/\* Adelson Velsekii and Landis Tree with Graphics Display \*/

/\*

Polytechnic University of the Philippines

Mabini Campus, Sta. Mesa, Manila

College of Computer Management and Information Technology

A Partial Fulfillment of the Subject

Design and Analysis of Algorithm

Programmed by:

Frederick A. Badion

BS Computer Science 2nd year (SY: 2K5-2K6)

Subject Professor:

Mrs. Ria Sagum

Chaiperson BSCompSci

frederick.badion@pup.edu.ph

\*/

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <dos.h>

#include <string.h>

#include <graphics.h>

#include <conio.h>

struct node

{

int element;

node \*left;

node \*right;

int height;

};

typedef struct node \*nodeptr;

void insert(int,nodeptr &);

void del(int, nodeptr &);

int deletemin(nodeptr &);

void find(int,nodeptr &);

nodeptr findmin(nodeptr);

nodeptr findmax(nodeptr);

void makeempty(nodeptr &);

nodeptr nodecopy(nodeptr &);

void preorder(nodeptr);

void inorder(nodeptr);

void postorder(nodeptr);

int bsheight(nodeptr);

nodeptr singlerotateleft(nodeptr &);

nodeptr doublerotateleft(nodeptr &);

nodeptr singlerotateright(nodeptr &);

nodeptr doublerotateright(nodeptr &);

int max(int,int);

int nonodes(nodeptr);

int gdriver=DETECT,gmode=0,errorcode;

char element[3];

int x=1,maxx,midx,xcoord,ycoord,level=320,prevlevel;

void GDisplay(nodeptr p,int xcoord,int ycoord)

{

if (p!=NULL)

{

level=level/2;

setfillstyle(1,BROWN);

setcolor(LIGHTGREEN);

if(p->left->element!=NULL)

line(xcoord,ycoord,xcoord-level,ycoord+50);

if(p->right->element!=NULL)

line(xcoord,ycoord,xcoord+level,ycoord+50);

fillellipse(xcoord,ycoord,10,10);

sprintf(element,"%d",p->element,xcoord,ycoord);

settextstyle(2,0,4);

setcolor(YELLOW);

outtextxy(xcoord-7,ycoord-7,element);

GDisplay(p->left,xcoord-(level),ycoord+50);

GDisplay(p->right,xcoord+(level),ycoord+50);

level=level\*2;

}

}//end of GDisplay

void insert(int x,nodeptr &p)

{

if (p == NULL)

{

p = new node;

p->element = x;

p->left=NULL;

p->right = NULL;

p->height=0;

if (p==NULL)

{

gotoxy(4,21);

printf("Out of Space");

}

}

else

{

if (x<p->element)

{

insert(x,p->left);

//GDisplay(root,midx,50);

if ((bsheight(p->left) - bsheight(p->right))==2)

{

if (x < p->left->element)

p = singlerotateleft(p); //single rotation left

else

p = doublerotateleft(p); //double rotation left

}

}

else if (x>p->element)

{

insert(x,p->right);

//GDisplay(root,midx,50);

if ((bsheight(p->right) - bsheight(p->left))==2)

{

if (x > p->right->element)

p = singlerotateright(p); //single rotation right

else

p = doublerotateright(p); //double rotation right

}

}

else

{

gotoxy(4,21);

printf("Element Exists");

}

}

int m,n,d;

m=bsheight(p->left);

n=bsheight(p->right);

d=max(m,n);

p->height = d + 1;

}

nodeptr findmin(nodeptr p)

{

if (p==NULL)

{

gotoxy(4,21); printf("Empty Tree");

return p;

}

else

{

while(p->left !=NULL)

p=p->left;

return p;

}

}

nodeptr findmax(nodeptr p)

{

if (p==NULL)

{

gotoxy(4,21); printf("Empty Tree");

return p;

}

else

{

while(p->right !=NULL)

p=p->right;

return p;

}

}

void find(int x,nodeptr &p)

{

if (p==NULL)

{

gotoxy(4,21);

printf("Element not found");

}

else if (x < p->element)

find(x,p->left);

else if (x>p->element)

find(x,p->right);

else

{

gotoxy(4,21);

printf("Element found !");

}

}

void del(int x,nodeptr &p)

{

nodeptr d;

if (p==NULL)

{

gotoxy(4,21);

printf("Element not found");

}

else if ( x < p->element)

{

del(x,p->left);

if ((bsheight(p->left) - bsheight(p->right))==2)

{

if (x < p->left->element)

p = singlerotateleft(p); //single rotation left

else

p = doublerotateleft(p); //double rotation left

}

}

else if (x > p->element)

{

del(x,p->right);

if ((bsheight(p->right) - bsheight(p->left))==2)

{

if (x > p->right->element)

p = singlerotateright(p); //single rotation right

else

p = doublerotateright(p); //double rotation right

}

}

else if ((p->left == NULL) && (p->right == NULL))

{

d=p;

free(d);

p=NULL;

gotoxy(4,21); printf("Element deleted !");

}

else if (p->left == NULL)

{

d=p;

free(d);

p=p->right;

gotoxy(4,21); printf("Element deleted !");

}

else if (p->right == NULL)

{

d=p;

p=p->left;

free(d);

gotoxy(4,21); printf("Element deleted !");

}

else

p->element = deletemin(p->right);

}

int deletemin(nodeptr &p)

{

int c;

gotoxy(4,21); printf("deltemin");

if (p->left == NULL)

{

c=p->element;

p=p->right;

return c;

}

else

{

c=deletemin(p->left);

return c;

}

}

void preorder(nodeptr p)

{

if (p!=NULL)

{

printf("%d-->",p->element);

preorder(p->left);

preorder(p->right);

}

}

void inorder(nodeptr p)

{

if (p!=NULL)

{

inorder(p->left);

printf("%d-->",p->element);

inorder(p->right);

}

}

void postorder(nodeptr p)

{

if (p!=NULL)

{

postorder(p->left);

postorder(p->right);

printf("%d-->",p->element);

}

}

int max(int value1, int value2)

{

if(value1 > value2)

return value1;

else

return value2;

}

int bsheight(nodeptr p)

{

int t;

if (p == NULL)

return -1;

else

{

t = p->height;

return t;

}

}

nodeptr singlerotateleft(nodeptr &p1)

{

nodeptr p2;

p2 = p1->left;

p1->left = p2->right;

p2->right = p1;

p1->height = max(bsheight(p1->left),bsheight(p1->right)) + 1;

p2->height = max(bsheight(p2->left),p1->height) + 1;

return p2;

}

nodeptr singlerotateright(nodeptr &p1)

{

nodeptr p2;

p2 = p1->right;

p1->right = p2->left;

p2->left = p1;

p1->height = max(bsheight(p1->left),bsheight(p1->right)) + 1;

p2->height = max(p1->height,bsheight(p2->right)) + 1;

return p2;

}

nodeptr doublerotateleft(nodeptr &p1)

{

p1->left = singlerotateright(p1->left);

return singlerotateleft(p1);

}

nodeptr doublerotateright(nodeptr &p1)

{

p1->right = singlerotateleft(p1->right);

return singlerotateright(p1);

}

int count=0;

int nonodes(nodeptr p)

{

if (p!=NULL)

{

nonodes(p->left);

nonodes(p->right);

count++;

}

return count;

}

void twolinebox(int x1,int y1,int x2,int y2){

int x,y;

textcolor(WHITE);

gotoxy(x1,y1); cprintf("É"); //201

gotoxy(x2,y1); cprintf("»"); //187

for(y=y1+1;y<y2;y++){

gotoxy(x1,y); cprintf("º"); //186

gotoxy(x2,y); cprintf("º"); //186

}

gotoxy(x1,y2); cprintf("È"); //200

gotoxy(x2,y2); cprintf("¼"); //188

for(x=x1+1;x<x2;x++){

gotoxy(x,y1); cprintf("Í"); //205

gotoxy(x,y2); cprintf("Í"); //205

}

gotoxy(x1+1,y1+1);

}

void cprintxy(int x,int y,int color,char string[]){

textcolor(color);

gotoxy(x,y); cprintf("%s",string);

}

void center(int y,int color,char string[]){

int x=(80-strlen(string)+1)/2;

textcolor(color);

gotoxy(x,y);cprintf("%s",string);

}

void Temp(void){

int x,y;

clrscr();

twolinebox(1,1,80,24);

for(y=23;y>=1;y--){

sound(60\*y);

center(y,YELLOW,"[Adelson-Velskii and Landis Tree]");

gotoxy(2,y+1); clreol();

twolinebox(1,1,80,24);

delay(40);

nosound();

}

center(1,YELLOW,"[Adelson-Velskii and Landis Tree]");

for(x=74;x>=3;x--){

sound(50\*x);

cprintxy(x,5,RED,"Press:"); clreol();

twolinebox(1,1,80,24);

center(1,YELLOW,"[Adelson-Velskii and Landis Tree]");

delay(20);

nosound();

}

twolinebox(1,1,80,12);

twolinebox(1,1,80,24);

center(1,YELLOW,"[Adelson-Velskii and Landis Tree]");

cprintxy(20,3,GREEN,"[A]- Insertion");

cprintxy(20,4,GREEN,"[B]- Find Minimum");

cprintxy(20,5,GREEN,"[C]- Find Maximum");

cprintxy(20,6,GREEN,"[D]- Search Node");

cprintxy(20,7,GREEN,"[E]- Display Tree");

cprintxy(43,3,GREEN,"[F]- Delete Node");

cprintxy(43,4,GREEN,"[G]- Preorder");

cprintxy(43,5,GREEN,"[H]- Inorder");

cprintxy(43,6,GREEN,"[I]- Postorder");

cprintxy(43,7,GREEN,"[J]- Height");

cprintxy(20,9,GREEN,"[any key]- Quit...");

cprintxy(20,11,LIGHTRED,"Enter your choice: ");

}

void main()

{

nodeptr root,min,max;

int a,findele,delele,leftele,rightele,flag;

char choice,value[2];

char ch='Y';

root = NULL;

textmode(C80);

Temp();

do

{

clrscr();

twolinebox(1,1,80,12);

twolinebox(1,1,80,24);

center(1,YELLOW,"[Adelson-Velskii and Landis Tree]");

cprintxy(5,3,LIGHTRED,"Press: ");

cprintxy(20,3,GREEN,"[A]- Insertion");

cprintxy(20,4,GREEN,"[B]- Find Minimum");

cprintxy(20,5,GREEN,"[C]- Find Maximum");

cprintxy(20,6,GREEN,"[D]- Search Node");

cprintxy(20,7,GREEN,"[E]- Display Tree");

cprintxy(43,3,GREEN,"[F]- Delete Node");

cprintxy(43,4,GREEN,"[G]- Preorder");

cprintxy(43,5,GREEN,"[H]- Inorder");

cprintxy(43,6,GREEN,"[I]- Postorder");

cprintxy(43,7,GREEN,"[J]- Height");

cprintxy(20,9,GREEN,"[any key]- Quit...");

cprintxy(20,11,LIGHTRED,"Enter your choice:\>");

choice=getch();

switch(toupper(choice))

{

case 'A':

do{

gotoxy(4,14); printf("Enter node value: ");

a=atoi(gets(value));

if(atoi(value)==0)

{

gotoxy(4,21); printf("Error!!! Enter a valid integer value only.");

gotoxy(4,22); printf("Press any key to continue...");

getch();

}

}while(atoi(value)==0);

insert(a,root);

gotoxy(4,15);

inorder(root);

/\*

initgraph(&gdriver,&gmode,"c:\tc\bgi");

errorcode = graphresult();

maxx=getmaxx();

midx=maxx/2,xcoord=midx/2,ycoord=40;

if (errorcode != grOk)

{

printf("Graphics error: %s

", grapherrormsg(errorcode));

printf("Press any key to stop");

getch();

exit(1);

}

cleardevice();

GDisplay(root,midx,50);

getch();

restorecrtmode();

\*/

break;

case 'B':

if (root !=NULL)

{

min=findmin(root);

gotoxy(4,14); printf("Min element : %d",min->element);

}

break;

case 'C':

if (root !=NULL)

{

max=findmax(root);

gotoxy(4,14); printf("Max element : %d",max->element);

}

break;

case 'D':

gotoxy(4,14); printf("Search node :");

scanf("%d",&findele);

if (root != NULL)

find(findele,root);

break;

case 'E':

initgraph(&gdriver,&gmode,"c:\tc\bgi");

errorcode = graphresult();

maxx=getmaxx();

midx=maxx/2;

xcoord=midx/2;

ycoord=40;

if (errorcode != grOk)

{

printf("Graphics error: %s

", grapherrormsg(errorcode));

printf("Press any key to stop");

getch();

exit(1);

}

cleardevice();

setbkcolor(LIGHTBLUE);

settextstyle(2,0,5);

outtextxy(20,10,"Adelson-Velskii and Landis Tree");

GDisplay(root,midx,50);

outtextxy(20,440,"Programmed by: Frederick Badion");

outtextxy(20,450,"Polytechnic University of the Philippines");

outtextxy(20,460,"2nd year Bachelor of Science in Computer

Science");

outtextxy(320,440,"A partial fulfilment for the subject: ");

outtextxy(320,450,"Design and Analysis of Algorithm");

outtextxy(320,460,"Prof. Ria Sagum -Chairperson BSCS-CCMIT

PUP-Sta.Mesa");

getch();

restorecrtmode();

break;

case 'F':

gotoxy(4,14); printf("Delete Node: ");

scanf("%d",&delele);

del(delele,root);

getch();

initgraph(&gdriver,&gmode,"c:\tc\bgi");

errorcode = graphresult();

maxx=getmaxx();

midx=maxx/2,xcoord=midx/2,ycoord=40;

if (errorcode != grOk)

{

printf("Graphics error: %s

", grapherrormsg(errorcode));

printf("Press any key to stop");

getch();

exit(1);

}

cleardevice();

setbkcolor(LIGHTBLUE);

settextstyle(2,0,5);

outtextxy(20,10,"Adelson-Velskii and Landis Tree");

GDisplay(root,midx,50);

getch();

restorecrtmode();

break;

case 'G':

gotoxy(4,14); printf(" Preorder Printing...");

gotoxy(4,15);

preorder(root);

break;

case 'H':

gotoxy(4,14); printf(" Inorder Printing...");

gotoxy(4,15);

inorder(root);

break;

case 'I':

gotoxy(4,14); printf(" Postorder Printing...");

gotoxy(4,15);

postorder(root);

break;

case 'J':

gotoxy(4,14); printf(" Height and Depth: %d",bsheight(root));

gotoxy(4,15); printf("No. of nodes: %d",nonodes(root));

break;

}

gotoxy(4,22); printf(" Do you want to continue (y/n)?");

ch=toupper(getch());

}while(ch!='N');

}

/\*OmDev's calendar library\*/

#include<stdio.h>

#include<string.h>

#include<conio.h>

int getNumberOfDays(int month,int year)

{

switch(month)

{

case 1 : return(31);

case 2 : if(year%4==0)

return(29);

else

return(28);

case 3 : return(31);

case 4 : return(30);

case 5 : return(31);

case 6 : return(30);

case 7 : return(31);

case 8 : return(31);

case 9 : return(30);

case 10: return(31);

case 11: return(30);

case 12: return(31);

default: return(-1);

}

}

char \*getName(int odd)

{

switch(odd)

{

case 0 :return("Sunday");

case 1 :return("Monday");

case 2 :return("Tuesday");

case 3 :return("Wednesday");

case 4 :return("Thursday");

case 5 :return("Friday");

case 6 :return("Saturday");

default:return("Error in getName() module.Invalid argument

passed");

}

}

int getOddNumber(int day,int mon,int year)

{

int res=0,t1,t2,y=year;

year = year-1600;

while(year>=100)

{

res=res+5;

year=year-100;

}

res=(res%7);

t1=((year-1)/4);

t2=(year-1)-t1;

t1=(t1\*2)+t2;

t1=(t1%7);

res = res+t1;

res=res%7;

t2=0;

for(t1=1;t1<mon;t1++)

{

t2+=getNumberOfDays(t1,y);

}

t2 = t2+day;

t2 = t2%7;

res = res+t2;

res = res%7;

if(y>2000)

res=res+1;

res = res%7;

return res;

}

char \*getWeek(int dd,int mm,int yy)

{

int odd;

if(!(mm>=1 && mm<=12))

{

return("Invalid month value");

}

if(!(dd>=1 && dd<=getNumberOfDays(mm,yy)))

{

return("Invalid date");

}

if(yy>=1600)

{

odd = getOddNumber(dd,mm,yy);

odd=odd%7;

return(getName(odd));

}

else

{

return("

Please give year more than 1600");

}

}

void printMonth(int mon,int year,int x,int y)

{

int nod,odd,cnt,d=1,x1=x,y1=y;

if(!(mon>=1 && mon<=12))

{

printf("

INVALID MONTH");

getch();

return;

}

if(!(year>=1600))

{

printf("

INVALID YEAR");

getch();

return;

}

if(x<=0)

x=wherex();

if(y<=0)

y=wherey();

gotoxy(x,y);

textcolor(RED);

cprintf("S");

textcolor(YELLOW);

cprintf(" M T W T F S");

/\* 1234567891234567891234567 \*/

textcolor(7);

cprintf("");

y++;

nod=getNumberOfDays(mon,year);

odd=getOddNumber(d,mon,year);

switch(odd)

{

case 0 : x=x;

cnt=1;

break;

case 1 : x=x+4;

cnt=2;

break;

case 2 : x=x+8;

cnt=3;

break;

case 3 : x=x+12;

cnt=4;

break;

case 4 : x=x+16;

cnt=5;

break;

case 5 : x=x+20;

cnt=6;

break;

case 6 : x=x+24;

cnt=7;

break;

default : printf("

INVALID DATA FROM THE getOddNumber()

MODULE");

return;

}

gotoxy(25,25);

gotoxy(x,y);

printf("%02d",d);

for(d=2;d<=nod;d++)

{

if(cnt%7==0)

{

y++;

cnt=0;

x=x1-4;

}

x = x+4;

cnt++;

gotoxy(x,y);

printf("%02d",d);

}

}

main()

{

char ch='k';

int dd,mm,yy;

while(ch!='0')

{

clrscr();

printf("

1.Know the day");

printf("

2.Print the month");

printf("

0.EXIT");

printf("

ENTER YOUR CHOICE : ");

flushall();

fflush(stdin);

ch=getche();

clrscr();

switch(ch)

{

case '1': printf("Enter date (DD MM YYYY) : ");

scanf("%d %d %d",&dd,&mm,&yy);

printf("

Day is : %s",getWeek(dd,mm,yy));

flushall();

getch();

break;

case '2' : printf("Enter month and year (MM YYYY) : ");

scanf("%d %d",&mm,&yy);

printf("

");

printMonth(mm,yy,-1,-1);

flushall();

getch();

break;

case '0' : exit(0);

}

}

Code :

# include <stdio.h>

# include <conio.h>

# include <graphics.h>

# include <alloc.h>

# include <dos.h>

# include <stdlib.h>

# define up 72

# define down 80

# define left 75

# define right 77

# define pause 25

# define space 57

# define settings 31

struct body

{

int x,y;

struct body \*next;

};

typedef struct body body;

void create\_snake(body\*);

void show\_settings();

void draw\_borders();

void draw\_egg();

void draw\_snake();

void status\_bar();

void draw\_head();

void erase\_tail();

void disp\_gameover();

int getkey();

int check\_snake(int,int);

body \*head,\*tail;

int dir=1;

int speed=100;

int leftx,rightx,topx,bottomx;

int color = RED,fil\_style=1,bor\_color=BLACK,size = 10;

int score=0,egg\_drawn=0,jump = 9;

int preveggx,preveggy;

void far \*egg;

char scr[15];

int eggx,eggy;

void main()

{

int gm=DETECT,gd;

int key=0,num;

struct time tim;

void move\_snake\_right();

void move\_snake\_down();

void move\_snake\_left();

void move\_snake\_up();

initgraph(&gm,&gm,"g:\tc\bgi");

leftx=(size\*2-1);

rightx=getmaxx()-(size\*2);

topx=(size\*2-1);

bottomx = getmaxy()-40;

setcolor(YELLOW);

setfillstyle(SOLID\_FILL,YELLOW);

fillellipse(10,10,size/3,size/2);

egg = farmalloc(imagesize(10-size/2,10-size/2,10+size/2,10+size/2));

if(egg ==NULL)

{

printf("Insufficient Memory!");

exit(0);

}

getimage(10-size/2,10-size/2,10+size/2,10+size/2,egg);

draw\_borders();

create\_snake(head);

start:

fflush(stdin);

flushall();

while(!kbhit())

{

delay(speed);

if(egg\_drawn==0)

{

status\_bar();

eggx=eggy=0;

repeat:

gettime(&tim);

num = (int)(tim.ti\_hund)\*(int)(tim.ti\_sec);

num = num%(rightx-leftx)/size;

eggx=num;

num = (int)(tim.ti\_hund)\*(int)tim.ti\_sec;

num = num%(bottomx-topx)/size;

eggy=num;

if(check\_snake(eggx,eggy))

goto repeat;

eggx =leftx+(eggx\*size);

eggy =topx+(eggy\*size);

preveggx =leftx+(eggx\*size);

preveggy =topx+(eggy\*size);

}

draw\_egg(eggx,eggy);

switch(dir)

{

case 1:move\_snake\_right();

break;

case 2:move\_snake\_down();

break;

case 3:move\_snake\_left();

break;

case 4:move\_snake\_up();

}

}

key = getkey();

switch(key)

{

case up:if(dir!=2)

dir = 4;

break;

case down:if(dir!=4)

dir = 2;

break;

case right:if(dir!=3)

dir = 1;

break;

case left:if(dir!=1)

dir = 3;

break;

case pause:getch();

break;

case settings:show\_settings();

setfillstyle(EMPTY\_FILL,color);

bar(leftx,topx,rightx,bottomx);

draw\_snake();

break;

case 1:closegraph();

exit(0);

}

goto start;

}

int getkey()

{

union REGS i,o;

i.h.ah = 0;

int86(0x16,&i,&o);

return o.h.ah;

}

void draw\_borders()

{

setfillstyle(SOLID\_FILL,BLUE);

floodfill(300,300,1);

setfillstyle(SOLID\_FILL,BLACK);

setcolor(BLACK);

rectangle(leftx,topx,rightx,bottomx);

setfillstyle(SOLID\_FILL,BLACK);

floodfill(300,300,BLACK);

rectangle(leftx,bottomx+5,rightx,getmaxy()-5);

}

void create\_snake()

{

body \*x;

setfillstyle(fil\_style,color);

setcolor(bor\_color);

if((x = malloc(sizeof(body)))==NULL)

{

printf("Short of memory!");

return;

}

x->x = size\*3+leftx;

x->y = topx;

head = x;

bar(x->x,x->y,x->x+size,x->y+size);

rectangle(x->x,x->y,x->x+size,x->y+size);

if((x->next = malloc(sizeof(body)))==NULL)

{

printf("Short of memory!");

return;

}

x = x->next;

x->x = size\*2+leftx;

x->y = topx;

bar(x->x,x->y,x->x+size,x->y+size);

rectangle(x->x,x->y,x->x+size,x->y+size);

if((x->next = malloc(sizeof(body)))==NULL)

{

printf("Short of memory!");

return;

}

x = x->next;

x->x = size+leftx;

x->y = topx;

bar(x->x,x->y,x->x+size,x->y+size);

rectangle(x->x,x->y,x->x+size,x->y+size);

if((x->next = malloc(sizeof(body)))==NULL)

{

printf("Short of memory!");

return;

}

x = x->next;

x->x = leftx;

x->y = topx;

bar(x->x,x->y,x->x+size,x->y+size);

rectangle(x->x,x->y,x->x+size,x->y+size);

x->next = NULL;

tail = x;

}

void move\_snake\_right()

{

body \*x;

int nextx,nexty;

nexty = head->y;

if(head->x+size>=rightx)

nextx = leftx;

else

nextx = head->x+size;

if(check\_snake(nextx,nexty)==1)

{

disp\_gameover();

}

if(nextx == eggx&&nexty == eggy)

{

egg\_drawn=0;

score+=jump;

x = malloc(sizeof(body));

if(x==NULL)

{

printf("Insufficient Memory!");

exit(0);

}

x->x = nextx;

x->y = nexty;

x->next = head;

head = x;

draw\_head();

}

else

{

erase\_tail();

x = head;

while((x->next)->next !=NULL)

x = x->next;

x->next = NULL;

tail->x = nextx;

tail->y = nexty;

tail->next = head;

head = tail;

tail = x;

draw\_head();

}

}

void move\_snake\_left()

{

body \*x;

int nextx,nexty;

if(head->x-size<leftx)

nextx = rightx-size;

else

nextx = head->x-size;

nexty = head->y;

if(check\_snake(nextx,nexty)==1)

{

disp\_gameover();

}

if(nextx==eggx&&nexty==eggy)

{

egg\_drawn=0;

score+=jump;

x = malloc(sizeof(body));

if(x==NULL)

{

printf("Insufficient Memory!");

exit(0);

}

x->x = nextx;

x->y = nexty;

x->next = head;

head = x;

draw\_head();

}

else

{

erase\_tail();

x = head;

while((x->next)->next !=NULL)

x = x->next;

x->next = NULL;

tail->x = nextx;

tail->y = nexty;

tail->next = head;

head = tail;

tail = x;

draw\_head();

}

}

void move\_snake\_up()

{

body \*x;

int nextx,nexty;

if(head->y-size<topx)

nexty = bottomx-size;

else

nexty = head->y-size;

nextx = head->x;

if(check\_snake(nextx,nexty)==1)

{

disp\_gameover();

}

if(nextx==eggx&&nexty==eggy)

{

egg\_drawn=0;

score+=jump;

x = malloc(sizeof(body));

if(x==NULL)

{

printf("Insufficient Memory!");

exit(0);

}

x->x = nextx;

x->y = nexty;

x->next = head;

head = x;

draw\_head();

}

else

{

erase\_tail();

x = head;

while((x->next)->next !=NULL)

x = x->next;

x->next = NULL;

tail->x = nextx;

tail->y = nexty;

tail->next = head;

head = tail;

tail = x;

draw\_head();

}

}

void move\_snake\_down()

{

body \*x;

int nextx,nexty;

if(head->y+size>=bottomx)

nexty = topx;

else

nexty = head->y+size;

nextx = head->x;

if(check\_snake(nextx,nexty)==1)

{

disp\_gameover();

}

if(nextx==eggx&&nexty==eggy)

{

egg\_drawn=0;

score+=jump;

x = malloc(sizeof(body));

if(x==NULL)

{

printf("Insufficient Memory!");

exit(0);

}

x->x = nextx;

x->y = nexty;

x->next = head;

head = x;

draw\_head();

}

else

{

erase\_tail();

x = head;

while((x->next)->next !=NULL)

x = x->next;

x->next = NULL;

tail->x = nextx;

tail->y = nexty;

tail->next = head;

head = tail;

tail = x;

draw\_head();

}

}

void draw\_egg(int x,int y)

{

putimage(x,y,egg,COPY\_PUT);

egg\_drawn = 1;

}

void status\_bar()

{

setfillstyle(SOLID\_FILL,LIGHTGRAY);

bar(leftx+1,bottomx+4,rightx-1,getmaxy()-6);

sprintf(scr,"Score:%d",score);

setcolor(RED);

settextstyle(2,0,6);

outtextxy(leftx+5,bottomx+10,scr);

outtextxy(rightx-75,bottomx+10,"S");

setcolor(DARKGRAY);

outtextxy(rightx-66,bottomx+10,"ettings");

setcolor(RED);

outtextxy(rightx-150,bottomx+10,"P");

setcolor(DARKGRAY);

outtextxy(rightx-142,bottomx+10,"ause");

}

void draw\_head()

{

setfillstyle(fil\_style,color);

bar(head->x,head->y,head->x+size,head->y+size);

setcolor(bor\_color);

rectangle(head->x,head->y,head->x+size,head->y+size);

}

void erase\_tail()

{

setfillstyle(EMPTY\_FILL,color);

bar(tail->x,tail->y,tail->x+size,tail->y+size);

}

int check\_snake(int x,int y)

{

body \*z;

z = head;

while(z->next!=NULL)

{

if(z->x==x&&z->y==y)

return 1;

z=z->next;

}

if(z->x==x&&z->y==y)

return 1;

return 0;

}

void disp\_gameover()

{

int x1,x2,y1,y2;

x1 = getmaxx()/2-100;

y1 = getmaxy()/2-65;

x2 = getmaxx()/2+100;

y2 = getmaxy()/2+65;

setfillstyle(SOLID\_FILL,9);

bar(x1,y1,x2-30,y2-75);

setcolor(YELLOW);

settextstyle(3,0,3);

outtextxy(x1+10,y1+10,"GAME OVER!!");

getch();

closegraph();

exit(0);

}

void show\_settings()

{

int key=0,maxspeed=25,least = 145;

int x1,y1,x2,y2,width;

int no,i;

setfillstyle(SOLID\_FILL,LIGHTGRAY);

x1 = (getmaxx()+1)/2-150;

x2 = (getmaxx()+1)/2+150;

y1 = (getmaxy()+1)/2-150;

y2 = (getmaxy()+1)/2+150;

setcolor(BLUE);

bar3d(x1,y1,x2,y2,20,1);

settextstyle(2,0,6);

setcolor(RED);

outtextxy(x1+20,y1+100,"Speed:");

outtextxy(x1+85,y1+95,"-");

outtextxy(x2-10,y1+95,"+");

outtextxy(x1+20,y1+150," Use the left and right");

outtextxy(x1+20,y1+180,"arrow keys to increase and ");

outtextxy(x1+20,y1+210," decrease the speed ");

outtextxy(x1+20,y1+240,"Press escape to return");

outtextxy(x1+20,y1+270," to the game");

setcolor(BLUE);

line(x1+100,y1+120,x2-20,y1+120);

line(x1+100,y1+120,x1+100,y1+90);

line(x2-20,y1+120,x2-20,y1+90);

width = ((x2-20)-(x1+100))/8;

start:

setfillstyle(SOLID\_FILL,LIGHTGRAY);

for(i=1;i<=no;i++)

{

bar(x1+100+1,y1+90+1,x1+100+(i\*width)-1,y1+120-1);

setcolor(LIGHTGRAY);

rectangle(x1+100+1,y1+90+1,x1+100+(i\*width)-1,y1+120-1);

}

no = (speed-25)/15;

setfillstyle(SOLID\_FILL,BLUE);

for(i=1;i<=no;i++)

{

bar(x1+100+1,y1+90+1,x1+100+(i\*width)-1,y1+120-1);

setcolor(LIGHTGRAY);

rectangle(x1+100+1,y1+90+1,x1+100+(i\*width)-1,y1+120-1);

}

def:

key = getkey();

switch(key)

{

case right:if(speed<least)

speed+=15;

goto start;

case left:if(speed>maxspeed)

speed-=25;

goto start;

case 1:return;

default:goto def;

}

}

void draw\_snake()

{

body \*x;

x = head;

setfillstyle(SOLID\_FILL,color);

setcolor(bor\_color);

while(x->next!=NULL)

{

bar(x->x,x->y,x->x+size,x->y+size);

rectangle(x->x,x->y,x->x+size,x->y+size);

x = x->next;

}

}

Code :

#include<stdio.h>

#include<dos.h>

#include<conio.h>

#include<graphics.h>

void object(int,int,int);

void bubble(int);

void flow(int,int);

void mixing(int,int);

int el[10];

void main()

{

int n,i,d=0,m=0;

clrscr();

printf("Enter the No of Elements : " );

scanf("%d",&n);

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

{

printf("Enter the %d Element : ",i+1);

scanf("%d",&el[i]);

}

initgraph(&d,&m,"");

settextstyle(3,0,2);

outtextxy(250,50,"GIVEN NUMBERS ");

outtextxy(250,260," AFTER SORTING "); //

line(0,getmaxy()/2,getmaxx(),getmaxy()/2);

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

{

object(100+i\*50,150,el[i]);

getch();

}

bubble(n);

/\*

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

{

object(100+i\*50,350,el[i]);

getch();

}

\*/

getch();

}

void bubble(int n)

{

int i,j,temp;

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

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

if(el[i]>el[j+1])

{

flow(i,j+1);

temp=el[i]; el[i]=el[j+1]; el[j+1]=temp;

}

}

void object(int x,int y,int no)

{

char s[8];

sprintf(s,"%d",no);

circle(x,y,15);

settextstyle(2,0,6);

outtextxy(x-3,y-10,s);

}

void flow(int f,int s)

{

int i;

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

{

setcolor(WHITE);

object(100+f\*50,150+i\*4,el[f]);

delay(15);

setcolor(0);

object(100+f\*50,150+i\*4,el[f]);

}

setcolor(WHITE);

object(100+f\*50,150+i\*4,el[f]);

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

{

setcolor(WHITE);

object(100+s\*50,150+i\*4,el[s]);

delay(10);

setcolor(0);

object(100+s\*50,150+i\*4,el[s]);

}

object(100+s\*50,150+i\*4,el[s]);

mixing(f,s);

for(i=50;i>0;i--)

{

setcolor(WHITE);

object(100+f\*50,150+i\*4,el[s]);

delay(10);

setcolor(0);

object(100+f\*50,150+i\*4,el[s]);

}

setcolor(WHITE);

object(100+f\*50,150+i\*4,el[s]);

for(i=50;i>0;i--)

{

setcolor(WHITE);

object(100+s\*50,150+i\*4,el[f]);

delay(15);

setcolor(0);

object(100+s\*50,150+i\*4,el[f]);

}

setcolor(WHITE);

object(100+s\*50,150+i\*4,el[f]);

}

void mixing(int f,int s)

{

int i;

for(i=0;i<(s-f)\*50;i++)

{

setcolor(WHITE);

object(100+f\*50+i,350,el[f]);

object(100+s\*50-i,350,el[s]);

delay(20);

setcolor(0);

object(100+f\*50+i,350,el[f]);

object(100+s\*50-i,350,el[s]);

}

setcolor(WHITE);

object(100+f\*50+i,350,el[f]);

object(100+s\*50-i,350,el[s]);

}

#include <conio.h>

#include <ctype.h>

#include <dos.h>

#include <graphics.h>

#include <stdlib.h>

#include <stdarg.h>

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <time.h>

#define MAXNUM 200 /\* maximum number of array elements to sort \*/

#define XAXIS 260 /\* x coordinate on graphics screen \*/

#define YAXIS 15 /\* y coordinate on graphics screen \*/

#define MAXPICKS 8 /\* maximum menu picks given to user \*/

#define TIMES 3 /\* how many times to perform... \*/

int xaxis = XAXIS;

int yaxis = YAXIS;

enum sort {bubble, delayed, shell, shell\_metzner,

quick, insertion, all, stop};

char \*sorts[MAXPICKS] =

{"Bubble Sort", "Delayed Exchange Sort", "Shell Sort",

"Shell-Metzner Sort", "QuickSort", "Insertion Sort",

"All", "Exit to Dos"};

/\*\*\*\*\* function prototypes \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main( void );

void driver( enum sort atype, int \*m, int elements,

int random, int delay\_factor );

enum sort pick\_sort( int \*elements, int \*random, int \*delay\_factor );

void Initialize( void );

void Setscreen( int \*m, int elements, int random );

int Swap\_Pixels( int \*m, int i, int j, int delay\_factor );

int gprintf( int \*xloc, int \*yloc, char \*fmt, ... );

void print\_menu( char \*mysorts[] );

void get\_number( int \*elements, int \*times, char \*tstring, int \*x, int

\*y );

void Showdata ( int \*m );

void Bubble( int \*m, int elements, int delay\_factor );

void Delayed( int \*m, int elements, int delay\_factor );

void Shell\_Metzner( int \*m, int elements, int delay\_factor );

void Quicksort( int \*m, int left, int right, int delay\_factor );

void Insertion( int \*m, int elements, int delay\_factor );

void Shell( int \*m, int elements, int delay\_factor );

/\*\*\*\*\* main \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main( void )

{

int array[MAXNUM]; /\* the array to be sorted \*/

int elements; /\* how many elements \*/

int random; /\* random or worst case \*/

int delay\_factor; /\* delay factor 0-1000 \*/

enum sort stype = all;

Initialize();

while( stype != stop )

{

random = 0;

elements = 0;

delay\_factor = 0;

stype = pick\_sort( &elements, &random, &delay\_factor );

if ( stype != stop )

{

driver( stype, array, elements, random, delay\_factor );

/\* Showdata( array ); \*/

delay( 1350 );

}

}

closegraph();

}

/\*\*\*\*\* pick\_sort

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Displays a simple menu and prompts the user for four choices.

called by: main

calls: print\_menu

gprintf

get\_number

returns : the sort desired (could be all sorts or quit)

parameters:

\*elements

\*random

\*delay\_factor

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

enum sort pick\_sort( int \*elements, int \*random, int \*delay\_factor )

{

static char query1[] = "Which Sort (1-8)?";

static char query2[] = "How Many Elements < 200?";

static char query3[] = "(R)andom or (W)orst Case?";

static char query4[] = "Delay Factor (0-1000)?";

static char achar[2] = "x";

char bchar = 0;

char nstring[TIMES + 1]; /\* string equivalent of elements \*/

int tens = TIMES; /\* power of ten we're using \*/

int \*tpower;

int x = 50;

int y = 30;

char pick = 0;

int x2;

int i; /\* scratch variable \*/

tpower = &tens;

cleardevice();

print\_menu( sorts );

/\*\*\*\*\*\*\*\*\*\*\*\*\*\* pick the sort \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

gprintf( &x, &y, query1 );

while ( pick <= 48 || pick >= 57 ) /\* allow digits 1-8 \*/

{

pick = getch();

}

achar[0] = pick;

x2 = x + 4 + textwidth( query1 );

outtextxy( x2, y, achar );

if ( pick != 56 )

{

y = 100;

/\*\*\*\*\*\*\*\* get number of elements desired \*\*\*\*\*/

gprintf( &x, &y, query2 );

x2 = x + 4 + textwidth( query2 );

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

nstring[i] = 0; /\* used to initialize string to nulls \*/

get\_number( elements, tpower, nstring, &x2, &y );

if ( \*elements == 0 || \*elements > MAXNUM ) \*elements = MAXNUM;

y += textheight("H" ) + 1;

/\*\*\*\*\*\* get random or worst case \*\*\*\*\*\*\*\*\*\*\*/

gprintf( &x, &y, query3 );

bchar = 0;

while( bchar != 82 && bchar != 87 )

{

bchar = toupper( getch( ) );

if ( bchar == 13 ) bchar = 82;

}

\*random = ( bchar ^ 87 ); /\* XOR checks for (W)orst \*/

achar[0] = bchar;

x2 = x + 4 + textwidth( query3 );

outtextxy( x2, y, achar );

y += textheight( "H" ) + 1;

/\*\*\*\*\*\* get delay factor \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

gprintf( &x, &y, query4 );

x2 = x + 4 + textwidth( query4 );

\*tpower = TIMES;

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

nstring[i] = 0; /\* used to initialize string to nulls \*/

get\_number( delay\_factor, tpower, nstring, &x2, &y );

}

switch( pick - 48 )

{

case 1:

return( bubble );

case 2:

return( delayed );

case 3:

return( shell );

case 4:

return( shell\_metzner );

case 5:

return( quick );

case 6:

return( insertion );

case 7:

return( all );

default:

return( stop );

}

}

/\*\*\*\*\* print\_menu \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

prints the selection menu to the graphics

screen like so:

1. Bubble Sort

2. Delayed Exchange Sort

3. Shell Sort

4. Shell Metzner Sort

5. Quicksort

6. Insertion Sort

7. All

8. Exit to Dos

called by: pick\_sort

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void print\_menu( char \*mysorts[] )

{

int x, y; /\* screen coordinates \*/

int i;

x = 240;

y = 10;

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

{

gprintf( &x, &y, "%d. %s", i+1, mysorts[i] );

y += textheight ( "H" ) + 1;

}

}

/\*\*\*\*\* get\_number \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

A recursive routine that accepts numbers using the getch() function,

and displays them on the graphics screen. Only the characters '0' to '9'

and CR are accepted -- the rest are ignored.

called by: pick\_sort, get\_number

parameters:

int \*a\_number holds the integer and returns it to the

calling function

int \*times maximum number of times that get\_number

will be called. acts as a flag to stop

the routine when the user enters the maximum

allowed digits or hits Carriage Return (CR).

char \*tstring Returns the string equivalent of \*a\_number

i.e. if \*a\_number = 123, \*tstring = "123".

It is initialized to all nulls before the initial

pass and its length is used to determine the

power of ten needed for each digit that is entered.

int \*x, \*y coordinates on the graphics screen to display

the digits as they are entered. \*x is increased

with

each call by the width of a character using the

textwidth function.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void get\_number( int \*a\_number, int \*times, char \*tstring, int \*x, int

\*y )

{

int power; /\* power of 10 to multiply a digit by \*/

char achar[2];

char bchar = 0;

achar[1] = 0;

while ( bchar <= 47 || bchar >= 59 ) /\* allow digits 0-9 \*/

{

bchar = getch();

if ( bchar == 13 ) /\* 13 = CR; if the user hits ENTER \*/

{

bchar = 48;

\*times = 0;

break;

}

}

if ( \*times )

{

achar[0] = bchar;

outtextxy( \*x, \*y, achar );

\*x = \*x + textwidth( achar );

tstring[TIMES - ( (\*times)--)] = achar[0];

if ( \*times )

get\_number( a\_number, times, tstring, x, y );

}

power = (int)( pow10(( strlen( tstring ) - ((\*times) + 1))));

bchar = tstring[\*times];

\*a\_number += ( power \* ( bchar - 48 ));

(\*times )++;

}

/\*\*\*\*\* driver \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

driver runs the sorts using parameters sent to it.

It gets a sort type, the address of the array to sort, the number of

elements, the random/worst case status and the delay factor and sets

them

in motion.

called by: main

calls: Setscreen, gprintf, all the sort functions

parameters:

enum sort atype the desired sort

int \*array the address of the array to sort

int elements how many elements

int random random = 1 worst case = 0

int delay\_factor 0 = no delay; 1000 = 1 second delay for

each switching of elements. The idea is to slow

down the animation so the user gets a feel for

what's going on. 1000 is \_very\_ slow.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void driver( enum sort atype, int \*array, int elements,

int random, int delay\_factor )

{

switch( atype )

{

case all :

case bubble :

Setscreen( array, elements, random );

gprintf( &xaxis, &yaxis, \*(sorts + bubble) );

Bubble( array, elements, delay\_factor );

if ( atype != all ) break; else delay( 1350 );

case delayed:

Setscreen( array, elements, random );

gprintf( &xaxis, &yaxis, \*(sorts + delayed) );

Delayed( array, elements, delay\_factor );

if ( atype != all ) break; else delay( 1350 );

case shell :

Setscreen( array, elements, random );

gprintf( &xaxis, &yaxis, \*(sorts + shell ));

Shell( array, elements, delay\_factor );

if ( atype != all ) break; else delay( 1350 );

case shell\_metzner:

Setscreen( array, elements, random );

gprintf( &xaxis, &yaxis, \*(sorts + shell\_metzner) );

Shell\_Metzner( array, elements, delay\_factor );

if ( atype != all ) break; else delay( 1350 );

case quick :

Setscreen( array, elements, random );

gprintf( &xaxis, &yaxis, \*(sorts + quick) );

Quicksort( array, 0, elements - 1, delay\_factor );

if ( atype != all ) break; else delay( 1350 );

case insertion:

Setscreen( array, elements, random );

gprintf( &xaxis, &yaxis, \*(sorts + insertion) );

Insertion( array, elements, delay\_factor );

if ( atype != all ) break; else delay( 1350 );

case stop:

default:;

}

}

/\*\*\*\*\* initialize \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* \*/

/\* Initializes the Borland graphics drivers. \*/

/\* \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Initialize( void )

{

int GraphDriver; /\* The Graphics device driver \*/

int GraphMode; /\* The Graphics mode value \*/

int ErrorCode; /\* Reports any graphics errors \*/

// if( registerbgidriver( ) < 0 ) exit(1);

/\* if( registerbgidriver( Herc\_driver ) < 0 ) exit(2);

if( registerbgidriver( EGAVGA\_driver ) <0 ) exit(3); \*/

GraphDriver = DETECT; /\* Request auto-detection \*/

initgraph( &GraphDriver, &GraphMode, "c:\tc\bgi" );

ErrorCode = graphresult(); /\* Read result of initialization\*/

if( ErrorCode != grOk ) /\* Error occured during init \*/

{

printf(" Graphics System Error: %s

", grapherrormsg( ErrorCode )

);

exit( 1 );

}

/\* settextstyle( SMALL\_FONT, HORIZ\_DIR, 0 ); \*/

}

/\*\*\*\*\* gprintf \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* \*/

/\* Used like PRINTF except the output is sent to \*/

/\* the screen in graphics mode at the specified \*/

/\* co-ordinate. From Borland International. \*/

/\* \*/

/\* The return value from gprintf is not used. \*/

/\* \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int gprintf( int \*xloc, int \*yloc, char \*fmt, ... )

{

va\_list argptr; /\* Argument list pointer \*/

char str[80]; /\* Buffer to build string into \*/

int count; /\* Result of vsprintf for return \*/

va\_start( argptr, fmt ); /\* Initialize va\_functions

\*/

count = vsprintf( str, fmt, argptr ); /\* prints string to buffer

\*/

outtextxy( \*xloc, \*yloc, str ); /\* Send string in graphics mode

\*/

va\_end( argptr ); /\* Close va\_ functions

\*/

return( count ); /\* Return the conversion count

\*/

}

/\*\*\*\*\* Setscreen

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Initializes graphics screen for the sort.

called by: driver

parameters:

int \*array the array to sort

int elements how many elements

int random random = 1 or worst case = 0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Setscreen( int \*array, int elements, int random )

{

int j;

cleardevice();

if ( random )

{

randomize();

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

{

\*( array + j) = random( elements );

putpixel( 3\*j, \*(array+j), 10);

}

}

else /\* initialize worst case \*/

{

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

{

\*(array + j) = elements - j;

putpixel( 3\*j, \*(array+j), 10);

}

}

}

/\*\*\*\*\* Showdata \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* \*/

/\* Displays the values in the first 20 elements \*/

/\* of the array. \*/

/\* \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Showdata( int \*array )

{

int i, j, x, y;

j = 0;

i = 20;

x = 570;

y = 0;

while ( j < i )

{

gprintf( &x, &y, "%2d: %d ", j+1, array[j] );

y += textheight( "H" ) + 1; /\* Advance to next line \*/

j++;

}

}

/\*\*\*\*\* Swap\_Pixels \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* \*/

/\* Swaps the data in two array elements and \*/

/\* changes their respective pixels accordingly. \*/

/\* The turning off and on of pixels is what gives \*/

/\* the illusion of movement. \*/

/\* \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Swap\_Pixels( int \*array, int i, int j, int delay\_factor )

{

int h;

h = \*(array + i);

putpixel( 3 \* i, \*(array + i), 0);

putpixel( 3 \* j, \*(array + i), 10 );

\*(array + i) = \*(array + j);

putpixel( 3 \* j, \*( array + j), 0 );

putpixel( 3 \* i, \*(array + j), 10 );

\*(array + j) = h;

delay( delay\_factor );

return( h );

}

/\*\*\*\*\* Bubble \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Bubble( int \*array, int elements, int delay\_factor )

{

int i,j;

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

for ( j = i + 1; j < elements; j++ )

{

if ((\*(array+i)) > (\*(array+j)))

{

Swap\_Pixels( array, i, j, delay\_factor );

}

}

}

/\*\*\*\*\* Delayed \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Delayed( int \*array, int elements, int delay\_factor )

{

int p, h, k, i, j;

for ( p = 0; p < elements-1; p++ )

{

h = p;

for ( k = p + 1; k < elements ; k++ )

if (\*(array+k) < \*(array+h))

h = k;

if ( p != h )

{

i = h;

j = p;

Swap\_Pixels( array, i, j, delay\_factor );

}

}

}

/\*\*\*\*\* Shell \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Shell( int \*array, int elements, int delay\_factor )

{

int p, f, i, j, m;

p = elements;

while ( p > 1)

{

p /= 2;

/\* gprintf( &xaxis, &yaxis, "%d", p );

y++; \*/

m = elements - p;

do{

f = 0;

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

{

i = j + p;

if (\*(array + j) > \*(array + i))

{

Swap\_Pixels( array, i, j, delay\_factor );

f = 1;

}

}

} while( f );

}

}

/\*\*\*\*\* Shell-Metzner \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Shell\_Metzner( int \*array, int elements, int delay\_factor )

{

int p, k, t, i, j;

p = elements;

p /= 2;

while ( p != 0 )

{

k = elements - p;

for ( t = 0; t < k; t++ )

{

i = t;

while ( i >= 0 )

{

j = i + p;

if (\*(array+j) < \*(array + i))

{

Swap\_Pixels( array, i, j, delay\_factor );

i = i - p;

}

else

break;

}

}

p /= 2;

}

}

/\*\*\*\*\* Quicksort \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Quicksort( int \*array, int left, int right, int delay\_factor )

{

int i, j, t;

if ( right > left )

{

i = left - 1; j = right;

do {

do i++;

while ( array[i] < array[right] );

do j--;

while ( array[j] > array[right] && j > 0 );

t = Swap\_Pixels( array, i, j, delay\_factor );

} while ( j > i );

putpixel( 3\*j, \*(array + j), 0);

array[j] =array[i];

putpixel( 3\*j, \*(array + j), 10 );

putpixel( 3\*i, \*(array + i), 0 );

/\* putpixel( 3\*right, \*(array + right), 0);

// <-- putting this stmt here causes duplicate pixels...

\*/

array[i] =array[right];

putpixel( 3\*i, \*(array + i), 10 );

/\*

On the other hand, this one works... why? maybe if array[i] ==array[right] there's a problem...

\*/

putpixel( 3\*right, \*(array + right), 0 );

array[right] = t;

putpixel( 3\*right, \*(array + right), 10 );

Quicksort( array, left, i - 1, delay\_factor );

Quicksort( array, i + 1, right, delay\_factor );

}

}

/\*\*\*\*\* Insertion \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Insertion( int \*array, int elements, int delay\_factor )

{

int p, j, t;

for ( p = 0; p < elements - 1; p++ )

{

t = \*(array + p + 1);

for ( j = p; j >= 0; j-- )

{

if ( t <= \*(array + j) )

{

\*(array + j + 1) = \*(array + j);

putpixel( 3\*(j + 1), \*(array + j + 1), 10 );

putpixel( 3\*j, \*(array + j + 1), 0 );

delay( delay\_factor );

}

else

break;

}

\*(array + j + 1) = t;

putpixel( 3\*(p + 1), t, 0 );

putpixel( 3\*(j + 1), t, 10 );

}

}

/\*\*\*\*\* end code for csort.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**how to create avl tree with graphic display(UPDATED)**

#include <stdio.h>  
#include <stdlib.h>  
#include <ctype.h>  
#include <dos.h>  
#include <string.h>  
#include <graphics.h>  
#include <conio.h>  
  
struct node  
{  
   int element;  
   node \*left;  
   node \*right;  
   int height;  
};  
  
    typedef struct node \*nodeptr;  
  
    void insert(int,nodeptr &);  
    void del(int, nodeptr &);  
    int deletemin(nodeptr &);  
    void find(int,nodeptr &);  
    nodeptr findmin(nodeptr);  
    nodeptr findmax(nodeptr);  
    void makeempty(nodeptr &);  
    nodeptr nodecopy(nodeptr &);  
    void preorder(nodeptr);  
    void inorder(nodeptr);  
    void postorder(nodeptr);  
    int bsheight(nodeptr);  
    nodeptr singlerotateleft(nodeptr &);  
    nodeptr doublerotateleft(nodeptr &);  
    nodeptr singlerotateright(nodeptr &);  
    nodeptr doublerotateright(nodeptr &);  
    int max(int,int);  
    int nonodes(nodeptr);  
  
  
int gdriver=DETECT,gmode=0,errorcode;  
char element[3];  
int x=1,maxx,midx,xcoord,ycoord,level=320,prevlevel;  
  
void GDisplay(nodeptr p,int xcoord,int ycoord)  
{  
  if (p!=NULL)  
    {  
       level=level/2;  
       setfillstyle(1,BROWN);  
       setcolor(LIGHTGREEN);  
       if(p->left->element!=NULL)  
        line(xcoord,ycoord,xcoord-level,ycoord+50);  
       if(p->right->element!=NULL)  
        line(xcoord,ycoord,xcoord+level,ycoord+50);  
       fillellipse(xcoord,ycoord,10,10);  
       sprintf(element,"%d",p->element,xcoord,ycoord);  
       settextstyle(2,0,4);  
       setcolor(YELLOW);  
       outtextxy(xcoord-7,ycoord-7,element);  
       GDisplay(p->left,xcoord-(level),ycoord+50);  
       GDisplay(p->right,xcoord+(level),ycoord+50);  
       level=level\*2;  
    }  
}//end of GDisplay  
  
  
void insert(int x,nodeptr &p)  
{  
   if (p == NULL)  
   {  
    p = new node;  
    p->element = x;  
    p->left=NULL;  
    p->right = NULL;  
    p->height=0;  
    if (p==NULL)  
        {  
        gotoxy(4,21);  
        printf("Out of Space");  
        }  
   }  
   else  
   {  
    if (x<p->element)  
    {  
       insert(x,p->left);  
       //GDisplay(root,midx,50);  
       if ((bsheight(p->left) - bsheight(p->right))==2)  
       {  
          if (x < p->left->element)  
        p = singlerotateleft(p);    //single rotation left  
          else  
        p = doublerotateleft(p);  //double rotation left  
       }  
    }  
    else if (x>p->element)  
    {  
          insert(x,p->right);  
          //GDisplay(root,midx,50);  
          if ((bsheight(p->right) - bsheight(p->left))==2)  
          {  
        if (x > p->right->element)  
            p = singlerotateright(p);  //single rotation right  
        else  
            p = doublerotateright(p); //double rotation right  
         }  
    }  
    else  
        {  
        gotoxy(4,21);  
        printf("Element Exists");  
        }  
    }  
    int m,n,d;  
    m=bsheight(p->left);  
    n=bsheight(p->right);  
    d=max(m,n);  
    p->height = d + 1;  
}  
  
nodeptr findmin(nodeptr p)  
{  
    if (p==NULL)  
    {  
        gotoxy(4,21); printf("Empty Tree");  
        return p;  
    }  
    else  
    {  
       while(p->left !=NULL)  
        p=p->left;  
       return p;  
    }  
}  
  
nodeptr findmax(nodeptr p)  
{  
    if (p==NULL)  
    {  
       gotoxy(4,21); printf("Empty Tree");  
       return p;  
    }  
    else  
    {  
       while(p->right !=NULL)  
           p=p->right;  
       return p;  
    }  
}  
  
void find(int x,nodeptr &p)  
{  
    if (p==NULL)  
    {  
       gotoxy(4,21);  
       printf("Element not found");  
    }  
    else if (x < p->element)  
       find(x,p->left);  
    else if (x>p->element)  
       find(x,p->right);  
    else  
    {  
       gotoxy(4,21);  
       printf("Element found !");  
    }  
}  
  
void del(int x,nodeptr &p)  
{  
    nodeptr d;  
    if (p==NULL)  
    {  
       gotoxy(4,21);  
       printf("Element not found");  
    }  
    else if ( x < p->element)  
    {  
       del(x,p->left);  
       if ((bsheight(p->left) - bsheight(p->right))==2)  
       {  
          if (x < p->left->element)  
        p = singlerotateleft(p);    //single rotation left  
          else  
        p = doublerotateleft(p);  //double rotation left  
       }  
    }  
    else if (x > p->element)  
    {  
       del(x,p->right);  
       if ((bsheight(p->right) - bsheight(p->left))==2)  
       {  
          if (x > p->right->element)  
        p = singlerotateright(p);  //single rotation right  
          else  
        p = doublerotateright(p); //double rotation right  
       }  
    }  
    else if ((p->left == NULL) && (p->right == NULL))  
    {  
       d=p;  
       free(d);  
       p=NULL;  
       gotoxy(4,21); printf("Element deleted !");  
    }  
    else if (p->left == NULL)  
    {  
      d=p;  
      free(d);  
      p=p->right;  
      gotoxy(4,21); printf("Element deleted !");  
    }  
    else if (p->right == NULL)  
    {  
      d=p;  
      p=p->left;  
      free(d);  
      gotoxy(4,21); printf("Element deleted !");  
    }  
    else  
      p->element = deletemin(p->right);  
}  
  
int deletemin(nodeptr &p)  
{  
    int c;  
    gotoxy(4,21); printf("deltemin");  
    if (p->left == NULL)  
    {  
      c=p->element;  
      p=p->right;  
      return c;  
    }  
    else  
    {  
      c=deletemin(p->left);  
      return c;  
    }  
}  
  
void preorder(nodeptr p)  
{  
    if (p!=NULL)  
    {  
      printf("%d-->",p->element);  
      preorder(p->left);  
      preorder(p->right);  
    }  
}  
  
void inorder(nodeptr p)  
{  
    if (p!=NULL)  
    {  
       inorder(p->left);  
       printf("%d-->",p->element);  
       inorder(p->right);  
    }  
}  
  
void postorder(nodeptr p)  
{  
    if (p!=NULL)  
    {  
       postorder(p->left);  
       postorder(p->right);  
       printf("%d-->",p->element);  
    }  
}  
  
int max(int value1, int value2)  
{  
    if(value1 > value2)  
      return value1;  
    else  
      return value2;  
}  
  
int bsheight(nodeptr p)  
{  
    int t;  
    if (p == NULL)  
        return -1;  
    else  
    {  
        t = p->height;  
        return t;  
    }  
}  
  
nodeptr singlerotateleft(nodeptr &p1)  
{  
    nodeptr p2;  
    p2 = p1->left;  
    p1->left = p2->right;  
    p2->right = p1;  
    p1->height = max(bsheight(p1->left),bsheight(p1->right)) + 1;  
    p2->height = max(bsheight(p2->left),p1->height) + 1;  
    return p2;  
}  
  
nodeptr singlerotateright(nodeptr &p1)  
{  
    nodeptr p2;  
    p2 = p1->right;  
    p1->right = p2->left;  
    p2->left = p1;  
    p1->height = max(bsheight(p1->left),bsheight(p1->right)) + 1;  
    p2->height = max(p1->height,bsheight(p2->right)) + 1;  
    return p2;  
}  
  
nodeptr doublerotateleft(nodeptr &p1)  
{  
    p1->left = singlerotateright(p1->left);  
    return singlerotateleft(p1);  
}  
  
nodeptr doublerotateright(nodeptr &p1)  
{  
    p1->right = singlerotateleft(p1->right);  
    return singlerotateright(p1);  
}  
  
int count=0;  
int nonodes(nodeptr p)  
{  
    if (p!=NULL)  
    {  
        nonodes(p->left);  
        nonodes(p->right);  
        count++;  
    }  
    return count;  
}  
  
void twolinebox(int x1,int y1,int x2,int y2){  
 int x,y;  
 textcolor(WHITE);  
 gotoxy(x1,y1); cprintf("É"); //201  
 gotoxy(x2,y1); cprintf("»"); //187  
  for(y=y1+1;y<y2;y++){  
    gotoxy(x1,y); cprintf("º"); //186  
    gotoxy(x2,y); cprintf("º"); //186  
  }  
 gotoxy(x1,y2); cprintf("È"); //200  
 gotoxy(x2,y2); cprintf("¼"); //188  
  for(x=x1+1;x<x2;x++){  
    gotoxy(x,y1); cprintf("Í"); //205  
    gotoxy(x,y2); cprintf("Í"); //205  
  }  
 gotoxy(x1+1,y1+1);  
}  
  
void cprintxy(int x,int y,int color,char string[]){  
 textcolor(color);  
 gotoxy(x,y); cprintf("%s",string);  
}  
  
  
void center(int y,int color,char string[]){  
int x=(80-strlen(string)+1)/2;  
textcolor(color);  
gotoxy(x,y);cprintf("%s",string);  
}  
  
  
void Temp(void){  
int x,y;  
clrscr();  
twolinebox(1,1,80,24);  
for(y=23;y>=1;y--){  
    sound(60\*y);  
    center(y,YELLOW,"[Adelson-Velskii and Landis Tree]");  
    gotoxy(2,y+1); clreol();  
    twolinebox(1,1,80,24);  
    delay(40);  
    nosound();  
}  
center(1,YELLOW,"[Adelson-Velskii and Landis Tree]");  
for(x=74;x>=3;x--){  
    sound(50\*x);  
    cprintxy(x,5,RED,"Press:");   clreol();  
    twolinebox(1,1,80,24);  
    center(1,YELLOW,"[Adelson-Velskii and Landis Tree]");  
    delay(20);  
    nosound();  
}  
twolinebox(1,1,80,12);  
twolinebox(1,1,80,24);  
center(1,YELLOW,"[Adelson-Velskii and Landis Tree]");  
cprintxy(20,3,GREEN,"[A]- Insertion");  
cprintxy(20,4,GREEN,"[B]- Find Minimum");  
cprintxy(20,5,GREEN,"[C]- Find Maximum");  
cprintxy(20,6,GREEN,"[D]- Search Node");  
cprintxy(20,7,GREEN,"[E]- Display Tree");  
cprintxy(43,3,GREEN,"[F]- Delete Node");  
cprintxy(43,4,GREEN,"[G]- Preorder");  
cprintxy(43,5,GREEN,"[H]- Inorder");  
cprintxy(43,6,GREEN,"[I]- Postorder");  
cprintxy(43,7,GREEN,"[J]- Height");  
cprintxy(20,9,GREEN,"[any key]- Quit...");  
  
cprintxy(20,11,LIGHTRED,"Enter your choice:   ");  
}  
  
void main()  
{  
  
    nodeptr root,min,max;  
    int a,findele,delele,leftele,rightele,flag;  
    char choice,value[2];  
    char ch='Y';  
    root = NULL;  
    textmode(C80);  
    Temp();  
    do  
    {  
        clrscr();  
        twolinebox(1,1,80,12);  
        twolinebox(1,1,80,24);  
        center(1,YELLOW,"[Adelson-Velskii and Landis Tree]");  
        cprintxy(5,3,LIGHTRED,"Press: ");  
        cprintxy(20,3,GREEN,"[A]- Insertion");  
        cprintxy(20,4,GREEN,"[B]- Find Minimum");  
        cprintxy(20,5,GREEN,"[C]- Find Maximum");  
        cprintxy(20,6,GREEN,"[D]- Search Node");  
        cprintxy(20,7,GREEN,"[E]- Display Tree");  
        cprintxy(43,3,GREEN,"[F]- Delete Node");  
        cprintxy(43,4,GREEN,"[G]- Preorder");  
        cprintxy(43,5,GREEN,"[H]- Inorder");  
        cprintxy(43,6,GREEN,"[I]- Postorder");  
        cprintxy(43,7,GREEN,"[J]- Height");  
        cprintxy(20,9,GREEN,"[any key]- Quit...");  
  
        cprintxy(20,11,LIGHTRED,"Enter your choice:\>");  
        choice=getch();  
        switch(toupper(choice))  
        {  
        case 'A':  
            do{  
  
            gotoxy(4,14); printf("Enter node value: ");  
            a=atoi(gets(value));  
            if(atoi(value)==0)  
            {  
                gotoxy(4,21); printf("Error!!! Enter a valid integer value only.");  
                gotoxy(4,22); printf("Press any key to continue...");  
                getch();  
            }  
            }while(atoi(value)==0);  
            insert(a,root);  
            gotoxy(4,15);  
            inorder(root);  
            /\*  
            initgraph(&gdriver,&gmode,"c:\tc\bgi");  
            errorcode = graphresult();  
            maxx=getmaxx();  
            midx=maxx/2,xcoord=midx/2,ycoord=40;  
            if (errorcode != grOk)  
             {  
               printf("Graphics error: %s  
", grapherrormsg(errorcode));  
               printf("Press any key to stop");  
               getch();  
               exit(1);  
             }  
             cleardevice();  
             GDisplay(root,midx,50);  
             getch();  
             restorecrtmode();  
             \*/  
            break;  
        case 'B':  
            if (root !=NULL)  
            {  
            min=findmin(root);  
            gotoxy(4,14); printf("Min element :  %d",min->element);  
            }  
            break;  
         case 'C':  
            if (root !=NULL)  
            {  
            max=findmax(root);  
            gotoxy(4,14); printf("Max element :  %d",max->element);  
            }  
            break;  
         case 'D':  
            gotoxy(4,14); printf("Search node :");  
            scanf("%d",&findele);  
            if (root != NULL)  
                find(findele,root);  
            break;  
         case 'E':  
            initgraph(&gdriver,&gmode,"c:\tc\bgi");  
            errorcode = graphresult();  
            maxx=getmaxx();  
            midx=maxx/2;  
            xcoord=midx/2;  
            ycoord=40;  
  
            if (errorcode != grOk)  
             {  
               printf("Graphics error: %s  
", grapherrormsg(errorcode));  
               printf("Press any key to stop");  
               getch();  
               exit(1);  
             }  
             cleardevice();  
             setbkcolor(LIGHTBLUE);  
             settextstyle(2,0,5);  
             outtextxy(20,10,"Adelson-Velskii and Landis Tree");  
             GDisplay(root,midx,50);  
             outtextxy(20,440,"Programmed by: Frederick Badion");  
             outtextxy(20,450,"Polytechnic University of the Philippines");  
             outtextxy(20,460,"2nd year Bachelor of Science in Computer   
Science");  
             outtextxy(320,440,"A partial fulfilment for the subject: ");  
             outtextxy(320,450,"Design and Analysis of Algorithm");  
             outtextxy(320,460,"Prof. Ria Sagum -Chairperson BSCS-CCMIT  
PUP-Sta.Mesa");  
             getch();  
             restorecrtmode();  
            break;  
         case 'F':  
            gotoxy(4,14); printf("Delete Node: ");  
            scanf("%d",&delele);  
            del(delele,root);  
            getch();  
            initgraph(&gdriver,&gmode,"c:\tc\bgi");  
            errorcode = graphresult();  
            maxx=getmaxx();  
            midx=maxx/2,xcoord=midx/2,ycoord=40;  
            if (errorcode != grOk)  
             {  
               printf("Graphics error: %s  
", grapherrormsg(errorcode));  
               printf("Press any key to stop");  
               getch();  
               exit(1);  
             }  
            cleardevice();  
            setbkcolor(LIGHTBLUE);  
            settextstyle(2,0,5);  
            outtextxy(20,10,"Adelson-Velskii and Landis Tree");  
            GDisplay(root,midx,50);  
            getch();  
            restorecrtmode();  
            break;  
  
         case 'G':  
            gotoxy(4,14); printf(" Preorder Printing...");  
            gotoxy(4,15);  
            preorder(root);  
            break;  
  
         case 'H':  
            gotoxy(4,14); printf(" Inorder Printing...");  
            gotoxy(4,15);  
            inorder(root);  
            break;  
  
         case 'I':  
            gotoxy(4,14); printf(" Postorder Printing...");  
            gotoxy(4,15);  
            postorder(root);  
            break;  
         case 'J':  
            gotoxy(4,14); printf(" Height and Depth: %d",bsheight(root));  
            gotoxy(4,15); printf("No. of nodes: %d",nonodes(root));  
            break;  
        }  
        gotoxy(4,22); printf(" Do you want to continue (y/n)?");  
        ch=toupper(getch());  
    }while(ch!='N');  
}

AvlTree.cpp - Implementation for AVL tree

#include "AvlTree.h"

#include <iostream.h>

/\*\*

\* Implements an unbalanced Avl search tree.

\* Note that all "matching" is based on the compares method.

\*/

/\*\*

\* Construct the tree.

\*/

template <class Comparable>

AvlTree<Comparable>::AvlTree( const Comparable & notFound ) :

ITEM\_NOT\_FOUND( notFound ), root( NULL )

{

}

/\*\*

\* Copy constructor.

\*/

template <class Comparable>

AvlTree<Comparable>::AvlTree( const AvlTree<Comparable> & rhs ) :

ITEM\_NOT\_FOUND( rhs.ITEM\_NOT\_FOUND ), root( NULL )

{

\*this = rhs;

}

/\*\*

\* Destructor for the tree.

\*/

template <class Comparable>

AvlTree<Comparable>::~AvlTree( )

{

makeEmpty( );

}

/\*\*

\* Insert x into the tree; duplicates are ignored.

\*/

template <class Comparable>

void AvlTree<Comparable>::insert( const Comparable & x )

{

insert( x, root );

}

/\*\*

\* Remove x from the tree. Nothing is done if x is not found.

\*/

template <class Comparable>

void AvlTree<Comparable>::remove( const Comparable & x )

{

cout << "Sorry, remove unimplemented; " << x <<

" still present" << endl;

}

/\*\*

\* Find the smallest item in the tree.

\* Return smallest item or ITEM\_NOT\_FOUND if empty.

\*/

template <class Comparable>

const Comparable & AvlTree<Comparable>::findMin( ) const

{

return elementAt( findMin( root ) );

}

/\*\*

\* Find the largest item in the tree.

\* Return the largest item of ITEM\_NOT\_FOUND if empty.

\*/

template <class Comparable>

const Comparable & AvlTree<Comparable>::findMax( ) const

{

return elementAt( findMax( root ) );

}

/\*\*

\* Find item x in the tree.

\* Return the matching item or ITEM\_NOT\_FOUND if not found.

\*/

template <class Comparable>

const Comparable & AvlTree<Comparable>::

find( const Comparable & x ) const

{

return elementAt( find( x, root ) );

}

/\*\*

\* Make the tree logically empty.

\*/

template <class Comparable>

void AvlTree<Comparable>::makeEmpty( )

{

makeEmpty( root );

}

/\*\*

\* Test if the tree is logically empty.

\* Return true if empty, false otherwise.

\*/

template <class Comparable>

bool AvlTree<Comparable>::isEmpty( ) const

{

return root == NULL;

}

/\*\*

\* Print the tree contents in sorted order.

\*/

template <class Comparable>

void AvlTree<Comparable>::printTree( ) const

{

if( isEmpty( ) )

cout << "Empty tree" << endl;

else

printTree( root );

}

/\*\*

\* Deep copy.

\*/

template <class Comparable>

const AvlTree<Comparable> &

AvlTree<Comparable>::

operator=( const AvlTree<Comparable> & rhs )

{

if( this != &rhs )

{

makeEmpty( );

root = clone( rhs.root );

}

return \*this;

}

/\*\*

\* Internal method to get element field in node t.

\* Return the element field or ITEM\_NOT\_FOUND if t is NULL.

\*/

template <class Comparable>

const Comparable & AvlTree<Comparable>::elementAt( AvlNode<Comparable> \*t ) const

{

if( t == NULL )

return ITEM\_NOT\_FOUND;

else

return t->element;

}

/\*\*

\* Internal method to insert into a subtree.

\* x is the item to insert.

\* t is the node that roots the tree.

\*/

template <class Comparable>

void AvlTree<Comparable>::insert( const Comparable & x, AvlNode<Comparable> \* & t ) const

{

if( t == NULL )

t = new AvlNode<Comparable>( x, NULL, NULL );

else if( x < t->element )

{

insert( x, t->left );

if( height( t->left ) - height( t->right ) == 2 )

if( x < t->left->element )

rotateWithLeftChild( t );

else

doubleWithLeftChild( t );

}

else if( t->element < x )

{

insert( x, t->right );

if( height( t->right ) - height( t->left ) == 2 )

if( t->right->element < x )

rotateWithRightChild( t );

else

doubleWithRightChild( t );

}

else

; // Duplicate; do nothing

t->height = max( height( t->left ), height( t->right ) ) + 1;

}

/\*\*

\* Internal method to find the smallest item in a subtree t.

\* Return node containing the smallest item.

\*/

template <class Comparable>

AvlNode<Comparable> \*

AvlTree<Comparable>::findMin( AvlNode<Comparable> \*t ) const

{

if( t == NULL)

return t;

while( t->left != NULL )

t = t->left;

return t;

}

/\*\*

\* Internal method to find the largest item in a subtree t.

\* Return node containing the largest item.

\*/

template <class Comparable>

AvlNode<Comparable> \*

AvlTree<Comparable>::findMax( AvlNode<Comparable> \*t ) const

{

if( t == NULL )

return t;

while( t->right != NULL )

t = t->right;

return t;

}

/\*\*

\* Internal method to find an item in a subtree.

\* x is item to search for.

\* t is the node that roots the tree.

\* Return node containing the matched item.

\*/

template <class Comparable>

AvlNode<Comparable> \*

AvlTree<Comparable>::find( const Comparable & x, AvlNode<Comparable> \*t ) const

{

while( t != NULL )

if( x < t->element )

t = t->left;

else if( t->element < x )

t = t->right;

else

return t; // Match

return NULL; // No match

}

/\*\*

\* Internal method to make subtree empty.

\*/

template <class Comparable>

void AvlTree<Comparable>::makeEmpty( AvlNode<Comparable> \* & t ) const

{

if( t != NULL )

{

makeEmpty( t->left );

makeEmpty( t->right );

delete t;

}

t = NULL;

}

/\*\*

\* Internal method to clone subtree.

\*/

template <class Comparable>

AvlNode<Comparable> \*

AvlTree<Comparable>::clone( AvlNode<Comparable> \* t ) const

{

if( t == NULL )

return NULL;

else

return new AvlNode<Comparable>( t->element, clone( t->left ),

clone( t->right ), t->height );

}

/\*\*

\* Return the height of node t or -1 if NULL.

\*/

template <class Comparable>

int AvlTree<Comparable>::height( AvlNode<Comparable> \*t ) const

{

return t == NULL ? -1 : t->height;

}

/\*\*

\* Return maximum of lhs and rhs.

\*/

template <class Comparable>

int AvlTree<Comparable>::max( int lhs, int rhs ) const

{

return lhs > rhs ? lhs : rhs;

}

/\*\*

\* Rotate binary tree node with left child.

\* For AVL trees, this is a single rotation for case 1.

\* Update heights, then set new root.

\*/

template <class Comparable>

void AvlTree<Comparable>::rotateWithLeftChild( AvlNode<Comparable> \* & k2 ) const

{

AvlNode<Comparable> \*k1 = k2->left;

k2->left = k1->right;

k1->right = k2;

k2->height = max( height( k2->left ), height( k2->right ) ) + 1;

k1->height = max( height( k1->left ), k2->height ) + 1;

k2 = k1;

}

/\*\*

\* Rotate binary tree node with right child.

\* For AVL trees, this is a single rotation for case 4.

\* Update heights, then set new root.

\*/

template <class Comparable>

void AvlTree<Comparable>::rotateWithRightChild( AvlNode<Comparable> \* & k1 ) const

{

AvlNode<Comparable> \*k2 = k1->right;

k1->right = k2->left;

k2->left = k1;

k1->height = max( height( k1->left ), height( k1->right ) ) + 1;

k2->height = max( height( k2->right ), k1->height ) + 1;

k1 = k2;

}

/\*\*

\* Double rotate binary tree node: first left child.

\* with its right child; then node k3 with new left child.

\* For AVL trees, this is a double rotation for case 2.

\* Update heights, then set new root.

\*/

template <class Comparable>

void AvlTree<Comparable>::doubleWithLeftChild( AvlNode<Comparable> \* & k3 ) const

{

rotateWithRightChild( k3->left );

rotateWithLeftChild( k3 );

}

/\*\*

\* Double rotate binary tree node: first right child.

\* with its left child; then node k1 with new right child.

\* For AVL trees, this is a double rotation for case 3.

\* Update heights, then set new root.

\*/

template <class Comparable>

void AvlTree<Comparable>::doubleWithRightChild( AvlNode<Comparable> \* & k1 ) const

{

rotateWithLeftChild( k1->right );

rotateWithRightChild( k1 );

}

/\*\*

\* Internal method to print a subtree in sorted order.

\* t points to the node that roots the tree.

\*/

template <class Comparable>

void AvlTree<Comparable>::printTree( AvlNode<Comparable> \*t ) const

{

if( t != NULL )

{

printTree( t->left );

cout << t->element << endl;

printTree( t->right );

}

}