng bộ nhớ động**

Thường thì bạn phải dựa vào phân bổ bộ nhớ động (bộ nhớ được cấp phát bằng cách sử dụng một trong các chức năng cấp phát bộ nhớ) vì bạn không thể biết người dùng sẽ cung cấp bao nhiêu dữ liệu người dùng. Khi cấp phát bộ nhớ, chương trình phải theo dõi từng khối bộ nhớ, thường là với một danh sách liên kết. Trong tình huống này, có thể (hoặc không) các liên kết được sắp xếp đơn giản theo thứ tự các khối bộ nhớ được cấp phát. Khi một khối bộ nhớ được giải phóng, nó sẽ bị xóa khỏi danh sách liên kết.

* **Danh sách dựa trên đĩa**

Khi bạn tạo danh sách được liên kết dưới dạng tệp dựa trên đĩa, các thành viên của danh sách phải có cùng kích thước. Nếu chương trình của bạn có các thành viên có kích thước khác nhau trong một danh sách được liên kết, giải pháp tốt nhất là sử dụng một liên hợp để tạo một bản ghi có kích thước chính xác. Quy mô của liên minh được xác định bởi thành viên lớn nhất của nó, vì vậy các thành viên sẽ có cùng quy mô.

* **Danh sách Liên kết Đôi**

Trong danh sách liên kết kép, mỗi thành viên có một con trỏ không chỉ đến người kế nhiệm của nó trong danh sách mà còn đến người tiền nhiệm của nó. Chú ý rằng con trỏ đến cuối danh sách là bắt buộc. Con trỏ này là cần thiết để có thể truy cập phần cuối của danh sách.  
Chương trình này được phát triển từ chương trình CDB.C, đã được trình bày trong Chương 8, “Phân bổ bộ nhớ động”. Trong phần này, bạn nhìn vào chương trình và mã quản lý danh sách được liên kết. Đầu tiên, trong đoạn mã sau, là một định nghĩa cấu trúc không cụ thể (vâng, đây là một định nghĩa, không phải một khai báo) tạo ra tên cấu trúc \_CUSTNAME:

struct \_CUSTNAME;

Điều này cho phép \_CUSTNAME được sử dụng trong khai báo cấu trúc như một tập hợp các con trỏ, vì dòng thứ ba và thứ tư trong đoạn mã sau hiển thị:

typedef struct \_CUSTNAME {  
int nRecordType;  
struct \_CUSTNAME \*NextCustomer;   
struct \_CUSTNAME \*PrevCustomer;  
char szName[61];  
char szAddr1[61];   
char szAddr2[61];   
char szCity[26];   
char szState[3];   
int lZip;  
int nRecordNumber;   
double dSalesTotal;  
} CUSTNAME;

Phần này của cấu trúc CUSTNAME khai báo các thành viên trỏ đến thành viên tiếp theo hoặc thành viên trước đó trong danh sách được liên kết.  
Đoạn mã sau cho biết cách con trỏ đến thành viên đầu tiên và cuối cùng trong danh sách liên kết được xác định:

PCUSTNAME FirstCustomer = NULL;  
PCUSTNAME LastCustomer = NULL;

Những dòng này có thể đã được mã hóa là  
struct \_CUSTNAME \* Khách hàng đầu tiên;  
struct \_CUSTNAME \* LastCustomer;

Tôi khuyên bạn nên sử dụng các tên con trỏ được xác định (nếu bạn viết nguyên mẫu cấu trúc của mình như tôi làm) khi bạn tạo cấu trúc typedef.  
Tiếp theo, một vài con trỏ được tạo để chương trình sử dụng khi một thành viên được tạo hoặc được chèn vào danh sách:

Khách hàng PCUSTNAME = NULL;  
PCUSTNAME TempCustomer = NULL;

Phần quan trọng tiếp theo của chương trình là phần thêm bản ghi, phần này được gọi khi người dùng nhập lệnh A. Đầu tiên, chương trình phân bổ một khối bộ nhớ để giữ cấu trúc CUSTNAME bằng cách sử dụng calloc (), khởi tạo bộ nhớ này bằng 0. (Hãy nhớ rằng, malloc () không khởi tạo bộ nhớ.) Sau khi bộ nhớ đã được cấp phát, chương trình sẽ nhắc tên được thêm vào:

case ‘A’:  
case ‘a’:  
Customer = (PCUSTNAME)calloc(sizeof(CUSTNAME),  
INCREMENT\_AMOUNT);  
printf(“Enter name %d: “, ++nRecord);  
gets(szBuffer);  
szBuffer[sizeof(Customer->szName) - 1] = ‘\0’;  
strcpy(Customer->szName, szBuffer);  
if (strlen(Customer->szName) > 0)

Nếu người dùng đã nhập tên (và không chỉ nhấn Return), thành viên này phải được thêm vào danh sách liên kết. Chương trình này sẽ chèn các thành viên vào danh sách theo thứ tự được sắp xếp. Chương trình của bạn có thể chèn thành viên dựa trên một tiêu chí khác, ví dụ: mã ZIP hoặc số khách hàng.  
Không có gì ngăn cản bạn có hai hoặc nhiều tập hợp liên kết. Bạn có thể có danh sách được liên kết dựa trên tên khách hàng, mã ZIP và số khách hàng. Tuy nhiên, mỗi khóa bổ sung sẽ làm chậm hiệu suất của chương trình khi bản ghi được chèn vào và yêu cầu thêm hai con trỏ cho cấu trúc khách hàng (con trỏ trước và con trỏ tiếp theo). Khi bạn tạo một danh sách được liên kết với nhiều hơn một tập hợp các liên kết, chỉ cần coi mỗi tập hợp các liên kết như một danh sách được liên kết riêng biệt.  
Khi một bản ghi được chèn vào danh sách liên kết, có bốn trường hợp có thể xảy ra. Một, danh sách có thể không có gì trong đó, và đây là thành viên ban đầu. Do đó, cả FirstCustomer và LastCustomer phải được khởi tạo cho thành viên này, như sau:  
if (FirstCustomer == NULL){  
printf(“It is first record \n”);  
Customer->NextCustomer = NULL;  
Customer->PrevCustomer = NULL;  
FirstCustomer = Customer;  
LastCustomer = Customer;  
TempCustomer = NULL;  
}  
else{  
TempCustomer = FirstCustomer;  
}  
while (TempCustomer){  
if (nDebug){  
printf(“TESTING FOR ADD: ‘%s’ ‘%s’\n”, Customer->szName, TempCustomer->szName);  
}

Nếu đây không phải là thành viên ban đầu của danh sách, chương trình phải đi xuống danh sách để tìm kiếm điểm chèn chính xác. Bản ghi có thể được chèn vào ba nơi:  
• Ở đầu danh sách, với tư cách là thành viên mới đầu tiên.  
• Ở giữa danh sách.  
• Ở cuối danh sách, với tư cách là thành viên cuối cùng.  
Đây là mã để chèn một thành viên vào đầu danh sách:  
if (strcmp(Customer->szName, TempCustomer->szName) < 0 || TempCustomer == LastCustomer){  
if (strcmp(Customer->szName, TempCustomer->szName) < 0 && TempCustomer == FirstCustomer){  
if (nDebug){  
printf(“Assigning as first\n”);  
}  
Customer->NextCustomer = FirstCustomer;  
FirstCustomer = Customer;  
Customer->PrevCustomer = NULL;  
TempCustomer = Customer->NextCustomer;  
TempCustomer->PrevCustomer = Customer;  
Khi thành viên sẽ là thành viên đầu tiên trong danh sách, chương trình sẽ cập nhật biến FirstCustomer và thành viên cũ đầu tiên. Biến FirstCustomer và con trỏ thành viên cũ của thành viên đầu tiên (-> PrevCustomer) trỏ đến thành viên mới này. Con trỏ thành viên trước đó của thành viên mới (-> PrevCustomer) trỏ đến NULL và con trỏ thành viên tiếp theo của thành viên mới (-> NextCustomer) trỏ đến thành viên cũ đầu tiên (đã trở thành thành viên thứ hai trong danh sách).  
Khi thành viên sẽ là thành viên cuối cùng trong danh sách, biến LastCustomer và thành viên cuối cùng cũ phải được cập nhật:  
}  
else{  
if (strcmp(Customer->szName, TempCustomer->szName) > 0 && TempCustomer == LastCustomer)

{  
Biến LastCustomer và con trỏ thành viên tiếp theo của thành viên cuối cùng cũ (-> NextCustomer) bây giờ sẽ trỏ đến thành viên mới. Con trỏ thành viên tiếp theo của thành viên mới (-> NextCustomer) sẽ trỏ đến NULL và con trỏ thành viên trước đó của thành viên mới (-> PrevCustomer) sẽ trỏ đến thành viên cũ cuối cùng (đã trở thành thành viên tiếp theo trong danh sách) .

Điểm chèn thứ ba là giữa danh sách. Sau đây là mã cho chèn một thành viên vào giữa danh sách liên kết:  
if (nDebug){  
printf(“Assigning as last\n”);  
}  
Customer->PrevCustomer = LastCustomer;  
LastCustomer = Customer;  
Customer->NextCustomer = NULL;  
TempCustomer = Customer->PrevCustomer;  
TempCustomer->NextCustomer = Customer;  
}  
else{  
if (nDebug){  
printf(“Assigning inside list\n”);  
}

Người dùng phải cung cấp cho chương trình các thông tin khác, chẳng hạn như địa chỉ, thành phố và tiểu bang. Chương trình có thể lấy thông tin này sau khi bản ghi đã được chèn vào danh sách. (Tuy nhiên, bạn có thể thay đổi chương trình để thông tin được lấy trước khi chèn bản ghi.)  
Customer->PrevCustomer =TempCustomer->PrevCustomer;  
Customer->NextCustomer = TempCustomer;  
TempCustomer->PrevCustomer = Customer;  
TempCustomer = Customer->PrevCustomer;  
TempCustomer->NextCustomer = Customer;  
}  
}  
Mã để hiển thị các bản ghi trong danh sách theo thứ tự được sắp xếp rất đơn giản vì chương trình duy trì các liên kết đã được sắp xếp.  
TempCustomer = FirstCustomer;  
printf(“Display customers\n”);  
while (TempCustomer){  
if (nDebug){  
printf(“Name ‘%10s’ Me %lp Next %lp Prev %lp\n”,TempCustomer->szName,TempCustomer,TempCustomer->NextCustomer,TempCustomer->PrevCustomer);  
}  
else{  
printf(“Name ‘%10s’ City ‘%10s’ State ‘%2s’ ““ZIP ‘%5.5d’\n”,TempCustomer->szName,TempCustomer->szCity,TempCustomer->szState,TempCustomer->nZip);  
}  
TempCustomer = TempCustomer->NextCustomer;  
}  
break;

Đầu tiên, chương trình nhận một con trỏ đến thành viên đầu tiên của danh sách và lưu con trỏ trong FirstCustomer. Khi có được thành viên đầu tiên của danh sách liên kết, nó sẽ được hiển thị. Thành viên đầu tiên (và mỗi thành viên sau, trừ thành viên cuối cùng) có một con trỏ đến thành viên tiếp theo. Thành viên cuối cùng trong danh sách có một con trỏ đến NULL, kết thúc vòng lặp while (). Ngay trước khi kết thúc vòng lặp while (), con trỏ đến bản ghi khách hàng tiếp theo được gán cho TempCustomer. Điều này cho phép vòng lặp hiển thị tất cả các bản ghi.  
Đầu ra của vòng lặp phụ thuộc vào chế độ gỡ lỗi của chương trình. Trong chế độ gỡ lỗi (được sử dụng khi chương trình được phát triển), các con trỏ được in ra; nếu không, tên và địa chỉ được in.  
Với danh sách liên kết, thật dễ dàng để lấy các bản ghi theo thứ tự đã được sắp xếp. Sử dụng nhiều liên kết, một chương trình có thể lấy các bản ghi dựa trên các tiêu chí khác nhau. Danh sách liên kết kép cho phép bạn truy cập danh sách theo thứ tự tiến hoặc lùi.  
Tuy nhiên, danh sách được liên kết tạo ra một vấn đề. Cách duy nhất để truy cập một thành viên cụ thể trong danh sách là tìm kiếm tuyến tính. Bởi vì các thành viên của danh sách có thể được đặt ngẫu nhiên trong bộ nhớ, quyền truy cập duy nhất mà bạn thường có đối với các thành viên của danh sách là đi theo chuỗi liên kết. Vì vậy, việc tìm kiếm một thành viên ở giữa danh sách không hơn không kém hiệu quả hơn so với việc tìm kiếm một thành viên cụ thể trong một danh sách chưa được sắp xếp. Chương trình của bạn sẽ biết khi nào trường khóa lớn hơn thành viên đang được kiểm tra mà không cần tìm kiếm toàn bộ danh sách. Nhưng bạn thường sẽ xem xét khoảng n / 2 thành viên (trong đó n là số thành viên trong danh sách) để lấy ra một thành viên cụ thể.