資料結構－使用C語言範例程式

**===============**

第2 章

P2-9 範例程式

/\* file name : polynominal.c \*/

/\* 利用陣列表示法做多項式相加 \*/

#include <stdio.h>

#include <stdlib.h>

#define DUMMY -1

void output\_P(int [],int );

void Padd(int [] ,int [] ,int [] );

char compare(int , int );

int main()

{

/\* 多項式的表示方式利用只儲存非零項法

分別儲存每一個非零項的指數及個數，

陣列第一元素放多項式非零項個數。

ex: 下列A多項式有3個非零項，其多項式為 :

5x四次方 + 3x二次方 + 2 \*/

int A[] = {DUMMY, 3, 4, 5, 2, 3, 0, 2};

int B[] = {DUMMY, 3, 3, 6, 2, 2, 0, 1};

int C[13] = {DUMMY};

Padd( A ,B , C ); /\*將A加B放至C \*/

/\*顯示各多項式結果\*/

printf("\nA = ");

output\_P(A, A[1]\*2 +1); /\*A[1]\*2 + 1為陣列A的大小\*/

printf("\nB = ");

output\_P(B, B[1]\*2 +1);

printf("\nC = ");

output\_P(C, C[1]\*2 +1);

printf("\n");

return 0;

}

void Padd(int a[] , int b[], int c[])

{

int p,q,r,m,n;

char result;

m = a[1]; n = b[1];

p = q = r = 2;

while ((p <= 2\*m) && (q <= 2\*n)) {

/\*比較a與b的指數\*/

result = compare ( a[p],b[q] );

switch (result) {

case '=' :

c[r+1] = a[p+1] + b[q+1]; /\*係數相加\*/

if (c[r+1] != 0) {

c[r] = a[p]; /\*指數assign給c \*/

r+=2;

}

p+=2; q+=2; /\*移至下一個指數位置\*/

break;

case '>' :

c[r+1] = a[p+1];

c[r] = a[p];

p+=2; r+= 2;

break;

case '<' :

c[r+1] = b[q+1];

c[r] = b[q];

q+=2; r+= 2;

break;

}

}

/\*將多項式 a 的餘項全部移至 c \*/

while (p <= 2\*m){

c[r+1] = a[p+1];

c[r] = a[p];

p+=2; r+=2;

}

/\*將多項式 b 的餘項全部移至 c \*/

while (q <= 2\*n) {

c[r+1] = b[q+1];

c[r] = b[q];

q+=2; r+=2;

}

c[1] = r/2 - 1; /\*計算c總共有多少非零項\*/

}

char compare(int x, int y)

{

if (x == y)

return '=';

else if (x > y)

return '>';

else

return '<';

}

void output\_P(int p[], int n)

{

int i;

printf("(");

for (i = 1; i <= n; i++)

printf("%3d",p[i]);

printf(" )");

}

P2-14 範例程式

/\* file name : oddMagic.c \*/

/\* 奇數魔術方陣實作 \*/

#include <stdio.h>

#include <stdlib.h>

#define MAX 15 /\* 矩陣最大為15 x 15 \*/

void Magic(void);

int Square[MAX][MAX]; /\* 定義整數矩陣v \*/

int N; /\* 矩陣行列大小變數 \*/

int main()

{

int i,j;

/\* 讀取魔術矩陣的大小 N,N 為奇數且0 < N <= 15 \*/

do {

printf("\n請輸入奇數的矩陣: ");

scanf("%d", &N);

if (N % 2 == 0 || N <= 0 || N > 15)

printf("Should be > 0 and <= 15 odd number");

else

break;

} while (1);

Magic(); /\* 將square 變為 N x N 的魔術矩陣 \*/

/\* 顯示魔術矩陣結果 \*/

printf("\nThe %d\*%d Magic Matrix\n", N, N);

for (i = 1; i <= 5\*N; i++)

printf("-");

printf("\n");

for (i = 0; i < N; i++) {

for (j = 0; j < N; j++)

printf("%5d",Square[i][j]);

printf("\n");

}

return 0;

}

void Magic()

{

int i,j,p,q,key;

/\* 初始化矩陣內容,矩陣全部清 0 \*/

for (i = 0; i < N; i++)

for (j = 0; j < N; j++)

Square[i][j] = 0;

Square[0][(N-1) /2] = 1; /\* 將1放至最上列中間位置 \*/

key = 2;

i = 0;

j = (N-1) / 2; /\* i, j 記錄目前所在位置 \*/

while (key <= N\*N) {

p = (i-1) % N; /\* p, q為下一步位置, i, j各減 1 表示往西北角移動 \*/

q = (j-1) % N;

/\* p < 0 (超出方陣上方)\*/

if (p < 0)

p = N - 1; /\* 則將 p 移至 N-1(最下列) \*/

if (q < 0)

q = N - 1; /\* q < 0 (超出方陣左方) \*/

/\* 則將q 移至N -1(最右行) \*/

if (Square[p][q] != 0) /\* 判斷下一步是否已有數字 \*/

i = (i + 1) % N; /\* 已有則 i 往下 ( 填在原值下方 \*/

else {

i = p; /\* 將下一步位置指定給目前位置 \*/

j = q;

}

Square[i][j] = key;

key++;

}

}

**===============**

第3章

p. 3-4的範例程式

/\* file name : stack.c \*/

/\* 使用堆疊處理資料--新增、刪除、輸出 \*/

#include <stdio.h>

#include <stdlib.h>

#include <stdlib.h>

#define MAX 10

void push\_f(void); /\* 新增函數 \*/

void pop\_f(void); /\* 刪除函數 \*/

void list\_f(void); /\* 輸出函數 \*/

void flushBuffer(void); /\* 清空緩衝區 \*/

char item[MAX][20];

int top = -1;

int main()

{

char option;

while (1) {

printf("\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" <1> insert (push)\n");

printf(" <2> delete (pop)\n");

printf(" <3> list\n");

printf(" <4> quit\n");

printf(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" 請輸入選項: ");

option = getchar();

flushBuffer();

switch (option) {

case '1':

push\_f();

break;

case '2':

pop\_f();

break;

case '3':

list\_f();

break;

case '4':

printf(" 程式結束\n");

exit(0);

default:

printf("\n 選項錯誤!\n 請輸入 1, 2, 3, 或 4\n");

}

}

return 0;

}

void push\_f(void)

{

if (top >= MAX-1) /\* 當堆疊已滿，則顯示錯誤 \*/

printf("\n 堆疊是滿的!\n");

else {

top++;

printf("\n 請輸入一字串: ");

scanf("%s", item[top]);

flushBuffer();

}

}

void pop\_f(void)

{

if (top < 0) /\* 當堆疊沒有資料存在，則顯示錯誤 \*/

printf("\n 堆疊是空的!\n");

else {

printf("\n %s 已被刪除\n", item[top]);

top--;

}

}

void list\_f(void)

{

int count = 0, i;

if (top < 0)

printf("\n\n 堆疊無資料\n");

else {

printf("\n\n 堆疊的資料如下: \n");

printf(" ------------------\n");

for (i = top; i >= 0; i--) {

printf(" %-20s\n", item[i]);

count++;

}

printf(" ------------------\n");

printf(" 共有: %d 字串\n", count);

}

}

void flushBuffer()

{

while (getchar() != '\n')

continue;

}

p. 3-14範例程式

/\* file name: circularQueue.c \*/

/\* 使用環形佇列處理資料--新增、刪除、輸出 \*/

#include <stdio.h>

#include <stdlib.h>

#include <stdlib.h>

#define MAX 10

void enqueue\_f(void); /\* 新增函數 \*/

void dequeue\_f(void); /\* 刪除函數 \*/

void list\_f(void); /\* 輸出函數 \*/

void flushBuffer(void); /\* 清空緩衝區 \*/

char item[MAX][20];

int front = MAX-1, rear = MAX-1, tag = 0;

/\* TAG為記憶FRONT所在是否有儲存資料，

0時為沒有存放資料，1時為有存放資料 \*/

int main(void)

{

char option;

while(1) {

printf("\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" <1> insert\n");

printf(" <2> delete\n");

printf(" <3> list\n");

printf(" <4> quit\n");

printf(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" 請輸入選項: ");

option = getchar();

flushBuffer();

switch (option) {

case '1':

enqueue\_f();

break;

case '2':

dequeue\_f();

break;

case '3':

list\_f();

break;

case '4':

printf(" 程式結束\n");

exit(0);

default:

printf("\n 選項錯誤!\n 請輸入 1, 2, 3, 或 4\n");

}

}

}

void enqueue\_f()

{

if (front == rear && tag == 1) /\* 當佇列已滿，則顯示錯誤 \*/

printf("\n\n 佇列是滿的 !\n");

else {

rear = (rear + 1) % MAX;

printf("\n 請輸入一字串: ");

scanf("%s", item[rear]);

flushBuffer();

if (front == rear)

tag = 1;

}

}

void dequeue\_f()

{

if (front == rear && tag == 0) /\* 當資料沒有資料存在，則顯示錯誤 \*/

printf("\n 佇列是空的!\n");

else {

front = (front + 1) % MAX;

printf("\n %s 已被刪除\n", item[front]);

if (front == rear)

tag = 0;

}

}

void list\_f()

{

int count = 0, i, num;

if (front == rear && tag == 0)

printf("\n 佇列無資料\n");

else {

printf("\n 佇列的資料如下: \n");

printf(" ------------------\n");

i = (front + 1) % MAX;

while (i != rear) {

printf(" %-20s\n", item[i]);

num = ++i % MAX;

i = num;

count++;

}

printf(" %-20s\n", item[i]);

printf(" ------------------\n");

printf(" 共有 %d 個字串\n", ++count);

}

}

void flushBuffer()

{

while (getchar() != '\n')

continue;

}

P3-23 範例程式

/\* file name: infixTopostfix.c \*/

/\* 將數學式子由中序表示法轉為後序表示法 \*/

#include <stdio.h>

#include <stdlib.h>

#define MAX 20

void infix\_to\_postfix(char [], int); /\* 由中序轉後序函數 \*/

int compare(char, char); /\* 比較兩個運算子函數 \*/

/\* 在中序表示法佇列及暫存堆疊中，運算子的優先順序表，其優先值為INDEX/2 \*/

char infix\_priority[9] = {'#', ')', '+', '-', '\*', '/', '^', '('};

char stack\_priority[8] = {'#', '(', '+', '-', '\*', '/', '^'};

int main()

{

int index = -1;

char infix\_q[MAX]; /\* 儲存使用者輸入中序式的佇列 \*/

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" -- Usable operator --\n");

printf(" ^: Exponentiation\n");

printf(" \*: Multiply /: Divide\n");

printf(" +: Add -: Subtraction\n");

printf(" (: Left Brace ): Right Brace\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("請輸入中序表示式: ");

while (infix\_q[index] != '\n')

infix\_q[++index] = getchar();

infix\_q[index] = '#'; /\* 結束時加入 # 為結束符號 \*/

printf("對應的後序表示式如下: ");

infix\_to\_postfix(infix\_q, index);

printf("\n");

return 0;

}

void infix\_to\_postfix(char infix\_q[], int index)

{

int top = 0, ctr, tag = 1 ;

char stack\_t[MAX]; /\* 用以儲存還不必輸出的運算子 \*/

stack\_t[top] = '#'; /\* 於堆疊最底下加入#為結束符號 \*/

for (ctr = 0; ctr <= index; ctr++) {

switch (infix\_q[ctr]) {

/\* 輸入為 )，則輸出堆疊內運算子，直到堆疊內為 ( \*/

case ')':

while (stack\_t[top] != '(')

printf("%c", stack\_t[top--]);

top--;

break;

/\* 輸入為 #，則將堆疊內還未輸出的運算子輸出 \*/

case '#':

while (stack\_t[top] != '#')

printf("%c", stack\_t[top--]);

break;

/\* 輸入為運算子，若小於TOP在堆疊中所指運算子，則將堆疊

的運算子輸出，直到堆疊內的運算子小於輸入的運算子，

若大於等於TOP在堆疊中所指運算子，則

將輸入之運算子置入堆疊 \*/

case '(':

case '^':

case '\*':

case '/':

while (compare(stack\_t[top], infix\_q[ctr]))

printf("%c", stack\_t[top--]);

stack\_t[++top] = infix\_q[ctr];

tag = 1;

break;

case '+':

case '-':

if (tag == 1) { /\* 判斷 (，^，\*，/ 後的 - 號是否表示負的 \*/

stack\_t[++top] = infix\_q[ctr];

tag = 2; /\* 將 tag 設為 2 \*/

}

else {

while (compare(stack\_t[top], infix\_q[ctr]))

printf("%c", stack\_t[top--]);

stack\_t[++top] = infix\_q[ctr];

tag = 1;

}

break;

/\* 輸入為運算元，則直接輸出 \*/

default:

printf("%c", infix\_q[ctr]);

if (tag == 2) /\* 將存放在堆疊的負號輸出 \*/

printf("%c", stack\_t[top--]);

tag = 0;

break;

}

}

}

/\* 比較兩運算子優先權，若輸入運算子小於堆疊中運算子，則傳回值為 1，否則為 0 \*/

int compare(char stack\_o, char infix\_o)

{

int index\_s = 0, index\_i = 0;

while (stack\_priority[index\_s] != stack\_o)

index\_s++;

while (infix\_priority[index\_i] != infix\_o)

index\_i++;

return index\_s/2 >= index\_i/2 ? 1 : 0;

}

**===============**

第4章

p. 4-7的範例程式

/\* sList.c \*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

/\* 函式的原型宣告 \*/

void insert(void);

void del(void);

void modify(void);

void display(void);

void processing(void);

/\* 宣告一個結構的模版 \*/

struct Node {

long int id;

char name[10];

double score;

struct Node \*next;

};

/\* 定義以下的結構全域變數 \*/

struct Node \*head, \*pNode, \*current, \*prev, temp;

/\* 定義一指向 FILE 的指標 \*/

FILE \*fptr;

int main()

{

head = (struct Node \*)malloc(sizeof(struct Node));

head->next = NULL;

/\* 利用一選單讓使用選擇功能項目 \*/

int choice;

do {

printf("鏈結串列的運作選單\n");

printf("1. 加入一節點\n");

printf("2. 刪除一節點\n");

printf("3. 修改一節點\n");

printf("4. 顯示所有節點\n");

printf("5. 結束\n");

printf("請選擇: ");

scanf("%d", &choice);

switch (choice) {

case 1:

insert();

break;

case 2:

del();

break;

case 3:

modify();

break;

case 4:

display();

break;

case 5:

printf("\n程式結束\n");

exit(0);

default:

printf("輸入號碼不正確，請重新輸入\n");

}

printf("\n");

} while(choice != 5);

getchar();

return 0;

}

/\* 按照分數由大至小加入一節點於鏈結串列 \*/

void insert()

{

/\* 利用 malloc() 函式配置記憶體給 n1 \*/

pNode = malloc(sizeof(struct Node));

printf("\n請輸入ID: ");

scanf("%ld", &pNode->id);

printf("請輸入姓名: ");

scanf("%s", pNode->name);

printf("請輸入分數: ");

scanf("%lf", &pNode->score);

/\* 加入一節點於鏈結串列 \*/

current = head->next;

prev = head;

/\* 先判斷鏈結串列是否為空 \*/

if (current == NULL) {

head->next = pNode;

pNode->next = NULL;

}

/\* 若不是空的，則找適當的位置加入於鏈結串列 \*/

else {

while ((current != NULL) && (pNode->score < current->score)) {

prev = current;

current = current->next;

}

prev->next = pNode;

pNode->next = current;

}

}

/\* 刪除某一節點\*/

void del()

{

long int deleteID;

/\* 將 current 指標指向 head 的下一個節點 \*/

current = head->next;

prev = head;

/\* 先判斷鏈結串列是否為空 \*/

if (current != NULL) {

/\* 若不是空的，則找尋欲刪除的節點 \*/

printf("\n請輸入欲刪除的 ID: ");

scanf("%ld", &deleteID);

/\* 找尋欲刪除的節點 \*/

while ((current != NULL) && (current->id != deleteID)) {

prev = current;

current = current->next;

}

/\* 若找到，則將它刪除 \*/

if (current != NULL) {

prev->next = current->next;

current->next = NULL;

printf("ID: %ld 已刪除\n", current->id);

free(current);

}

/\* 若沒有找到，則輸出找不到欲刪除節點的訊息\*/

else {

printf("\n找不到欲刪除的節點\n");

}

}

/\* 若是空的，則輸出鏈結串列是空的訊息 \*/

else {

printf("鏈結串列是空的\n");

}

}

/\* 修改某一節點 \*/

void modify()

{

struct Node \*temp;

long int modifyID;

double modifyScore;

int flag = 0;

printf("\n請輸入欲修改節點的 ID: ");

scanf("%ld", &modifyID);

current = head->next;

prev = head;

/\* 找尋欲修改的節點 \*/

while (current != NULL) {

if (current->id == modifyID) {

printf("目前欲修改節點的資料如下:\n");

printf("%6ld %10s %8.1f\n\n", current->id,

current->name, current->score);

printf("請輸入欲修改的分數: ");

scanf("%lf", &modifyScore);

current->score = modifyScore;

flag = 1;

break;

}

else {

prev = current;

current = current->next;

}

}

/\* 判斷是否有找到欲修改的節點 \*/

if (flag != 0) {

/\* 將 current 的節點指定給 temp \*/

temp = current;

prev->next = current->next;

/\* 將 temp 節點加入於鏈結串列 \*/

current = head->next;

prev = head;

while ((current != NULL) &&

(temp->score < current->score)) {

prev = current;

current = current->next;

}

prev->next = temp;

temp->next = current;

}

else {

printf("找不到欲修改的節點\n");

}

}

/\* 顯示鏈結串列的所有節點資料 \*/

void display()

{

/\* 印出鏈結串列所有節點的資料 \*/

current = head->next;

/\* 檢視鏈結串列是否為空 \*/

if (current != NULL) {

printf("\n%6s %10s %8s\n", "ID", "Name", "Score");

while (current != NULL) {

printf("%6ld %10s %8.1f\n", current->id,

　　　　　　　　　　　　　　　current->name, current->score);

/\* 將指標移到下一個節點 \*/

current = current->next;

}

}

/\* 若是空的，則輸出鏈結串列無資料 \*/

else {

printf("\n鏈結串列無資料\n");

}

}

p. 4-30的範例程式

/\* file name: dList.c \*/

/\* 雙向鍵結串列的加入與刪除 \*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void init\_f(void); /\* 初始化串列，建立一空節點為 head \*/

void insert\_f(void); /\* 插入函數 \*/

void sort\_f(void); /\* 排序函數 \*/

void delete\_f(void); /\* 刪除函數 \*/

void display\_f(void); /\* 輸出函數 \*/

void modify\_f(void); /\* 修改函數 \*/

void flushBuffer(void);

struct Student {

char name[20]; /\* 姓名 \*/

int score; /\* 分數 \*/

struct Student \*llink; /\* 節點左鏈結 \*/

struct Student \*rlink; /\* 節點右鏈結 \*/

};

struct Student \*ptr, \*head, \*tail, \*current, \*prev, \*temp;

int main(void)

{

char option1;

init\_f();

while(1) {

printf("\n雙向鏈結串列的運作選單\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" 1.insert\n");

printf(" 2.delete\n");

printf(" 3.display\n");

printf(" 4.modify\n");

printf(" 5.quit\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("請輸入選項(1-5): ");

option1 = getchar();

flushBuffer();

switch (option1) {

case '1':

insert\_f();

break;

case '2':

delete\_f();

break;

case '3':

display\_f();

break;

case '4':

modify\_f();

break;

case '5':

printf("\n程式結束\n");

exit(0);

}

}

return 0;

}

void init\_f(void) /\* 設一個 head，將左右鏈結皆指向本身 \*/

{

ptr = (struct Student \*) malloc(sizeof(struct Student));

strcpy(ptr->name, "0");

ptr->llink = ptr;

ptr->rlink = ptr;

head = ptr;

tail = ptr;

}

void insert\_f(void)

{

ptr = (struct Student \*) malloc(sizeof(struct Student));

printf("\n 姓名: ");

scanf("%s", ptr->name);

printf(" 成績: ");

scanf("%d", &ptr->score);

flushBuffer();

/\* 以分數高低排列 \*/

prev = head;

current = head->rlink;

while (current != head && current->score > ptr->score) {

prev = current;

current = current->rlink;

}

ptr->rlink = current;

ptr->llink = prev;

prev->rlink = ptr;

current->llink = ptr;

}

void delete\_f(void)

{

char del\_name[20];

printf("\n 欲刪除姓名: ");

scanf("%s", del\_name);

flushBuffer();

prev = head;

current = head->rlink;

while (current != head && strcmp(del\_name, current->name) != 0) {

prev = current;

current = current->rlink;

}

if (current != head) {

prev->rlink = current->rlink;

current->rlink->llink = prev;

printf(" %s 已被刪除\n", del\_name);

free(current);

}

else /\* 找不到資料則顯示錯誤 \*/

printf(" %s 不在串列中\n", del\_name);

}

void modify\_f(void)

{

char n\_temp[20];

printf("\n 欲修改的姓名: ");

scanf("%s", n\_temp);

prev = head;

current = head->rlink;

while (current != head && strcmp(n\_temp, current->name)) {

prev = current;

current = current->rlink;

}

if (current == head) {

printf(" %s 沒有在串列中\n", n\_temp);

}

else {

printf(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" 姓名 : %s\n", current->name);

printf(" 分數: %d\n", current->score);

printf(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" 請輸入新的分數: ");

scanf("%d", &current->score);

flushBuffer();

printf(" %s 已被修改\n", n\_temp);

//將修改的節點加入於適當的位置

//先將 current 的節點指定給 temp，並重新調整左、右節點

temp = current;

prev->rlink = current->rlink;

current->rlink->llink = prev;

//再將 temp 節點加入於串列中

/\* 以分數高低排列 \*/

prev = head;

current = head->rlink;

while (current != head && current->score > temp->score) {

prev = current;

current = current->rlink;

}

temp->rlink = current;

temp->llink = prev;

prev->rlink = temp;

current->llink = temp;

}

}

void display\_f(void)

{

int count = 0;

if (head->rlink == head) {

printf("\n 串列無資料\n");

}

else {

printf("\n");

printf(" NAME SCORE\n");

printf(" ----------------------\n");

current = head->rlink;

while (current != head) {

printf(" %-15s %3d\n", current->name, current->score);

count++;

current = current->rlink;

}

printf(" ----------------------\n");

printf(" 總共有 %d 筆資料\n", count);

}

}

void flushBuffer(void)

{

while(getchar() != '\n')

continue;

}

p. 4-42的範例程式

/\* polyadd.c \*/

/\* 多項式相加--使用降冪排列輸入兩個格式為 ax^b 的多項式相加 \*/

#include <stdio.h>

#include <stdlib.h>

struct poly {

int coef; /\* 多項式係數 \*/

int exp; /\* 多項式指數 \*/

struct poly \*next;

};

void input(struct poly \*, struct poly \*, struct poly \*); /\* 輸入函數 \*/

void poly\_add(void); /\* 多項式相加函數 \*/

void show\_ans(void); /\* 顯示多項式相加結果函數 \*/

void display\_func(struct poly \*, struct poly \*);

void flushBuffer(void);

struct poly \*ptr, \*ans\_h;

struct poly \*head1, \*this\_n1, \*prev1, \*eq\_h1;

struct poly \*head2, \*this\_n2, \*prev2, \*eq\_h2;

int main()

{

head1=(struct poly \*) malloc(sizeof(struct poly));

head1->next = NULL;

head2=(struct poly \*) malloc(sizeof(struct poly));

head2->next = NULL;

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" -- Polynomial format is ax^b --\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Please enter the first equation, terminate at -1\n");

input(prev1, head1, this\_n1);

printf("Please enter the second equation, terminate at -1\n");

input(prev2, head2, this\_n2);

poly\_add();

show\_ans();

return 0;

}

void input(struct poly \*prev, struct poly \*head, struct poly \*this\_n)

{

do {

ptr = (struct poly \*) malloc(sizeof(struct poly));

ptr->next = NULL;

/\* 取得輸入資料 \*/

printf("Coefficient: ");

scanf("%d", &ptr->coef);

flushBuffer();

printf("Exponential: ");

scanf("%d", &ptr->exp);

flushBuffer();

if (ptr->coef == -1 && ptr->exp == -1) {

break;

}

//插入資料

prev = head;

this\_n = head->next;

while ((this\_n != NULL) && (this\_n->exp > ptr->exp)) {

prev = this\_n;

this\_n = this\_n->next;

}

ptr->next = this\_n;

prev->next = ptr;

} while(ptr->exp != -1);

display\_func(this\_n, head);

}

void poly\_add(void)

{

struct poly \*prev;

prev = NULL;

this\_n1 = head1->next;

this\_n2 = head2->next;

/\* 當兩個多項式皆相加完畢則結束 \*/

while (this\_n1 != NULL || this\_n2 != NULL) {

ptr = (struct poly \*) malloc(sizeof(struct poly));

ptr->next = NULL;

/\* 第一個多項式指數大於第二個多項式 \*/

if (this\_n1 != NULL && (this\_n2 == NULL

|| this\_n1->exp > this\_n2->exp)) {

ptr->coef = this\_n1->coef;

ptr->exp = this\_n1->exp;

this\_n1 = this\_n1->next;

}

else {

/\* 第一個多項式指數小於第二個多項式 \*/

if (this\_n1 == NULL || this\_n1->exp < this\_n2->exp) {

ptr->coef = this\_n2->coef;

ptr->exp = this\_n2->exp;

this\_n2 = this\_n2->next;

}

else { /\* 兩個多項式指數相等，進行相加 \*/

ptr->coef = this\_n1->coef + this\_n2->coef;

ptr->exp = this\_n1->exp;

if(this\_n1 != NULL)

this\_n1 = this\_n1->next;

if(this\_n2 != NULL)

this\_n2 = this\_n2->next;

}

}

if(ptr->coef != 0) { /\* 當相加結果不等於0，則放入答案多項式中 \*/

if(ans\_h == NULL)

ans\_h = ptr;

else

prev->next = ptr;

prev = ptr;

}

else free(ptr);

}

}

void show\_ans(void)

{

struct poly \*this\_n;

this\_n = ans\_h;

printf("The equation: ");

while(this\_n != NULL) {

printf("%dx^%d", this\_n->coef, this\_n->exp);

if(this\_n->next != NULL && this\_n->next->coef >= 0)

printf("+");

this\_n = this\_n->next;

}

printf("\n");

}

void display\_func(struct poly \*this\_n, struct poly \*head)

{

int count=0;

if (head->next == NULL) {

printf(" No item in polynominal\n");

}

else {

printf("\n coef exp\n");

printf(" -----------------\n");

this\_n=head->next;

while (this\_n != NULL) {

printf(" %-7d %3d\n", this\_n->coef, this\_n->exp);

count++;

this\_n=this\_n->next;

}

printf(" -----------------\n");

printf(" Total %d item(s) found\n\n", count);

}

}

void flushBuffer()

{

while(getchar() != '\n') {

continue;

}

}

**===============**

第5章

p. 5-4的範例程式

/\*

file name: factorUsingRecursive.c

Description: 利用遞迴呼叫計算 N 階乘

\*/

#include <stdio.h>

#include <ctype.h>

#include <stdlib.h>

/\* 函數原型宣告 \*/

long Factorial(long);

void flushBuffer(void);

int main()

{

char ch;

unsigned long n;

printf("-----Factorial counting Using Recursive----");

do {

printf("\nEnter a number( 0<=n<=12 ) to count n!: ");

scanf("%ld", &n);

flushBuffer();

/\* n 的值在一般系統中超過 13 會產生 overflow 得到不正確的值 \*/

if (n < 0 || n >12)

printf("超出範圍!\n");

else

printf("%ld! = %ld\n", n, Factorial(n) );

printf("Continue(y/n)? ");

ch = toupper(getchar());

} while (ch == 'Y');

return 0;

}

/\* 利用遞迴呼叫自己計算N 階乘\*/

long Factorial(long n)

{

if ( n == 1 || n== 0)

return (1);

else

return( n \* Factorial(n-1));

}

void flushBuffer()

{

while (getchar() != '\n') {

continue;

}

}

p. 5-6的範例程式

/\*

file name: factorUsingIterative.c

Description: Factorial numbers count using iterative

利用迴圈做N階乘的計算

\*/

#include <stdio.h>

#include <ctype.h>

#include <stdlib.h>

/\* 函數原型宣告 \*/

long Factorial(long);

void flushBuffer(void);

int main()

{

char ch;

long n;

printf("-----Factorial counting using Iterative-----");

do {

printf("\nEnter a number(0 <= n <= 12) to count n!: ");

scanf("%ld",&n);

flushBuffer();

if (n < 0 || n > 12)

printf("Input out of range!\n");

else

printf("%ld! = %ld\n", n, Factorial(n));

printf("Continue(y/n)? ");

ch = toupper(getchar());

} while (ch == 'Y');

return 0;

}

long Factorial(long n)

{

long sum = 1;

int i;

if (n == 0 || n ==1) /\* 當 n=0 或 n=1 時, 0!=1, 1!=1 \*/

return (1); /\* 故直接傳回1 \*/

else {

for (i = 2; i<= n; i++) /\* sum 記錄目前階乘之和 \*/

sum \*= i; /\* sum 與 i 相乘之和放回 sum 中 \*/

}

return (sum);

}

void flushBuffer()

{

while (getchar() != '\n') {

continue;

}

}

p. 5-7範例程式

/\*

file name: fibUsingRecursive.c

description: Fibonacci numbers

利用函數遞迴呼叫做費氏數列計算

費氏數列為0,1,1,2,3,5,8,12,21,…

其中某一項為前二項之和,且第0項為0,第1項為1

\*/

#include <stdio.h>

#include <ctype.h>

#include <stdlib.h>

/\* 函數原型宣告 \*/

long Fibonacci(long);

void flushBuffer(void);

int main()

{

char ch;

long n;

printf("-----Fibonacii numbers Using Recursive-----");

do {

printf("\nEnter a number(n >= 0): ");

scanf("%ld", &n);

flushBuffer();

/\* n 值大於0 \*/

if (n < 0)

printf("Number must be > 0\n");

else

printf("Fibonacci(%ld) = %ld\n", n, Fibonacci(n));

printf("Contiune(y/n)? ");

ch = toupper(getchar());

} while (ch == 'Y');

return 0;

}

/\* 利用遞迴函數呼叫本身函數來計算Fibonacci numbers \*/

long Fibonacci(long n)

{

if (n == 0) /\* 第 0 項為 0 \*/

return 0;

else if (n == 1) /\* 第 1 項為 1 \*/

return 1;

else /\* 遞迴呼叫函數第 N 項為 n-1 跟 n-2 項之和 \*/

return(Fibonacci(n-1) + Fibonacci(n-2));

}

void flushBuffer()

{

while (getchar() != '\n') {

continue;

}

}

p. 5-9的範例程式

/\*

file name: fibUsingIterative.c

Description: Fibonacci numbers count using iterative

利用迴圈法計算費氏數列

費氏數列為0,1,1,2,3,5,8,13,21,…

其中某一項為前二項之和,且第0項為0,第1項為1

\*/

#include <stdio.h>

#include <ctype.h>

#include <stdlib.h>

/\* 函數原型宣告 \*/

long Fibonacci(long);

void flushBuffer(void);

int main()

{

char ch;

long n;

printf("-----Fibonacci numbers Using Iterative-----");

do

{

printf("\nEnter a number(n >= 0): ");

scanf("%ld", &n);

flushBuffer();

/\* n 值大於 0 \*/

if ( n < 0 )

printf("Input number must be > 0!\n");

else

printf("Fibonacci(%ld) = %ld\n", n, Fibonacci(n));

printf("Continu(y/n)? ");

ch = toupper(getchar());

} while (ch == 'Y');

return 0;

}

long Fibonacci(long n)

{

long backitem1; /\*前一項值\*/

long backitem2; /\*前二項值\*/

long thisitem = 0; /\*目前項數值\*/

long i;

if (n == 0) /\* 費氏數列第0項為0 \*/

return (0);

else if (n == 1) /\* 第二項為1 \*/

return (1);

else {

backitem2 = 0;

backitem1 = 1;

/\* 利用迴圈將前二項相加後放入目前項 \*/

/\* 之後改變前二項的值至到第n項求得 \*/

for (i = 2; i <= n; i++) {

/\* F(i) = F(i-1) + F(i-2) \*/

thisitem = backitem1 + backitem2;

/\*改變前二項之值\*/

backitem2 = backitem1;

backitem1 = thisitem;

}

return thisitem;

}

}

void flushBuffer()

{

while (getchar() != '\n') {

continue;

}

}

p. 5-15範例程式

/\*

Name: hanoiTower.c

Description: 利用函數遞迴法求河內塔問題之解

Rules:

河內塔問題目的乃在三根柱子中,將n個盤子從

A 柱子搬到 C 柱中, 每次只移動一盤子, 而且必須遵守

每個盤子都比其上面的盤子還要大的原則。

Ans:

河內塔問題的想法必須針對最底端的盤子。

我們必須先把 A 柱子頂端 n-1 個盤子想辦法(借助 C 柱)移至 B 柱子

然後才能將想最底端的盤子移至 C 柱。

此時 C 有最大的盤子, B總共 n-1 個盤子, A 柱則無。

只要再借助 A 柱子，將 B 柱 n-1 個盤子移往 C 柱即可 :

HanoiTower(n-1, A, C, B);

將 A 頂端 n-1 個盤子借助 C 移至 B

HanoiTower(n-1, B, A, C);

將 B 上的 n-1 個盤子借助 A 移至 C

\*/

#include <stdio.h>

#include <stdlib.h>

/\* 函數原型宣告 \*/

void HanoiTower(int, char, char, char);

int main()

{

int n;

char A = 'A', B = 'B', C = 'C';

printf("-----Hanoi Tower Implementaion----\n");

/\*輸入共有幾個盤子在A柱子中\*/

printf("How many disks in A ? ");

scanf("%d", &n);

if (n == 0)

printf("no disk to move\n");

else

HanoiTower(n, A, B, C);

return 0;

}

/\* 遞迴函數呼叫求河內塔之解 \*/

void HanoiTower(int n,char a,char b,char c)

{

if ( n == 1 )

printf("Move disk 1 from %c -> %c\n", a, c);

else {

/\* 將A上n-1個盤子借助C移至B \*/

HanoiTower(n-1,a,c,b);

printf("Move disk %d from %c -> %c\n", n, a, c);

/\* 將B上n-1個盤子借助A移至C \*/

HanoiTower(n-1, b, a, c);

}

}

p. 5-20範例程式

/\*

file name: queen.c

Description: 利用遞迴法求出 8 個皇后問題之解

\*/

#include <stdio.h>

#include <stdlib.h>

#define TRUE 1

#define FALSE 0

#define MAXQUEEN 5

#define ABS(x) ((x>0) ?(x): -(x)) /\* 求x之絕對值 \*/

/\* 存放 5 個皇后之列位置,陣列註標為皇后的行列值 \*/

int queen[MAXQUEEN];

int total\_solution; /\* 計算共有幾組解 \*/

/\* 函數原型宣告 \*/

void place(int);

int attack(int, int);

void output\_solution(void);

int main()

{

place(0); /\* 從第 0 個皇后開始擺放至棋盤 \*/

return 0;

}

void place(int q)

{

int i;

i = 0;

while ( i < MAXQUEEN ) {

if (!attack(i, q)) { /\*皇后未受攻擊\*/

queen[q] = i; /\* 儲存皇后所在的列位置 \*/

/\* 判斷是否找到一組解 \*/

if (q == MAXQUEEN-1)

output\_solution(); /\* 列出此組解 \*/

else

place(q+1); /\* 否則繼續擺下一個皇后 \*/

}

i++;

}

}

/\* 測試在(row, col)上的皇后是否遭受攻擊

若遭受攻擊則傳回值為 1，否則傳回 0 \*/

int attack(int row, int col)

{

int i, atk = FALSE;

int offset\_row, offset\_col;

i = 0;

while (!atk && i < col) {

offset\_col = ABS(i - col);

offset\_row = ABS(queen[i] - row);

/\* 判斷兩皇后是否在同一行,皇后是否在對角線上 \*/

/\* 若皇后同一行或對角線上，則會產生攻擊, atk == TRUE \*/

atk = (queen[i] == row) || (offset\_row == offset\_col);

i++;

}

return atk;

}

/\*列出 5 個皇后之解\*/

void output\_solution()

{

int x,y;

total\_solution += 1;

printf("Solution #%d\n\t", total\_solution);

for ( x = 0; x < MAXQUEEN; x++ ) {

for ( y = 0; y< MAXQUEEN; y++ )

if ( x == queen[y] )

printf("Q");

else

printf("-");

printf("\n\t");

}

printf("\n");

}

**===============**

第7章

P7-7 範例程式

/\* file name: binarySearchTree.c \*/

/\* 利用二元搜尋樹處理資料－載入、儲存、新增、刪除、修改、輸出 \*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

/\* 定義student結構 \*/

struct student {

char name[20]; /\* 學生姓名 \*/

int score; /\* 學生成績 \*/

struct student \*llink; /\* 左子鏈結 \*/

struct student \*rlink; /\* 右子鏈結 \*/

};

void insert\_f(void); /\* 新增函數 \*/

void delete\_f(void); /\* 刪除函數 \*/

void modify\_f(void); /\* 修改函數 \*/

void show\_f(void); /\* 輸出函數 \*/

void process(char [], int); /\* 將資料加入二元搜尋樹 \*/

void removing(char []); /\* 將資料從二元搜尋樹中移除 \*/

struct student \*replace(struct student \*); /\* 尋找替代節點 \*/

void connecting(struct student \*, char); /\* 調整鏈結 \*/

void inorder(struct student \*); /\* 資料以中序法輸出 \*/

void flushBuffer(void);

struct student \*search(char []); /\* 搜尋節點 \*/

struct student \*search\_re\_r(struct student \*); /\* 搜尋右子樹替代節點 \*/

struct student \*search\_re\_l(struct student \*); /\* 搜尋左子樹替代節點 \*/

struct student \*search\_p(struct student \*); /\* 搜尋父節點 \*/

struct student \*root, \*ptr;

int main(void)

{

char option;

while(1) {

puts("");

puts("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

puts(" <1> insert");

puts(" <2> delete");

puts(" <3> modify");

puts(" <4> show");

puts(" <5> quit");

puts("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("Enter your choice: ");

option = getchar();

flushBuffer();

printf("\n");

switch(option) {

case '1':

insert\_f();

break;

case '2':

delete\_f();

break;

case '3':

modify\_f();

break;

case '4':

show\_f();

break;

case '5':

exit(0);

default :

puts("選項錯誤!");

}

}

return 0;

}

/\* 新增函數，新增一筆新的資料 \*/

void insert\_f(void)

{

char name[20];

int score;

puts("=====INSERT DATA=====");

printf("姓名: ");

scanf("%s", name);

printf("分數: ");

scanf("%d", &score);

flushBuffer();

process(name, score);

}

/\* 刪除函數，將資料從二元搜尋樹中刪除 \*/

void delete\_f(void)

{

char name[20];

if (root == NULL) {

puts("二元搜尋樹是空的!");

return;

}

puts("=====DELETE DATA=====");

printf("請輸入欲刪除的姓名: ");

scanf("%s", name);

flushBuffer();

removing(name);

}

/\* 修改資料，修改學生成績 \*/

void modify\_f(void)

{

struct student \*node;

char name[20];

/\* 判斷根節點是否為NULL \*/

if (root == NULL) {

puts("二元搜尋樹是空的!");

return;

}

puts("=====MODIFY DATA===== ");

printf("請輸入欲修改的姓名: ");

scanf("%s", name);

flushBuffer();

if ((node = search(name)) == NULL)

printf("%s 不在二元搜尋中!\n", name);

else {

/\* 列出原資料狀況 \*/

printf("姓名: %s\n", node->name);

printf("分數: %d\n\n", node->score);

printf("請輸入新的分數: ");

scanf("%d", &node->score);

flushBuffer();

printf("%s 已被修改\n", name);

}

}

/\* 輸出函數，將資料輸出至螢幕 \*/

void show\_f(void)

{

/\* 判斷根節點是否為NULL \*/

if (root == NULL) {

puts("二元搜尋樹是空的!");

return;

}

puts("=====SHOW DATA=====");

inorder(root); /\* 以中序法輸出資料 \*/

}

/\* 處理二元搜尋樹，將新增資料加入至二元搜尋樹中 \*/

void process(char name[], int score)

{

struct student \*node, \*prev;

/\* 資料已存在則顯示錯誤 \*/

if (search(name) != NULL) {

printf("%s 已存在!\n", name);

return;

}

ptr = (struct student \*) malloc(sizeof(struct student));

strcpy(ptr->name, name);

ptr->score = score;

ptr->llink = ptr->rlink = NULL;

if (root == NULL) /\* 當根節點為NULL的狀況 \*/

root = ptr;

else { /\* 當根節點不為NULL的狀況 \*/

node = root;

while (node != NULL) { /\* 搜尋資料插入點 \*/

prev = node;

if(strcmp(ptr->name, node->name) < 0)

node = node->llink;

else

node = node->rlink;

}

if (strcmp(ptr->name, prev->name) < 0)

prev->llink = ptr;

else

prev->rlink = ptr;

}

}

/\* 將資料從二元搜尋樹中移除 \*/

void removing(char name[])

{

struct student \*del\_node;

if((del\_node = search(name)) == NULL) /\* 找不到資料則顯示錯誤 \*/

{

printf("%s 不在此二元搜尋樹中!\n", name);

return;

}

/\* 節點不為樹葉節點的狀況 \*/

if (del\_node->llink != NULL || del\_node->rlink != NULL)

del\_node = replace(del\_node);

else /\* 節點為樹葉節點的狀況 \*/

if(del\_node == root)

root = NULL;

else

connecting(del\_node, 'n');

free(del\_node); /\* 釋放記憶體 \*/

printf("%s 已被刪除!\n", name);

}

/\* 尋找刪除非樹葉節點的替代節點 \*/

struct student \*replace(struct student \*node)

{

struct student \*re\_node;

/\* 當右子樹找不到替代節點，會搜尋左子樹是否存在替代節點 \*/

if ((re\_node = search\_re\_r(node->rlink)) == NULL)

re\_node = search\_re\_l(node->llink);

if (re\_node->rlink != NULL) /\* 當替代節點有右子樹存在的狀況 \*/

connecting(re\_node, 'r');

else

if (re\_node->llink != NULL) /\* 當替代節點有左子樹存在的狀況 \*/

connecting(re\_node, 'l');

else /\* 當替代節點為樹葉節點的狀況 \*/

connecting(re\_node, 'n');

strcpy(node->name, re\_node->name);

node->score = re\_node->score;

return re\_node;

}

/\* 調整二元搜尋樹的鏈結，link 為 r 表示處理右鏈結，為 l 表處理左鏈結，

為 m 則將鏈結指向 NULL \*/

void connecting(struct student \*node, char link)

{

struct student \*parent;

parent = search\_p(node); /\* 搜尋父節點 \*/

/\* 節點為父節點左子樹的狀況 \*/

if (strcmp(node->name, parent->name) < 0)

if(link == 'r') /\* link為r \*/

parent->llink = node->rlink;

else /\* link為m \*/

parent->llink = NULL;

else /\* 節點為父節點右子樹的狀況 \*/

if (link == 'l') /\* link為l \*/

parent->rlink = node->llink;

else /\* link為m \*/

parent->rlink = NULL;

}

/\* 以中序法輸出資料，採遞迴方式 \*/

void inorder(struct student \*node)

{

if(node != NULL) {

inorder(node->llink);

printf("%-10s %d\n", node->name, node->score);

inorder(node->rlink);

}

}

/\* 搜尋target所在節點 \*/

struct student \*search(char target[])

{

struct student \*node;

node = root;

while(node != NULL)

{

if (strcmp(target, node->name) == 0)

return node;

else

/\* target小於目前節點，往左搜尋 \*/

if (strcmp(target, node->name) < 0)

node = node->llink;

else /\* target大於目前節點，往右搜尋 \*/

node = node->rlink;

}

return node;

}

/\* 搜尋右子樹替代節點 \*/

struct student \*search\_re\_r(struct student \*node)

{

struct student \*re\_node;

re\_node = node;

while (re\_node != NULL && re\_node->llink != NULL)

re\_node = re\_node->llink;

return re\_node;

}

/\* 搜尋左子樹替代節點 \*/

struct student \*search\_re\_l(struct student \*node)

{

struct student \*re\_node;

re\_node = node;

while (re\_node != NULL && re\_node->rlink != NULL)

re\_node = re\_node->rlink;

return re\_node;

}

/\* 搜尋node的父節點 \*/

struct student \*search\_p(struct student \*node)

{

struct student \*parent;

parent = root;

while (parent != NULL) {

if (strcmp(node->name, parent->name) < 0) {

if (strcmp(node->name, parent->llink->name) == 0)

return parent;

else

parent = parent->llink;

}

else {

if (strcmp(node->name, parent->rlink->name) == 0)

return parent;

else

parent = parent->rlink;

}

}

return NULL;

}

void flushBuffer()

{

while (getchar() != '\n')

continue;

}

**===============**

第8章

P8-7範例程式

/\* file name: heap.c \*/

/\* 利用堆積樹(heap tree)處理會員進出資料--載入、儲存、插入、刪除、輸出 \*/

#include <stdio.h>

#include <stdlib.h>

#define MAX 100 /\* 設定上限 \*/

void insert\_f(void); /\* 插入函數 \*/

void delete\_f(void); /\* 刪除函數 \*/

void display\_f(void); /\* 輸出函數 \*/

void create(int); /\* 建立資料於堆積樹 \*/

void removes(int); /\* 從堆積樹中刪除資料 \*/

void show(char); /\* 印出資料於螢幕 \*/

void adjust\_u(int [], int); /\* 從下而上調整資料 \*/

void adjust\_d(int [], int, int); /\* 從上而下調整資料 \*/

void exchange(int \*, int \*); /\* 交換資料 \*/

int search(int); /\* 搜尋資料 \*/

void flushBuffer(void);

int heap\_tree[MAX]; /\* 堆積樹陣列 \*/

int last\_index = 0; /\* 最後一筆資料的INDEX \*/

int main(void)

{

char option;

do {

printf("\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" <1> login\n");

printf(" <2> logout\n");

printf(" <3> show\n");

printf(" <4> quit\n");

printf(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" Enter your choice: ");

option = getchar();

flushBuffer();

printf("\n");

switch(option) {

case '1':

insert\_f();

break;

case '2':

delete\_f();

break;

case '3':

display\_f();

break;

case '4':

exit(0);

default :

puts("選項錯誤!");

}

} while(option != '4');

return 0;

}

void insert\_f(void)

{

int id\_temp;

if (last\_index >= MAX) { /\* 資料數超過上限，顯示錯誤訊息 \*/

printf("\n Login members are more than %d!!\n", MAX);

printf(" Please wait for a minute!!\n");

}

else {

printf("\n Enter login ID number: ");

scanf("%d", &id\_temp);

flushBuffer();

create(id\_temp); /\* 建立堆積 \*/

printf(" Login successfully!!\n");

}

}

void delete\_f(void)

{

int id\_temp, del\_index;

if (last\_index < 1) { /\* 無資料存在，顯示錯誤訊息 \*/

printf("\n No member to logout!!\n");

printf(" Please check again!!\n");

}

else {

printf("\n Enter logout ID number: ");

scanf("%d", &id\_temp);

flushBuffer();

del\_index = search(id\_temp); /\* 尋找欲刪除資料 \*/

if (del\_index == 0) /\* 沒找到資料，顯示錯誤訊息 \*/

printf(" %d not found!!\n", id\_temp);

else {

removes(del\_index); /\* 刪除資料，並調整堆積樹 \*/

printf(" %d is logout!!\n", id\_temp);

}

}

}

void display\_f(void)

{

char option;

if (last\_index < 1) /\* 無資料存在，顯示錯誤訊息 \*/

printf("\n 堆積中無資料!\n");

else {

printf("\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" <1> increase\n"); /\* 選擇第一項為由小到大排列 \*/

printf(" <2> decrease\n"); /\* 選擇第二項為由大到小排列 \*/

printf(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

do {

printf(" 請選擇: ");

option = getchar();

flushBuffer();

printf("\n");

} while(option != '1' && option != '2');

show(option);

}

}

void create(int id\_temp) /\* ID\_TEMP為新增資料 \*/

{

heap\_tree[++last\_index] = id\_temp; /\* 將資料新增於最後 \*/

adjust\_u(heap\_tree, last\_index); /\* 調整新增資料 \*/

}

void removes(int index\_temp) /\* INDEX\_TEMP為欲刪除資料之INDEX \*/

{ /\* 以最後一筆資料代替刪除資料 \*/

heap\_tree[index\_temp] = heap\_tree[last\_index];

heap\_tree[last\_index--] = 0;

if (last\_index > 1) { /\* 當資料筆數大於1筆，則做調整 \*/

/\* 當替代資料大於其PARENT NODE，則往上調整 \*/

if(heap\_tree[index\_temp] > heap\_tree[index\_temp / 2] && index\_temp > 1)

adjust\_u(heap\_tree, index\_temp);

else /\* 替代資料小於其CHILDEN NODE，則往下調整 \*/

adjust\_d(heap\_tree, index\_temp, last\_index-1);

}

}

void show(char op)

{

int heap\_temp[MAX+1];

int c\_index;

/\* 將堆積樹資料複製到另一個陣列作排序工作 \*/

for (c\_index = 1; c\_index <= last\_index; c\_index++)

heap\_temp[c\_index] = heap\_tree[c\_index];

/\* 將陣列調整為由小到大排列 \*/

for (c\_index = last\_index-1; c\_index > 0; c\_index--) {

exchange(&heap\_temp[1], &heap\_temp[c\_index+1]);

adjust\_d(heap\_temp, 1, c\_index);

}

printf("\n ID number\n");

printf(" =====================\n");

/\* 選擇第一種方式輸出，以遞增方式輸出--使用堆疊

選擇第二種方式輸出，以遞減方式輸出--使用佇列 \*/

switch(op) {

case '1':

for(c\_index = 1; c\_index <= last\_index; c\_index++)

printf("%14d\n", heap\_temp[c\_index]);

break;

case '2':

for(c\_index = last\_index; c\_index > 0; c\_index--)

printf("%14d\n", heap\_temp[c\_index]);

break;

}

printf(" =====================\n");

printf(" Total member: %d\n", last\_index);

}

void adjust\_u(int temp[], int index) /\* INDEX為目前資料在陣列之INDEX \*/

{

while (index > 1) { /\* 將資料往上調整至根為止 \*/

if(temp[index] <= temp[index/2]) /\* 資料調整完畢就跳出，否則交換資料 \*/

break;

else

exchange(&temp[index], &temp[index/2]);

index /= 2;

}

}

/\* INDEX1為目前資料在陣列之INDEX，INDEX2為最後一筆資料在陣列之INDEX \*/

void adjust\_d(int temp[], int index1, int index2)

{ /\* ID\_TEMP記錄目前資料，INDEX\_TEMP則是目前資料之CHILDEN NODE的INDEX \*/

int id\_temp, index\_temp;

id\_temp = temp[index1];

index\_temp = index1 \* 2;

/\* 當比較資料之INDEX不大於最後一筆資料之INDEX，則繼續比較 \*/

while (index\_temp <= index2) {

if ((index\_temp < index2) && (temp[index\_temp] <

temp[index\_temp+1]))

index\_temp++; /\* INDEX\_TEMP記錄目前資料之CHILDEN NODE中較大者 \*/

if (id\_temp >= temp[index\_temp]) /\* 比較完畢則跳出，否則交換資料 \*/

break;

else {

temp[index\_temp/2] = temp[index\_temp];

index\_temp \*= 2;

}

}

temp[index\_temp/2] = id\_temp;

}

/\* 交換傳來之 ID1 及 ID2 儲存之資料 \*/

void exchange(int \*id1, int \*id2)

{

int id\_temp;

id\_temp = \*id1;

\*id1 = \*id2;

\*id2 = id\_temp;

}

int search(int id\_temp) /\* 尋找陣列中ID\_TEMP所在 \*/

{

int c\_index;

for (c\_index = 1; c\_index <= MAX; c\_index++)

if (id\_temp == heap\_tree[c\_index])

return c\_index; /\* 找到則回傳資料在陣列中之INDEX \*/

return 0; /\* 沒找到則回傳0 \*/

}

/\* 清空緩衝區 \*/

void flushBuffer()

{

while (getchar() != '\n')

continue;

}

**===============**

第9章

P9-17範例程式

/\* file name: avlTree.c \*/

/\* 利用 AVL-TREE 處理學生資料--取檔、存檔、插入、刪除、修改、輸出 \*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct student {

char name[20]; /\* 姓名 \*/

int score; /\* 分數 \*/

int bf; /\* 節點BF值 \*/

struct student \*llink, \*rlink; /\* 節點子鏈結 \*/

};

void insert\_f(void); /\* 插入函數 \*/

void delete\_f(void); /\* 刪除函數 \*/

void modify\_f(void); /\* 修改函數 \*/

void list\_f(void); /\* 輸出函數 \*/

void sort\_f(char [], int); /\* 插入檔案後排序 \*/

void inorder(struct student \*); /\* 輸出使用中序追蹤 \*/

void bf\_count(struct student \*); /\* 計算節點BF值 \*/

int height\_count(struct student \*); /\* 計算節點高度 \*/

void pivot\_find(void); /\* 找出pivot所在節點 \*/

int type\_find(void); /\* 找出改善方法 \*/

void type\_ll(void); /\* 使用LL型態 \*/

void type\_rr(void); /\* 使用RR型態 \*/

void type\_lr(void); /\* 使用LR型態 \*/

void type\_rl(void); /\* 使用RL型態 \*/

void flushBuffer(void);

struct student \*ptr, \*root, \*this\_n, \*prev, \*pivot, \*pivot\_prev;

int nodecount = 0; /\* 用來計算node個數 \*/

int main()

{

char option;

while(1) {

printf("\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" <1> insert\n");

printf(" <2> delete\n");

printf(" <3> modify\n");

printf(" <4> list\n");

printf(" <5> exit\n");

printf(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" Enter your choice: ");

option = getchar();

flushBuffer();

switch(option) {

case '1':

insert\_f();

break;

case '2':

delete\_f();

break;

case '3':

modify\_f();

break;

case '4':

list\_f();

break;

case '5':

exit(0);

}

}

return 0;

}

void insert\_f(void)

{

char name\_t[20];

int score\_t;

printf("\n 請輸入姓名: ");

scanf("%s", name\_t);

flushBuffer();

printf(" 請輸入分數: ");

scanf("%d", &score\_t);

flushBuffer();

nodecount++;/\* 將node加1 \*/

sort\_f(name\_t, score\_t); /\* 呼叫 SORT\_F 函數作排序及平衡 \*/

}

void sort\_f(char name\_t[], int score\_t)

{

int op;

this\_n = root;

while((this\_n != NULL) && (strcmp(name\_t, this\_n->name) != 0)) {

/\* 插入資料小於目前位置，則往左移 \*/

if(strcmp(name\_t, this\_n->name) < 0) {

prev = this\_n;

this\_n = this\_n->llink;

}

else { /\* 若大於目前位置，則往右移 \*/

prev = this\_n;

this\_n = this\_n->rlink;

}

}

/\* 找到插入位置，無重覆資料存在 \*/

if(this\_n == NULL || strcmp(name\_t, this\_n->name) != 0) {

ptr = (struct student \*) malloc(sizeof(struct student));

strcpy(ptr->name, name\_t);

ptr->score = score\_t;

ptr->llink = NULL;

ptr->rlink = NULL;

if(root == NULL)

root = ptr; /\* ROOT不存在，則將ROOT指向插入資料 \*/

if(prev != NULL) {

if(strcmp(ptr->name, prev->name) < 0)

prev->llink = ptr;

else

prev->rlink = ptr;

}

bf\_count(root);

pivot\_find();

/\* PIVOT存在，則須改善為AVL-TREE \*/

if(pivot != NULL) {

op = type\_find();

switch(op) {

case 11:

type\_ll();

break;

case 22:

type\_rr();

break;

case 12:

type\_lr();

break;

case 21:

type\_rl();

break;

}

}

bf\_count(root); /\* 重新計算每個節點的BF值 \*/

}

/\* 欲插入資料KEY已存在，則顯示錯誤 \*/

else {

printf(" %s 不存在\n", name\_t);

}

}

void delete\_f(void)

{

struct student \*clear;

char name\_t[20];

int op;

/\* 若根不存在，則顯示錯誤 \*/

if(root == NULL)

printf(" AVL Tree 無資料\n");

else {

printf("\n 輸入欲刪除的姓名: ");

scanf("%s", name\_t);

flushBuffer();

this\_n = root;

while(this\_n != NULL && strcmp(name\_t, this\_n->name) != 0) {

/\* 若刪除資料鍵值小於目前所在資料，則往左子樹 \*/

if(strcmp(name\_t, this\_n->name) < 0) {

prev = this\_n;

this\_n = this\_n->llink;

}

/\* 否則則往右子樹 \*/

else {

prev = this\_n;

this\_n = this\_n->rlink;

}

}

/\* 找到欲刪除資料的狀況 \*/

if(this\_n != NULL) {

/\* 當欲刪除資料底下無左右子樹存在的狀況 \*/

if(this\_n->llink == NULL && this\_n->rlink == NULL) {

clear = this\_n;

if(strcmp(name\_t, root->name) == 0) /\* 欲刪除資料為根 \*/

root = NULL;

else {

/\* 若不為根，則判斷其為左子樹或右子樹 \*/

if(strcmp(name\_t, prev->name) < 0)

prev->llink = NULL;

else

prev->rlink = NULL;

}

free(clear);

}

else { /\* 刪除的節點有左右子樹的存在 \*/

/\* 以左子樹最大點代替刪除資料 \*/

if(this\_n->llink != NULL) {

clear = this\_n->llink;

while(clear->rlink != NULL) {

prev = clear;

clear = clear->rlink;

}

strcpy(this\_n->name, clear->name);

this\_n->score = clear->score;

if(this\_n->llink == clear)

this\_n->llink = clear->llink;

else

prev->rlink = clear->llink;

}

/\* 以右子樹最小點代替刪除資料 \*/

else {

clear = this\_n->rlink;

while(clear->llink != NULL) {

prev = clear;

clear = clear->llink;

}

strcpy(this\_n->name, clear->name);

this\_n->score = clear->score;

if(this\_n->rlink == clear)

this\_n->rlink = clear->rlink;

else

prev->llink = clear->rlink;

}

free(clear);

}

bf\_count(root);

if(root != NULL) { /\* 若根不存在，則無需作平衡改善 \*/

pivot\_find(); /\* 尋找PIVOT所在節點 \*/

while(pivot != NULL) {

op = type\_find();

switch(op) {

case 11:

type\_ll();

break;

case 22:

type\_rr();

break;

case 12:

type\_lr();

break;

case 21:

type\_rl();

break;

} /\* end of switch \*/

bf\_count(root);

pivot\_find(); /\* 重覆檢查 \*/

}

}

nodecount--; /\* 將node減1 \*/

printf(" %s 已被刪除\n", name\_t);

}

/\* 找不到刪除資料，則顯示錯誤 \*/

else

printf(" %s 找不到\n", name\_t);

}

}

void modify\_f(void)

{

char name\_t[20];

printf("\n 請輸入欲更新的姓名: ");

scanf("%s", name\_t);

flushBuffer();

this\_n = root;

/\* 尋找欲更改資料所在節點 \*/

while((this\_n != NULL) && (strcmp(name\_t, this\_n->name) != 0)) {

if(strcmp(name\_t, this\_n->name) < 0)

this\_n = this\_n->llink;

else

this\_n = this\_n->rlink;

}

/\* 若找到欲更改資料，則列出原資料，並要求輸入新的資料 \*/

if(this\_n != NULL) {

printf(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf(" 姓名: %s\n", this\_n->name);

printf(" 分數: %d\n", this\_n->score);

printf(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

printf(" 請輸入新的分數: ");

scanf("%d", &this\_n->score);

flushBuffer();

printf(" 資料更新完成\n");

}

/\* 沒有找到資料則顯示錯誤 \*/

else

printf(" %s 找不到\n", name\_t);

}

void list\_f(void)

{

if(root == NULL)

printf("\n AVL Tree 無資料\n");

else {

printf("\n Name Score\n");

printf(" --------------------\n");

inorder(root); /\* 使用中序法輸出資料 \*/

}

}

/\* 中序使用遞迴 \*/

void inorder(struct student \*trees)

{

if(trees != NULL) {

inorder(trees->llink);

printf(" %-15s %3d\n", trees->name, trees->score);

inorder(trees->rlink);

}

}

void bf\_count(struct student \*trees) /\* 計算BF值，使用後序法逐一計算 \*/

{

if(trees != NULL) {

bf\_count(trees->llink);

bf\_count(trees->rlink);

/\* BF值計算方式為左子樹高減去右子樹高 \*/

trees->bf = height\_count(trees->llink) - height\_count(trees->rlink);

}

}

int height\_count(struct student \*trees)

{

if(trees == NULL) return 0;

else

if(trees->llink == NULL && trees->rlink == NULL)

return 1;

else

return 1 + (height\_count(trees->llink) >

height\_count(trees->rlink) ?

height\_count(trees->llink) :

height\_count(trees->rlink));

}

void pivot\_find(void)

{

int i;

this\_n = root;

pivot = NULL;

for (i =0; i<=nodecount-1; i++) {

/\* 當BF值的絕對值大於等於1，則將PIVOT指向此節點 \*/

if(this\_n->bf < -1 || this\_n->bf > 1) {

pivot = this\_n;

if(pivot != root) pivot\_prev = prev;

printf("pivot name: %s", this\_n->name);

break;

}

if(this\_n->bf > 0) { /\* 左子樹的高度較高 \*/

prev = this\_n;

this\_n = this\_n->llink;

}

else if (this\_n->bf < 0 ) { /\* 右子樹的高度較高 \*/

prev = this\_n;

this\_n = this\_n->rlink;

}

}

}

int type\_find(void)

{

int i, op\_r = 0;

this\_n = pivot;

for(i = 0; i < 2; i++) {

if(this\_n->bf > 0) { /\* 左子樹的高度較高 \*/

this\_n = this\_n->llink;

if(op\_r == 0) op\_r+=10;

else op\_r++;

}

else if (this\_n->bf < 0 ) { /\* 右子樹的高度較高 \*/

this\_n = this\_n->rlink;

if(op\_r == 0) op\_r+=20;

else op\_r+=2;

}

}

/\* 傳回值11、22、12、21分別代表LL、RR、LR、RL型態 \*/

return op\_r;

}

void type\_ll(void) /\* LL型態 \*/

{

struct student \*pivot\_next, \*temp;

pivot\_next = pivot->llink;

temp = pivot\_next->rlink;

pivot\_next->rlink = pivot;

pivot->llink = temp;

if(pivot == root) root = pivot\_next;

else

if(pivot\_prev->llink == pivot)

pivot\_prev->llink = pivot\_next;

else

pivot\_prev->rlink = pivot\_next;

}

void type\_rr(void) /\* RR型態 \*/

{

struct student \*pivot\_next, \*temp;

pivot\_next = pivot->rlink;

temp = pivot\_next->llink;

pivot\_next->llink = pivot;

pivot->rlink = temp;

if(pivot == root) root = pivot\_next;

else

if(pivot\_prev->llink == pivot)

pivot\_prev->llink = pivot\_next;

else

pivot\_prev->rlink = pivot\_next;

}

void type\_lr(void) /\* LR型態 \*/

{

struct student \*pivot\_next, \*temp;

pivot\_next = pivot->llink;

temp = pivot\_next->rlink;

pivot->llink = temp->rlink;

pivot\_next->rlink = temp->llink;

temp->llink = pivot\_next;

temp->rlink = pivot;

if(pivot == root) root = temp;

else

if(pivot\_prev->llink == pivot)

pivot\_prev->llink = temp;

else

pivot\_prev->rlink = temp;

}

void type\_rl(void) /\* RL型態 \*/

{

struct student \*pivot\_next, \*temp;

pivot\_next = pivot->rlink;

temp = pivot\_next->llink;

pivot->rlink = temp->llink;

pivot\_next->llink = temp->rlink;

temp->rlink = pivot\_next;

temp->llink = pivot;

if(pivot == root) root = temp;

else

if(pivot\_prev->llink == pivot)

pivot\_prev->llink = temp;

else

pivot\_prev->rlink = temp;

}

void flushBuffer(void)

{

while(getchar() != '\n')

continue;

}

**===============**

第12章

P12-16 範例程式

/\* file name: dfs.c \*/

/\* 圖形的追蹤: 相鄰串列與蹤向優先搜尋法(DFS)\*/

#include <stdio.h>

#include <stdlib.h>

#define MAX\_V 100 /\*最大節點數\*/

#define TRUE 1

#define FALSE 0

/\*定義資料結構\*/

typedef struct node\_tag {

int vertex;

struct node\_tag \*link;

} Node;

Node \*adjlist[MAX\_V+1]; /\*宣告相鄰串列\*/

int visited[MAX\_V+1]; /\*記錄頂點是否已拜訪\*/

int total\_vertex;

void build\_adjlist(void);

void show\_adjlist(void);

void dfs(int);

Node \*searchlast(Node \*);

int main()

{

build\_adjlist(); /\*以相鄰串列表示圖形\*/

show\_adjlist(); /\*顯示串列之資料\*/

puts("\n------Depth Fisrt Search------");

dfs(1); /\*圖形之蹤向優先搜尋，以頂點1為啟始頂點\*/

printf("\n");

return 0;

}

void build\_adjlist()

{

FILE \*fptr;

Node \*node,\*lastnode;

int vi,vj ,weight;

fptr = fopen("dfs.dat", "r");

if (fptr == NULL) {

perror("dfs.dat");

exit(0);

}

/\* 讀取節點總數 \*/

fscanf(fptr, "%d", &total\_vertex);

for (vi = 1; vi <= total\_vertex; vi++) {

/\*設定陣列及各串列啟始值\*/

visited[vi] = FALSE;

adjlist[vi] = (Node \*)malloc(sizeof(Node));

adjlist[vi]->vertex = vi;

adjlist[vi]->link = NULL;

}

/\* 讀取節點資料 \*/

for (vi = 1; vi <= total\_vertex; vi++)

for (vj = 1; vj <= total\_vertex; vj++) {

fscanf(fptr,"%d",&weight);

/\* 資料檔以相鄰矩陣格式儲存,以1代表相鄰

0 代表不相鄰，將相鄰頂點鏈結在各串列後 \*/

if (weight != 0) {

node = (Node \*)malloc(sizeof(Node));

node->vertex = vj;

node->link = NULL;

lastnode = searchlast(adjlist[vi]);

lastnode->link = node;

}

}

fclose(fptr);

}

/\*顯示各相鄰串列之資料\*/

void show\_adjlist()

{

int index;

Node \*ptr;

puts("Head adjacency nodes");

puts("------------------------------");

for (index = 1; index <= total\_vertex; index++) {

printf("V%-2d ",adjlist[index]->vertex);

ptr = adjlist[index]->link;

while (ptr != NULL) {

printf("--> V%d ",ptr->vertex);

ptr = ptr->link;

}

printf("\n");

}

}

/\*圖形之蹤向優先搜尋\*/

void dfs(int v)

{

Node \*ptr;

int w;

printf("V%d ",adjlist[v]->vertex);

visited[v] = TRUE; /\*設定v頂點為已拜訪過\*/

ptr = adjlist[v]->link; /\*拜訪相鄰頂點\*/

do {

/\* 若頂點尚未走訪，則以此頂點為新啟始點繼續

做蹤向優先搜尋法走訪，否則找與其相鄰的頂點

直到所有相連接的節點都已走訪 \*/

w = ptr->vertex;

if (!visited[w])

dfs(w);

else

ptr = ptr->link;

} while (ptr != NULL);

}

/\*搜尋串列最後節點函數\*/

Node \*searchlast( Node \*linklist )

{

Node \*ptr;

ptr = linklist;

while (ptr->link != NULL)

ptr = ptr->link;

return ptr;

}

輸入檔 dfs.dat

10

0 1 1 0 0 0 0 0 0 0

1 0 0 1 1 0 0 0 0 0

1 0 0 0 0 1 1 0 0 0

0 1 0 0 0 0 0 1 0 0

0 1 0 0 0 0 0 1 0 0

0 0 1 0 0 0 0 0 1 0

0 0 1 0 0 0 0 0 1 0

0 0 0 1 1 0 0 0 0 1

0 0 0 0 0 1 1 0 0 1

0 0 0 0 0 0 0 1 1 0

========

P12-24 範例程式

/\* file name: kruskal.c \*/

/\* 使用 kruskal's 演算法求最小成本擴展樹 \*/

#include <stdio.h>

#include <stdlib.h>

#define MAX\_V 100 /\*最大節點數\*/

#define TRUE 1

#define FALSE 0

typedef struct {

int vertex1;

int vertex2;

int weight;

int edge\_deleted;

} Edge;

typedef struct {

int vertex[MAX\_V];

int edges;

} Graph;

Edge E[MAX\_V];

Graph T;

int total\_vertex;

int total\_edge;

int adjmatrix[MAX\_V][MAX\_V]; /\*store matrix weight\*/

void kruskal(void);

void addEdge(Edge);

void build\_adjmatrix(void);

Edge mincostEdge(void);

int cyclicT(Edge e);

void showEdge(void);

int main()

{

Edge e;

int i , j ,weight;

build\_adjmatrix();

for (i = 1; i <= total\_vertex; i++)

for (j = i+1; j <= total\_vertex; j++) {

weight = adjmatrix[i][j];

if (weight != 0) {

e.vertex1 = i;

e.vertex2 = j;

e.weight = weight;

e.edge\_deleted = FALSE;

addEdge(e);

}

}

showEdge();

/\*init T\*/

for (i = 1; i <= total\_vertex; i++)

T.vertex[i] = 0;

T.edges = 0;

puts("\nMinimum cost spanning tree using Kruskal");

puts("-------------------------------------------");

kruskal();

return 0;

}

void build\_adjmatrix()

{

FILE \*fptr;

int vi,vj;

fptr = fopen("kruskal.dat", "r");

if (fptr == NULL) {

perror("kruskal.dat");

exit(0);

}

/\*讀取節點總數\*/

fscanf(fptr, "%d", &total\_vertex);

for (vi = 1; vi <= total\_vertex; vi++)

for (vj = 1; vj <= total\_vertex; vj++)

fscanf(fptr, "%d", &adjmatrix[vi][vj]);

fclose(fptr);

}

void addEdge(Edge e)

{

E[++total\_edge] = e;

}

void showEdge()

{

int i = 1;

printf("total vertex = %d ",total\_vertex);

printf("total\_edge = %d\n",total\_edge);

while (i <= total\_edge) {

printf("V%d <-----> V%d weight= %d\n",E[i].vertex1,

E[i].vertex2,E[i].weight);

i++;

}

}

Edge mincostEdge()

{

int i , min;

long minweight = 10000000;

for (i = 1; i <= total\_edge; i++) {

if (!E[i].edge\_deleted && E[i].weight < minweight) {

minweight = E[i].weight;

min = i;

}

}

E[min].edge\_deleted = TRUE;

return E[min];

}

void kruskal()

{

Edge e;

int loop = 1;

while (T.edges != total\_vertex - 1) {

e = mincostEdge();

if (!cyclicT(e)) {

printf("%dth min edge : ",loop++);

printf("V%d <-----> V%d weight= %d\n",

e.vertex1, e.vertex2,e.weight);

}

}

}

int cyclicT(Edge e)

{

int v1 = e.vertex1;

int v2 = e.vertex2;

T.vertex[v1]++;

T.vertex[v2]++;

T.edges++;

if (T.vertex[v1] >= 2 && T.vertex[v2] >= 2) {

if(v2 == 2)

return FALSE;

T.vertex[v1]--;

T.vertex[v2]--;

T.edges--;

return TRUE;

}

else

return FALSE;

}

輸入檔kruskal.dat

P12-35 範例程式

/\* file name: shortestPath.c \*/

/\* 利用 Dijkstra 演算法求最短路徑 \*/

#include <stdio.h>

#include <stdlib.h>

#define MAX\_V 100 /\*最大節點數\*/

#define VISITED 1

#define NOTVISITED 0

#define Infinite 1073741823

/\* A[1..N][1..N] 為圖形的相鄰矩陣 \*/

/\* D[i] i=1..N 用來儲存某起始頂點到i節點的最短距離 \*/

/\* S[1..N] 用來記錄頂點是否已經拜訪過 \*/

/\* P[1..N] 用來記錄最近經過的中間節點 \*/

long int A[MAX\_V+1][MAX\_V+1];

long int D[MAX\_V+1];

long int S[MAX\_V+1], P[MAX\_V+1];

int source , sink , N;

int step = 1;

int top = -1; /\*堆疊指標\*/

int Stack[MAX\_V+1]; /\*堆疊空間\*/

void init(void);

int minD(void);

void output\_step(void);

void output\_path(void);

void push(int);

int pop(void);

int main()

{

int t,I;

init();

output\_step();

for (step =2;step <=N; step++) {

/\* minD 傳回一值t使得D[t] 為最小 \*/

t = minD();

S[t] = VISITED;

/\* 找出經過t點會使路徑縮短的節點\*/

for (I=1; I <= N; I++)

if ((S[I] == NOTVISITED) && (D[t]+A[t][I] <= D[I])) {

D[I] = D[t] + A[t][I];

P[I] = t;

}

output\_step();

printf("\n");

}

output\_path();

return 0;

}

void init()

{

FILE \*fptr;

int i,j;

int weight;

fptr = fopen("shortestPath.dat","r");

if (fptr == NULL) {

perror("shortestPath.dat");

exit(1);

}

fscanf(fptr, "%d", &N); /\*讀取圖形節點數\*/

for (i=1; i<=N; i++ )

for (j=1; j<=N; j++)

A[i][j] = Infinite; /\*起始A[1..N][1..N]相鄰矩陣\*/

while (fscanf(fptr,"%d %d %d", &i, &j, &weight) != EOF)

A[i][j] = weight; /\*讀取i節點到j節點的權weight \*/

fclose(fptr);

printf("Enter source node : ");

scanf("%d", &source);

printf("Enter sink node : ");

scanf("%d", &sink);

/\* 起始各陣列初值\*/

for (i = 1; i <= N; i++) {

S[i] = NOTVISITED; /\*各頂點設為尚未拜訪\*/

D[i] = A[source][i]; /\*記錄起始頂點至各頂點最短距離\*/

P[i] = source;

}

S[source] = VISITED; /\*始起節點設為已經走訪\*/

D[source] = 0;

}

int minD()

{

int i,t;

long int minimum = Infinite;

for (i=1; i<=N; i++)

if ((S[i] == NOTVISITED) && D[i] < minimum) {

minimum = D[i];

t = i;

}

return t;

}

/\* 顯示目前的D陣列與P陣列狀況 \*/

void output\_step()

{

int i;

printf("\n Step #%d",step);

printf("\n================================================\n");

for (i=1; i<=N; i++)

printf(" D[%d]", i);

printf("\n");

for (i=1; i<=N; i++)

if (D[i] == Infinite)

printf(" ----");

else

printf("%6ld",D[i]);

printf("\n================================================\n");

for (i=1; i<=N; i++)

printf(" P[%d]", i);

printf("\n");

for (i=1; i<=N;i++)

printf("%6ld", P[i]);

}

/\*顯示最短路徑\*/

void output\_path()

{

int node = sink;

/\*判斷是否起始頂點等於終點或無路徑至終點\*/

if ((sink == source) || (D[sink] == Infinite)) {

printf("\nNode %d has no Path to Node %d", source, sink);

return;

}

printf("\n");

printf(" The shortest Path from V%d to V%d :", source, sink);

printf("\n------------------------------------------\n");

/\*由終點開始將上一次經過的中間節點推入堆疊至到起始節點\*/

printf(" V%d", source);

while (node != source) {

push(node);

node = P[node];

}

while(node != sink) {

node = pop();

printf(" --%ld-->",A[P[node]][node]);

printf("V%d", node);

}

printf("\n Total length : %ld\n", D[sink]);

}

void push(int value)

{

if (top >= MAX\_V) {

printf("Stack overflow!\n");

exit(1);

}

else

Stack[++top] = value;

}

int pop()

{

if (top < 0) {

printf("Stack empty!\n");

exit(1);

}

return Stack[top--];

}

輸入檔 shortestPath.dat

P12-44範例程式

/\* file name: topologicalSort.c \*/

/\* 拓樸排序 \*/

#include <stdio.h>

#include <stdlib.h>

#define MAX\_V 100 /\* 最大節點數 \*/

#define TRUE 1

#define FALSE 0

/\*定義資料結構\*/

typedef struct node\_tag {

int vertex;

struct node\_tag \*link;

} Node;

Node \*adjlist[MAX\_V+1]; /\* 宣告相鄰串列 \*/

int visited[MAX\_V+1]; /\* 記錄頂點是否已拜訪 \*/

int Top\_order[MAX\_V+1];

int N;

int place;

void build\_adjlist(void);

void show\_adjlist(void);

void topological(void);

void top\_sort(int);

Node \*searchlast(Node \*);

int main()

{

int i;

build\_adjlist(); /\* 以相鄰串列表示圖形 \*/

show\_adjlist(); /\* 顯示串列之資料 \*/

topological(); /\* 圖形之蹤向優先搜尋，以頂點 1 為啟始頂點 \*/

puts("\n------Topological sort------");

for (i = 0; i < N; i++)

printf("V%d ", Top\_order[i]);

printf("\n");

return 0;

}

void build\_adjlist()

{

FILE \*fptr;

Node \*node, \*lastnode;

int vi, vj;

fptr = fopen("topologicalSort.dat", "r");

if (fptr == NULL) {

perror("topologicalSort.dat");

exit(0);

}

/\* 讀取節點總數 \*/

fscanf(fptr, "%d", &N);

for (vi = 1; vi <= N; vi++) {

/\* 設定陣列及各串列啟始值 \*/

adjlist[vi] = (Node \*)malloc(sizeof(Node));

adjlist[vi]->vertex = vi;

adjlist[vi]->link = NULL;

}

/\* 讀取節點資料 \*/

while (fscanf(fptr,"%d %d",&vi,&vj) != EOF) {

node = (Node \*)malloc(sizeof(Node));

node->vertex = vj;

node->link = NULL;

if (adjlist[vi]->link == NULL)

adjlist[vi]->link = node;

else {

lastnode = searchlast(adjlist[vi]);

lastnode->link = node;

}

}

fclose(fptr);

}

/\* 顯示各相鄰串列之資料 \*/

void show\_adjlist()

{

int v;

Node \*ptr;

puts("Head adjacency nodes");

puts("--------------------------");

for (v = 1; v <= N; v++) {

printf("V%d ",adjlist[v]->vertex);

ptr = adjlist[v]->link;

while (ptr != NULL) {

printf("--> V%d ",ptr->vertex);

ptr = ptr->link;

}

printf("\n");

}

}

/\* 圖形之蹤向優先搜尋 \*/

void topological()

{

int v;

for (v = 1; v <= N; v++)

visited[v] = FALSE;

place = N;

for (v = 1; v <= N; v++)

if (!visited[v])

top\_sort(v);

}

void top\_sort(int k)

{

Node \*ptr;

int w;

visited[k] = TRUE; /\* 設定 v 頂點為已拜訪過 \*/

ptr = adjlist[k]->link; /\* 拜訪 v 相鄰頂點 \*/

while (ptr != NULL) {

w = ptr->vertex; /\* w 為 v 的立即後繼者 \*/

if (!visited[w])

top\_sort(w);

ptr = ptr->link;

}

Top\_order[--place] = k;

}

/\* 搜尋串列最後節點函數 \*/

Node \*searchlast(Node \*linklist)

{

Node \*ptr;

ptr = linklist;

while (ptr->link != NULL)

ptr = ptr->link;

return ptr;

}

輸入檔topologicalSort.dat

**===============**

第13章

P13-4 範例程式

/\* file name: bubbleSort.c \*/

/\* 氣泡排序 \*/

#include <stdio.h>

void bubbleSort(int[], int);

int main(void)

{

int data[20];

int size = 0, i;

printf("\n請輸入資料(輸入 0 表示結束): ");

/\* 要求輸入數字直到輸入數字為 0 \*/

do {

scanf("%d", &data[size]);

} while(data[size++] != 0);

printf("未排序的資料 : ");

for (i = 0; i < size-1; i++)

printf("%d ", data[i]);

printf("\n");

for (i = 0; i < 50; i++)

printf("-");

printf("\n");

bubbleSort(data, --size); /\* --size用於將資料為零者排除 \*/

for (i = 0; i < 50; i++)

printf("-");

printf("\n由小至大排序的資料 : ");

for (i = 0; i < size; i++)

printf("%d ", data[i]);

printf("\n");

return 0;

}

void bubbleSort(int data[], int size)

{

int i, j, k, temp, flag;

/\* 讓資料兩兩比較，將小的置於前 \*/

for (i=0; i<size-1; i++) {

flag=0;

printf("#%d pass: \n", i+1);

for (j=0; j<size-i-1; j++) {

if (data[j] > data[j+1]) {

flag=1;

temp = data[j];

data[j] = data[j+1];

data[j+1] = temp;

}

//印出每次的比較

printf(" #%d compare: ", j+1);

for (k = 0; k < size-i; k++)

printf(" %d ", data[k]);

printf("\n");

}

/\* 若在某一pass中沒有交換的話，則結束 \*/

if (flag != 1)

break;

printf("\n");

}

}

P13-6 範例程式

/\* file name: selectionSort.c \*/

/\* 選擇排序 \*/

#include <stdio.h>

#include <stdlib.h>

void select\_sort(int[], int);

void printDashLine(void);

int main(void)

{

int data[20];

int size = 0, i;

printf("\n請輸入資料(輸入 0 表示結束): ");

/\* 要求輸入數字直到輸入數字為 0 \*/

do {

scanf("%d", &data[size]);

} while(data[size++] != 0);

printf("未排序的資料 : ");

for (i = 0; i < size-1; i++)

printf("%d ", data[i]);

printf("\n");

printDashLine();

select\_sort(data, --size);

printDashLine();

printf("由小至大排序後的資料: ");

for (i = 0; i < size; i++)

printf("%d ", data[i]);

printf("\n");

return 0;

}

void select\_sort(int data[], int size)

{

int base, compare, min, temp, i;

for (base = 0; base < size-1; base++) {

/\* 將目前資料與後面資料中最小的對調 \*/

min = base;

for (compare = base+1; compare < size; compare++)

if(data[compare] < data[min])

min = compare;

printf("#%d PASS 選取 %d\n", base+1, data[min]);

/\* 交換的動作 \*/

temp = data[min];

data[min] = data[base];

data[base] = temp;

for (i = 0; i < size; i++)

printf(" %d ", data[i]);

printf("\n\n");

}

}

void printDashLine()

{

for (int i = 0; i < 40; i++)

printf("-");

printf("\n");

}

P13-9 範例程式

/\* file name: insertionSort.c \*/

/\* 插入排序 \*/

#include <stdio.h>

void insertion\_sort(int[], int);

void printDashLine(void);

int main()

{

int data[20];

int size = 0, i;

printf("\n請輸入資料(輸入 0 表示結束): ");

/\* 要求輸入數字直到輸入數字為 0 \*/

do {

scanf("%d", &data[size]);

} while(data[size++] != 0);

printf("未排序的資料 : ");

for (i = 0; i < size-1; i++)

printf("%d ", data[i]);

printf("\n");

printDashLine();

insertion\_sort(data, --size);

printDashLine();

printf("由小至大排序後的資料: ");

for (i = 0; i < size; i++)

printf("%d ", data[i]);

printf("\n");

return 0;

}

void insertion\_sort(int data[], int size)

{

int base, compare, temp, i;

for (base = 0; base < size; base++) {

/\* 將此資料與位於它前面的資料比對。找出插入位置 \*/

temp = data[base];

compare = base;

printf("加入 %d\n", temp);

while (compare > 0 && data[compare-1] > temp) {

data[compare] = data[compare-1];

compare--;

}

data[compare] = temp;

printf("#%d PASS : ", base+1);

for (i = 0; i <= base; i++)

printf("%5d ", data[i]);

printf("\n\n");

}

}

void printDashLine()

{

for (int i = 0; i < 40; i++)

printf("-");

printf("\n");

}

P 13-12範例程式

/\* file name: mergeSort.c\*/

/\* 合併排序 \*/

#include <stdio.h>

#include <stdlib.h>

void selectionSort(int[], int);

void mergeSort(int[], int[], int[], int, int);

void flushBuffer(void);

void printDashLine(void);

int main()

{

int data1[10], data2[10], data3[20];

int size1 = 0, size2 = 0, i;

printf("\n請輸入data1陣列的資料(輸入 0 表示結束): ");

/\* 要求輸入數字直到輸入數字為 0 \*/

do {

scanf("%d", &data1[size1]);

} while(data1[size1++] != 0);

flushBuffer();

printf("未排序的資料 : ");

for (i = 0; i < size1-1; i++)

printf("%d ", data1[i]);

printf("\n");

printf("\n請輸入data2陣列的資料(輸入 0 表示結束): ");

/\* 要求輸入數字直到輸入數字為 0 \*/

do {

scanf("%d", &data2[size2]);

} while(data2[size2++] != 0);

flushBuffer();

printf("未排序的資料 : ");

for (i = 0; i < size2-1; i++)

printf("%d ", data2[i]);

printf("\n");

/\* 先使用選擇排序將兩數列排序，再作合併 \*/

selectionSort(data1, --size1);

selectionSort(data2, --size2);

printDashLine();

/\* 印出這兩個陣列，分別排序後的資料 \*/

printf("利用選擇排序後的資料如下: \n");

printf("Data1 : ");

for(i = 0; i < size1; i++)

printf("%d ", data1[i]);

printf("\n");

printf("Data2 : ");

for(i = 0; i < size2; i++)

printf("%d ", data2[i]);

printf("\n");

printDashLine();

mergeSort(data1, data2, data3, size1, size2);

printDashLine();

printf("由小至大排序後的資料: ");

for (i = 0; i < size1+size2; i++)

printf("%d ", data3[i]);

printf("\n");

return 0;

}

void selectionSort(int data[], int size)

{

int base, compare, min, temp;

for (base = 0; base < size-1; base++) {

min = base;

for (compare = base+1; compare < size; compare++)

if (data[compare] < data[min])

min = compare;

temp = data[min];

data[min] = data[base];

data[base] = temp;

}

}

void mergeSort(int data1[], int data2[], int data3[], int size1, int size2)

{

int arg1, arg2, arg3, i;

data1[size1] = 32767;

data2[size2] = 32767;

arg1 = 0;

arg2 = 0;

for (arg3 = 0; arg3 < size1+size2; arg3++) {

/\* 比較兩數列，資料小的先存於合併後的數列 \*/

if (data1[arg1] < data2[arg2]) {

data3[arg3] = data1[arg1];

arg1++;

}

else {

data3[arg3] = data2[arg2];

arg2++;

}

printf("#%2d Step: ", arg3+1);

for (i = 0; i < arg3+1; i++)

printf("%d ", data3[i]);

printf("\n");

}

}

void flushBuffer()

{

while (getchar() != '\n')

continue;

}

void printDashLine()

{

for(int i = 0; i < 48; i++)

printf("-");

printf("\n");

}

P13-17 範例程式

/\* file name: quickSort.c \*/

/\* 快速排序 \*/

#include <stdio.h>

void quickSort(int[], int, int, int);

void printDashLine(void);

int main()

{

int data[20];

int size = 0, i;

printf("\n請輸入資料(輸入 0 表示結束): ");

/\* 要求輸入數字直到輸入數字為 0 \*/

do {

scanf("%d", &data[size]);

} while(data[size++] != 0);

printDashLine();

quickSort(data, 0, size-2, size);

printDashLine();

printf("由小至大排序後的資料: ");

for (i = 0; i < size-1; i++)

printf("%d ", data[i]);

printf("\n");

return 0;

}

void quickSort(int data[], int left, int right, int size)

{

/\* left 與 right 分別表欲排序資料兩端 \*/

int lbase, rbase, temp, i;

if (left < right) {

lbase = left+1;

while(data[lbase] < data[left])

lbase++;

rbase = right;

while(data[rbase] > data[left])

rbase--;

/\* 若lbase小於rbase，則兩資料對調 \*/

while (lbase < rbase) {

temp = data[lbase];

data[lbase] = data[rbase];

data[rbase] = temp;

lbase++;

while(data[lbase] < data[left])

lbase++;

rbase--;

while (data[rbase] > data[left])

rbase--;

}

/\* 此時lbase大於rbase，則rbase的資料與第一筆對調 \*/

temp = data[left];

data[left] = data[rbase];

data[rbase] = temp;

printf("Processing: ");

for (i = 0; i < size-1; i++)

printf("%3d ", data[i]);

printf("\n");

quickSort(data, left, rbase-1, size);

quickSort(data, rbase+1, right, size);

}

}

void printDashLine()

{

for(int i = 0; i < 62; i++)

printf("-");

printf("\n");

}

P13-23 範例程式

/\* file name: heapSort.c \*/

/\* 堆積排序 \*/

#include <stdio.h>

void adjust(int, int);

void printDashLine(void);

int data[11] = {0, 75, 23, 98, 44, 57, 12, 29, 64, 38, 82};

int main()

{

int i, k, temp;

printf("\n<< Heap sort >>\n");

printf("\n排序前的資料 : ");

for(k = 1; k <= 10; k++)

printf("%d ", data[k]);

printf("\n");

printDashLine();

for(i = 10/2; i > 0; i--)

adjust(i, 10);

printf("\nHeap : ");

for (k = 1; k <= 10; k++)

printf("%3d ", data[k]);

for (i = 9; i > 0; i--) {

temp = data[i+1];

data[i+1] = data[1];

data[1] = temp; /\* 將樹根和最後的節點交換 \*/

adjust(1, i); /\* 再重新調整為堆積樹 \*/

printf("\nProcess: ");

for (k = 1; k <= 10; k++)

printf("%3d ", data[k]);

}

printf("\n\n由小至大排序後的資料 : ");

for(k = 1; k <= 10; k++)

printf("%d ", data[k]);

printf("\n");

return 0;

}

void adjust(int i, int n) /\* 將資料調整為堆積樹 \*/

{

int j, k, done = 0;

k = data[i];

j = 2\*i;

while ((j <= n) && (done == 0)) {

if ((j < n) && (data[j] < data[j+1]))

j++;

if (k >= data[j])

done = 1;

else {

data[j/2] = data[j];

j \*= 2;

}

}

data[j/2] = k;

}

void printDashLine()

{

for (int k = 0; k < 52; k++)

printf("-");

printf("\n");

}

P13-27 範例程式

/\* file name: binaryTreeSort.c \*/

/\* 二元樹排序 \*/

#include <stdio.h>

#include <stdlib.h>

struct data {

int num;

struct data \*lbaby, \*rbaby;

} \*root, \*tree, \*leaves;

void find(int, struct data \*);

void output(struct data \*);

void printDashLine(void);

int main()

{

int data[10] = {75, 23, 98, 44, 57, 12, 29, 64, 38, 82};

int i;

printf("\n<< Binary tree sort >>\n");

printf("\n未排序的資料: ");

for (i = 0; i < 10; i++)

printf("%d ", data[i]);

printf("\n");

printDashLine();

root = (struct data \*) malloc(sizeof(struct data));

root->num = data[0]; /\* 建樹根 \*/

root->lbaby = NULL;

root->rbaby = NULL;

printf("\n#%2d Step : ", 1);

output(root);

leaves = (struct data \*) malloc(sizeof(struct data));

for (i = 1; i < 10; i++) { /\* 建樹枝 \*/

leaves->num = data[i];

leaves->lbaby = NULL;

leaves->rbaby = NULL;

find(leaves->num, root);

if (leaves->num > tree->num) /\* 若比父節點大，則放右子樹 \*/

tree->rbaby = leaves;

else /\* 否則放在左子樹 \*/

tree->lbaby = leaves;

printf("\n#%3d Step : ", i+1);

output(root);

leaves = (struct data \*) malloc(sizeof(struct data));

}

puts("");

printDashLine();

printf("\n由小至大排序後的資料 : ");

output(root);

printf("\n");

return 0;

}

/\* 尋找新節點存放的位置 \*/

void find(int input, struct data \*papa)

{

if ((input > papa->num) && (papa->rbaby != NULL))

find(input, papa->rbaby);

else if ((input < papa->num) && (papa->lbaby != NULL))

find(input, papa->lbaby);

else

tree = papa;

}

/\* 用中序追蹤將資料印出 \*/

void output(struct data \*node)

{

if (node != NULL) {

output(node->lbaby);

printf("%d ", node->num);

output(node->rbaby);

}

}

void printDashLine()

{

for(int i = 0; i < 58; i++)

printf("-");

}

P13-30 範例程式

/\* file name: shellSort.c\*/

/\* 謝耳排序 \*/

#include <stdio.h>

#include <stdlib.h>

void printDashLine(void);

int main()

{

int data[11] = {0, 75, 23, 98, 44, 57, 12, 29, 64, 38, 82};

int i, j, k, incr, temp;

printf("\n<< Shell sort >>\n");

printf("\n未排序前的資料: ");

for (i = 1; i <= 10; i++)

printf("%d ", data[i]);

printf("\n");

printDashLine();

incr = 10/2;

while (incr > 0) {

for (i = incr+1; i <= 10; i++) {

j = i - incr;

while (j > 0)

if (data[j] > data[j+incr]) {

/\* 大小順序不對則交換 \*/

temp = data[j];

data[j] = data[j+incr];

data[j+incr] = temp;

j = j - incr;

}

else

j = 0;

}

printf("\nProcessing : ");

for(k = 1; k <= 10; k++)

printf("%d ", data[k]);

incr = incr/2;

}

printf("\n");

printDashLine();

printf("由小至大排序後的資料: ");

for (i = 1; i <= 10; i++)

printf("%d ", data[i]);

printf("\n");

return 0;

}

void printDashLine()

{

for(int i = 0; i < 58; i++)

printf("-");

printf("\n");

}

**===============**

第14章

P14-2 範例程式

/\* file name: sequentialSearch.c \*/

/\* 循序搜尋 \*/

#include <stdio.h>

#include <stdlib.h>

int main()

{

int data[10] = {75, 23, 98, 44, 57, 12, 29, 64, 38, 82};

int i, input;

printf("\n<< Squential search >>\n");

printf("\n資料如下: ");

for (i = 0; i < 10; i++)

printf("%d ", data[i]);

printf("\n");

printf("\n請輸入要找尋的資料: ");

scanf("%d", &input);

printf("\nSearching.....\n");

/\* 依序搜尋資料 \*/

for (i = 0; i < 10; i++) {

printf("\n找尋第 %2d 次，資料是 %d", i+1, data[i]);

if(input == data[i])

break;

}

if (i == 10)

printf("\n\n抱歉, %d 找不到!", input);

else

printf("\n\n找到了, %d 在陣列的第 %d 個!", input, i+1);

printf("\n");

return 0;

}

P14-5 範例程式

/\* file name: binarySearch.c \*/

/\* 二元搜尋 \*/

#include <stdio.h>

#include <stdlib.h>

int main()

{

int data[10] = {12, 23, 29, 38, 44, 57, 64, 75, 82, 98};

int i, left = 0, right = 9, mid, cnt = 0, input, ok = 0;

printf("\n<< Binary search >>\n");

printf("\n已排序好的資料: ");

for(i = 0; i < 10; i++)

printf("%d ", data[i]);

printf("\n");

printf("\n請輸入欲找尋的資料: ");

scanf("%d", &input);

printf("\nSearching....\n");

mid = (left + right)/2; /\* 鍵值在第M筆 \*/

while (left <= right && ok == 0) {

printf("\n第%2d 次找的資料是: %d!", ++cnt, data[mid]);

/\* 欲搜尋的資料小於鍵值，則資料在鍵值的前面 \*/

if(data[mid] > input) {

right = mid - 1;

printf("\n ---> 要找的資料小於 %d\n", data[mid]);

}

/\* 否則資料在鍵值的後面 \*/

else {

if(data[mid] < input) {

left = mid + 1;

printf("\n ---> 要找的資料大於 %d\n", data[mid]);

}

else {

printf("\n\n找到了, %d 是在索引 %d 的位置!\n", input, mid);

ok = 1;

}

}

mid = (left + right)/2;

}

if(ok == 0)

printf("\n\n抱歉, %d 找不到!", input);

printf("\n");

return 0;

}

P14-10 範例程式

/\* file name: hashingTableUsingList.c \*/

/\* 雜湊法 : 使用鏈結串列處理碰撞 \*/

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define MAX\_NUM 100 /\* 最大資料筆數 \*/

#define PRIME 97 /\* MAX\_NUM之質數 \*/

/\* 定義資料結構 \*/

typedef struct Person {

long id;

char name[21];

struct Person \*link;

} Student;

/\* 建立雜湊表串列 \*/

Student \*Hashtab[MAX\_NUM], \*current;

/\*函數原型宣告\*/

long hashfun(long);

void insert(void);

void del(void);

Student \*search(Student \*,Student \*);

void query(void);

void show(void);

void flushBuffer(void);

int main()

{

char \*menu\_prompt =

"\n=== Hashing Table Program ==\n"

" 1. Insert\n"

" 2. Delete\n"

" 3. Show\n"

" 4. Search\n"

" 5. Quit\n"

" 請輸入選項: ";

char choice;

int i;

/\* 起始雜湊串列，將各串列指向 NULL \*/

for (i = 0; i< MAX\_NUM; i++)

Hashtab[i] = NULL;

do {

printf("%s", menu\_prompt);

choice = toupper(getchar());

flushBuffer();

printf("\n");

switch (choice) {

case '1':

insert();

break;

case '2':

del();

break;

case '3':

show();

break;

case '4':

query();

break;

case '5':

puts("Bye Bye ^\_^");

break;

default:

puts("錯誤選項!!");

}

} while (choice != '5');

return 0;

}

/\* 雜湊函數: \*/

/\* 以除法運算傳求出記錄應儲存的位址 \*/

long hashfun(long key)

{

return (key % PRIME);

}

void insert()

{

Student \*newnode;

long index;

/\*輸入記錄\*/

newnode = (Student \*)malloc(sizeof(Student));

newnode->link = NULL;

printf("Enter id: ");

scanf("%ld",&newnode->id);

printf("Enter Name: ");

scanf("%s",newnode->name);

flushBuffer();

/\* 利用雜湊函數求得記錄位址 \*/

index = hashfun(newnode->id);

printf("index 為 %ld\n", index);

/\* 判斷該串列是否為空，若為空則建立此鏈結串列 \*/

if (Hashtab[index] == NULL) {

Hashtab[index] = newnode;

printf("Node insert ok!\n");

}

else {

printf("有碰撞，加入串列中...\n");

/\* 加入於串列中 \*/

current = Hashtab[index];

while (current->link != NULL) {

current = current->link;

}

current->link = newnode;

}

}

/\* 刪除節點函數 \*/

void del()

{

Student \*node ,\*node\_parent;

long index;

node = (Student \*)malloc(sizeof(Student));

printf("Enter ID: ");

scanf("%ld",&node->id);

flushBuffer();

/\* 利用雜湊函數轉換記錄位址 \*/

index = hashfun(node->id);

/\* 搜尋節點是否存在並傳回指向該節點指標 \*/

node = search(Hashtab[index], node);

if (node == NULL)

printf("此資料找不到...\n");

else {

/\* 如節點為串列首，則將串列指向NULL

否則找到其父節點，並將父節點link向節點後端 \*/

printf("ID : %ld Name : %s\n",node->id,node->name);

if (node == Hashtab[index])

Hashtab[index] = NULL;

else {

node\_parent = Hashtab[index];

while (node\_parent->link->id != node->id)

node\_parent = node\_parent->link;

node\_parent->link = node->link;

}

free(node);

printf("此筆資料已刪除....\n");

}

}

/\* 搜尋節點函數

如找到節點則傳回指向該節點之指標

否則傳回NULL \*/

Student \*search(Student \*linklist, Student \*Node)

{

Student \*ptr = linklist;

if (ptr == NULL)

return NULL;

while (ptr->id != Node->id && ptr->link != NULL)

ptr = ptr->link;

if (ptr == NULL)

return NULL;

else

return ptr;

}

/\*查詢節點函數\*/

void query()

{

Student \*query\_node;

long index;

query\_node = (Student \*)malloc(sizeof(Student));

printf("Enter ID: ");

scanf("%ld", &query\_node->id);

flushBuffer();

index = hashfun(query\_node->id);

/\* 搜尋節點 \*/

query\_node = search(Hashtab[index], query\_node);

if (query\_node == NULL)

printf("雜湊表沒有此筆資料...\n");

else {

printf("ID: %ld Name: %s\n", query\_node->id,query\_node->name);

}

}

/\* 顯示節點函數，

從雜湊串列一一尋找是否有節點存在 \*/

void show()

{

int i, flag=0;

Student \*ptr;

puts("ID NAME");

puts("------------------------");

for ( i = 0; i < MAX\_NUM; i++ ) {

if (Hashtab[i] != NULL){

flag = 1;

ptr = Hashtab[i];

/\* 串列後面若還有資料，則將整串列顯示出 \*/

while (ptr) {

printf("%-5ld %15s\n",ptr->id,ptr->name);

ptr = ptr->link;

}

}

}

if (flag == 0)

printf("The Hashing table is empty !!!\n");

}

void flushBuffer()

{

while (getchar() != '\n')

continue;

}