ภาคผนวก 1
การดำเนินการเพื่อวิ่งโปรแกรม

ภาษา C เป็น compiled language การสั่งวิ่งโปรแกรม จึงต้องมีขั้นตอนเหมือนภาษาอื่น ๆ ที่เป็น compiled languageซึ่งต่างกับภาษาเบสิคที่เราพิมพ์คำว่า RUN เมื่อประสงค์จะวิ่งโปรแกรม การวิ่งโปรแกรมภาษา C จะเกิดขึ้นได้เมื่อมี main ( ) โดยมีขั้นตอนเพื่อสั่งวิ่งดังนี้

1. สร้าง source programหรือ source file หรือ text file ซึ่งการทำงานในขั้นนี้จะทำงานโดย text editor หรือ word processor และเราจะต้องใช้ชื่อไฟล์นี้ว่า name.c คำว่า name หมายถึง ชื่อโปรแกรม

2. การแปลคำสั่ง (compilation) ขั้นนี้คอมไพเลอร์จะรับ source file มาแปลเป็น assemble language file ตามกฎเกณฑ์ของภาษา C หากพบข้อผิดพลาดเช่นผิดไวยากรณ์ก็จะแจ้ง error message ออกมาเพื่อให้แก้ไขให้ถูกต้อง โดยปกติชื่อไฟล์นี้ (assembly language file)จะใช้ชื่อว่า name.asm หรือคล้าย ๆ กับแบบนี้

3. แปลงรหัส (assembly) ขั้นนี้แอสเซมเบลอร์จะรับ assembly language file มาแปลงเป็น relocatable object code file ซึ่งจะเป็นข้อมูลและคำสั่งที่จำเป็นสำหรับ linker โดยปกติจะใช้ชื่อ relocatable object code file ว่า name.o

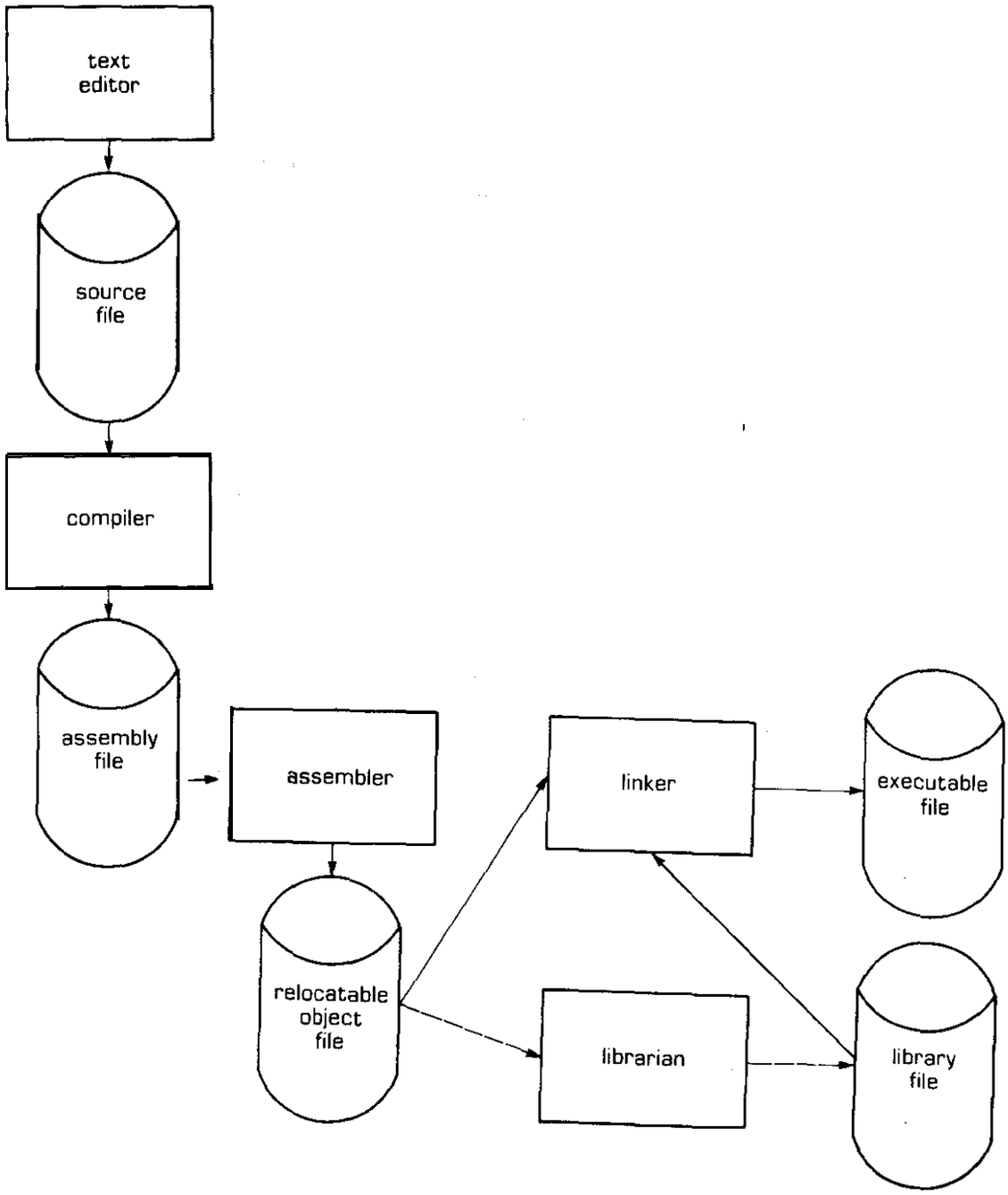
4. linkage ขั้นนี้ linkerจะตรวจสอบ relocatable object code file ว่าจำเป็นต้องใช้ฟังก์ชันและ routine ใดใน library fileบ้าง ถ้าพบว่าต้องใช้ก็จะเชื่อมโยงฟังก์ชัน และ routine เหล่านั้นเข้าด้วยกันเป็นไฟล์ใหม่ เรียกว่า - executable code file ขั้นนี้จะเป็นไฟล์ของภาษาเครื่องล้วน ๆ ที่พร้อมจะทำงานได้ทันที ไฟล์นี้โดยมากจะใช้ชื่อว่า name.exe หรือ name.com

โดยปกติ linker จะมีความสามารถที่เชื่อมโยง relocatable object code fileหลาย ๆ ไฟล์เข้าด้วยกันได้ในเวลาเดียวกันโดยไม่ต้องแยก linkage ครางละไฟล์

อย่างไรก็ตามหาก source file นั้นประกอบไปด้วย routine ที่เตรียมไว้เพื่อใช้งานในโปรแกรมอื่น กรณีนี้ librarian จะทำงานแทน linker librarian จะรวม relocatable code fileต่าง ๆ เข้าไว้ด้วยกันเป็นไฟล์ใหม่เรียกว่า library file ซึ่งก็สามารถเรียกใช้ได้ในขณะที่ทำการ linkge ได้ เราจะใช้ชื่อไฟล์นี้ว่า name.lib

5. สั่งวิ่งโปรแกรม จากขั้นที่ 4 เราจะได้ executable file ที่พร้อมที่จะทำงานแล้ว เมื่อเราพิมพ์ชื่อไฟล์ (name) โปรแกรมจะวิ่งหรือทำงานตามความต้องการของเราทันที

ภายแสดงการดำเนินการทั้งปวงที่กล่าวมาแล้วปรากฏดังนี้



สำหรับคำสั่ง (command) ที่ใช้เพื่อคอมไพล์ และเชื่อมโยงคำสั่งนั้นแตกต่างกันไปตามรุ่นของคอมไพเลอร์แต่โดยปกติจะใช้คำสั่งเหล่านี้ (ขอให้ตรวจสอบคู่มือคอมไพเลอร์ของท่านดูก่อนเพราะอาจใช้คำสั่งต่างกัน)

cc name.c	ใช้สำหรับการคอมไพล์
asm name.asm	ใช้สำหรับขั้น assembly
link name.0 my.lib system.lib	ใช้สำหรับเชื่อมโยง (linkage) กับ library file 2 ไฟล์ คือ my.lib และ system.lib
name	ใช้สำหรับวิ่งโปรแกรม (execute)

สำหรับการบรรจุ routine เข้าไว้ใน library file จะต้องใช้คำสั่งดังนี้

c name.c	ใช้สำหรับการคอมไพล์
asm name.asm	ใช้สำหรับขั้น assembly
lib my.lib name.0	ใช้สำหรับเก็บ routine ลงใน library file ชื่อ my.lib

สำหรับโปรแกรมควบคุมระบบของ UNIX จะผนวกเอาขั้นตอน 1 ถึง 4 ไว้ด้วยกันโดยเพียงแต่เราสั่งว่า cc name.cเท่านั้น จากนั้นถ้าเราประสงค์จะวิ่งโปรแกรมให้สั่งว่า a.out

ภาคผนวก 2
โปรแกรมตัวอย่าง

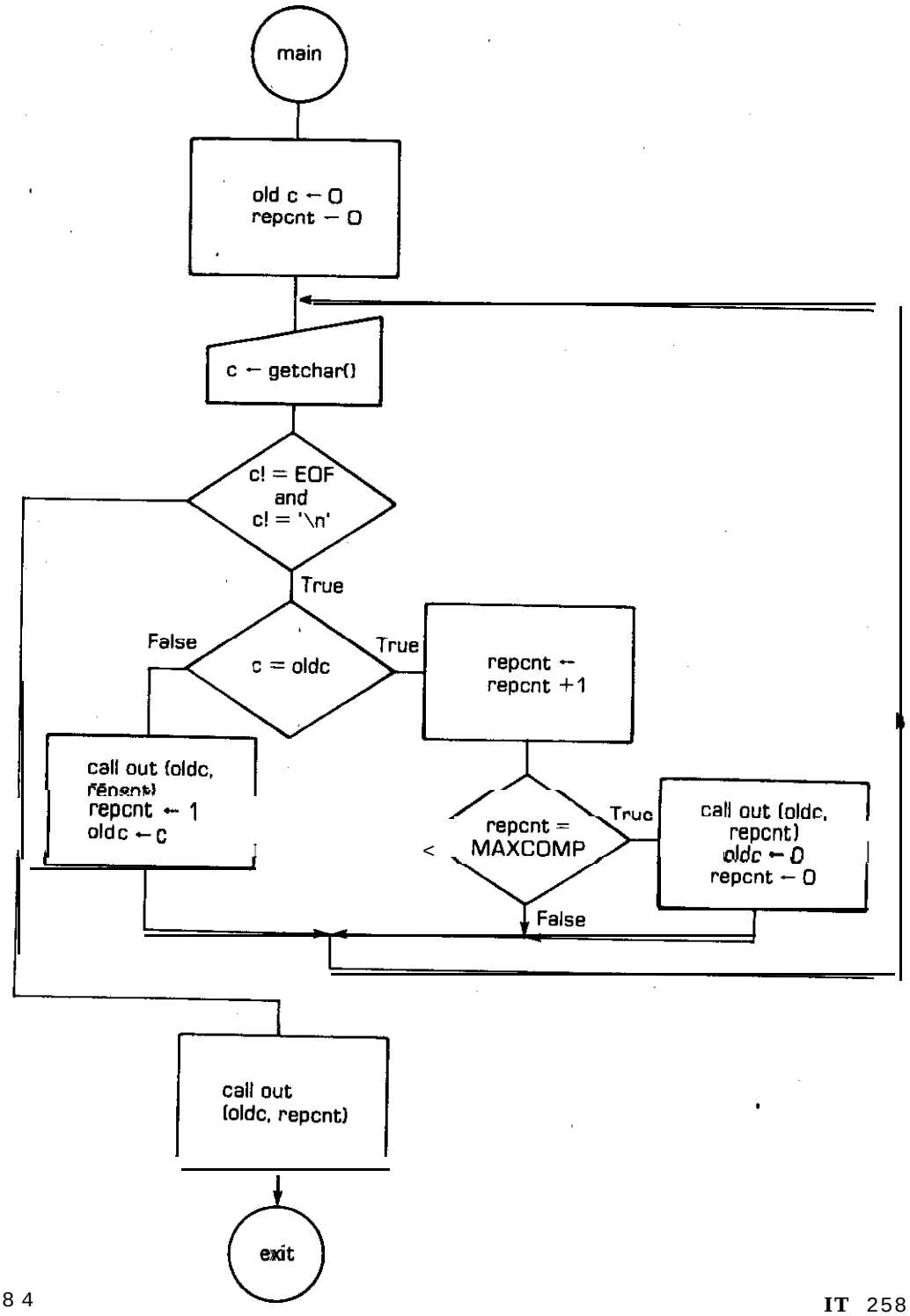
1. compress/decompress

เพื่อประหยัดเนื้อที่ในดิสค์หรือประหยัดเวลาในการสื่อสารระหว่างคอมพิวเตอร์ เราควรใช้กระบวนการที่เรียกว่า compress เพื่อย่นย่อจำนวนอักขระที่ซ้ำ ๆ กันในสตริง วิธีการก็คือใช้โปรแกรม compress ตรวจดูสตริงว่ามีอักขระที่ซ้ำ ๆ กันอยู่ หรือไม่ ถ้าตรวจพบก็ให้ใช้อักขระพิเศษเช่น & พร้อมทั้งจำนวนของอักขระที่ซ้ำกันนั้นและชื่อของอักขระนั้นแทนจำนวนอักขระเหล่านั้น เช่น

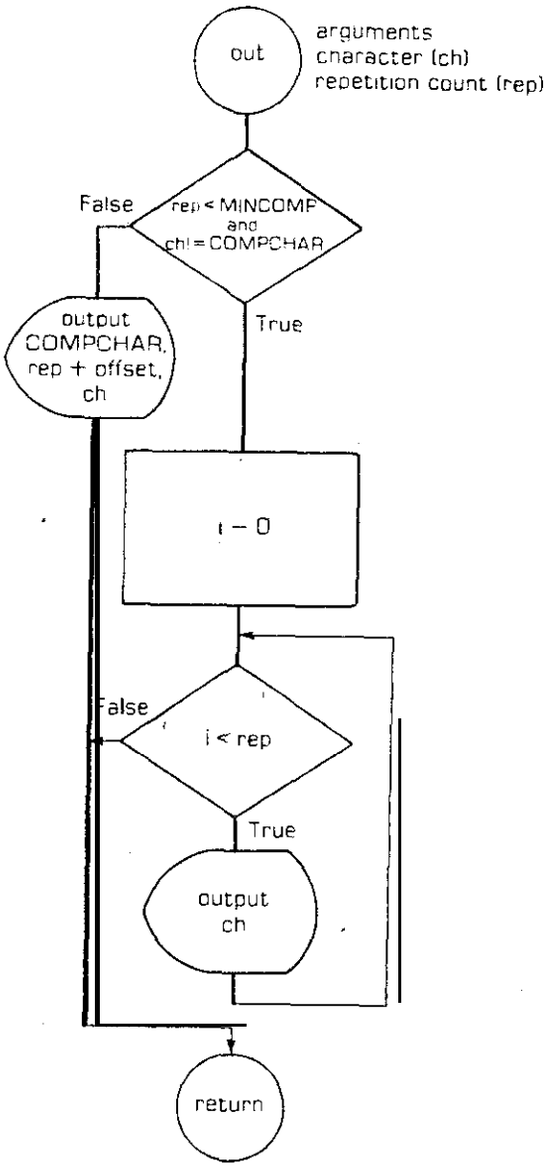
สตริงคือ ABCDDDDDEF โปรแกรม compress จะย่อเป็น ABC&5DEF และปิดท้ายสตริงด้วย '\0' หรือ ' '

ผังควบคุมและโปรแกรมปรากฏดังนี้ จากผังควบคุมสัญลักษณ์  หมายถึง input from terminal และ  หมายถึง output to terminal

Flowchart A.1 COMPRESSION



Flowchart A.1 (continued)



COMPRESS

```
#include "stdio.h"

#define MINCOMP 3 I* minimum number of chars to compress */
#define COMPCHAR '&' /* compression character */
#define OFFSET '0' /* offset to use for repetition count */
#define MAXCOMP 126-OFFSET /* maximum number of chars to compress */
main ( )
/* compresses the character string on stdin and puts it onto stdio */

        int oldc = 0;
        int repcnt = 0;
        int c;
        /* the test for \n is for terminal input */
        while ( ((c = getchar( )) != EOF) && (c! = '\n') )
              {
              if (c = = oldc)
                  {
                  /* if duplicate character. increment repetition counter */


                   repcnt + + ;
                   if (repcnt = = MAXCOMP)

                   out(oldc,repcnt);
                   oldc = 0:
                   repcnt = 0;


         else
               {
               /* output the previous character */
               out(oldc,repcnt):
               repcnt = 1;
               oldc = c;


I' output the last character */
out(oldc,repcnt);
exit(0):
}
```

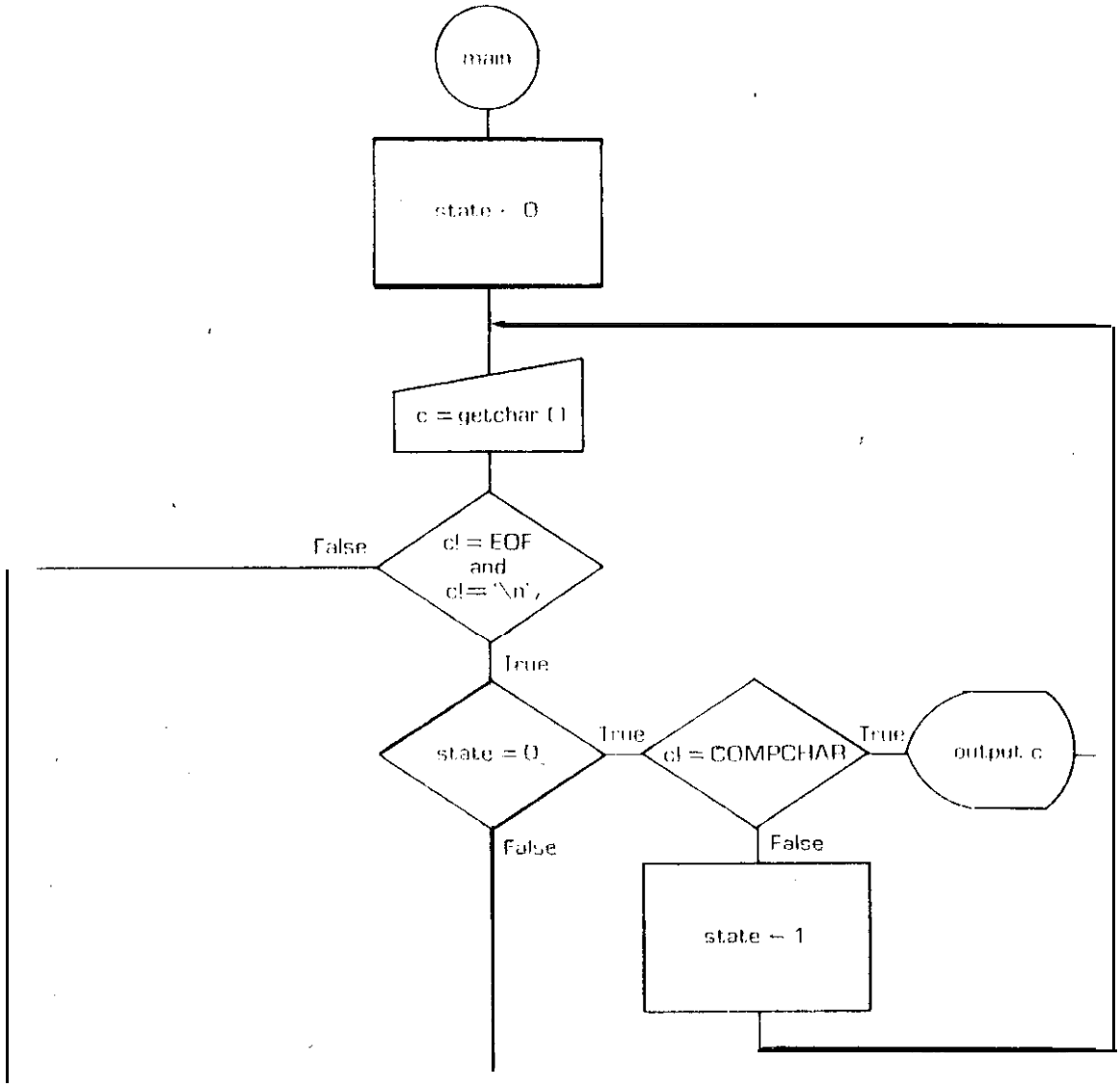
```
out(ch,rep)
int ch;
int rep;
/* outputs appropriate string for a character ch that has appeared rep times */
        {
        int i;
        if (rep < MINCOMP && ch != COMPCHAR)
                {
                /* less than the minimum to compress and not equal to the compression
                   character */
                for (i=0;i<rep;i++)
                        {
                        putchar(ch);
                        }
                }
        else
                        {
                        /* compression string */
                        putchar(COMPCHAR);
                        putchar (rep+OFFSET);
                        putchar (ch);
                        }
                }
```

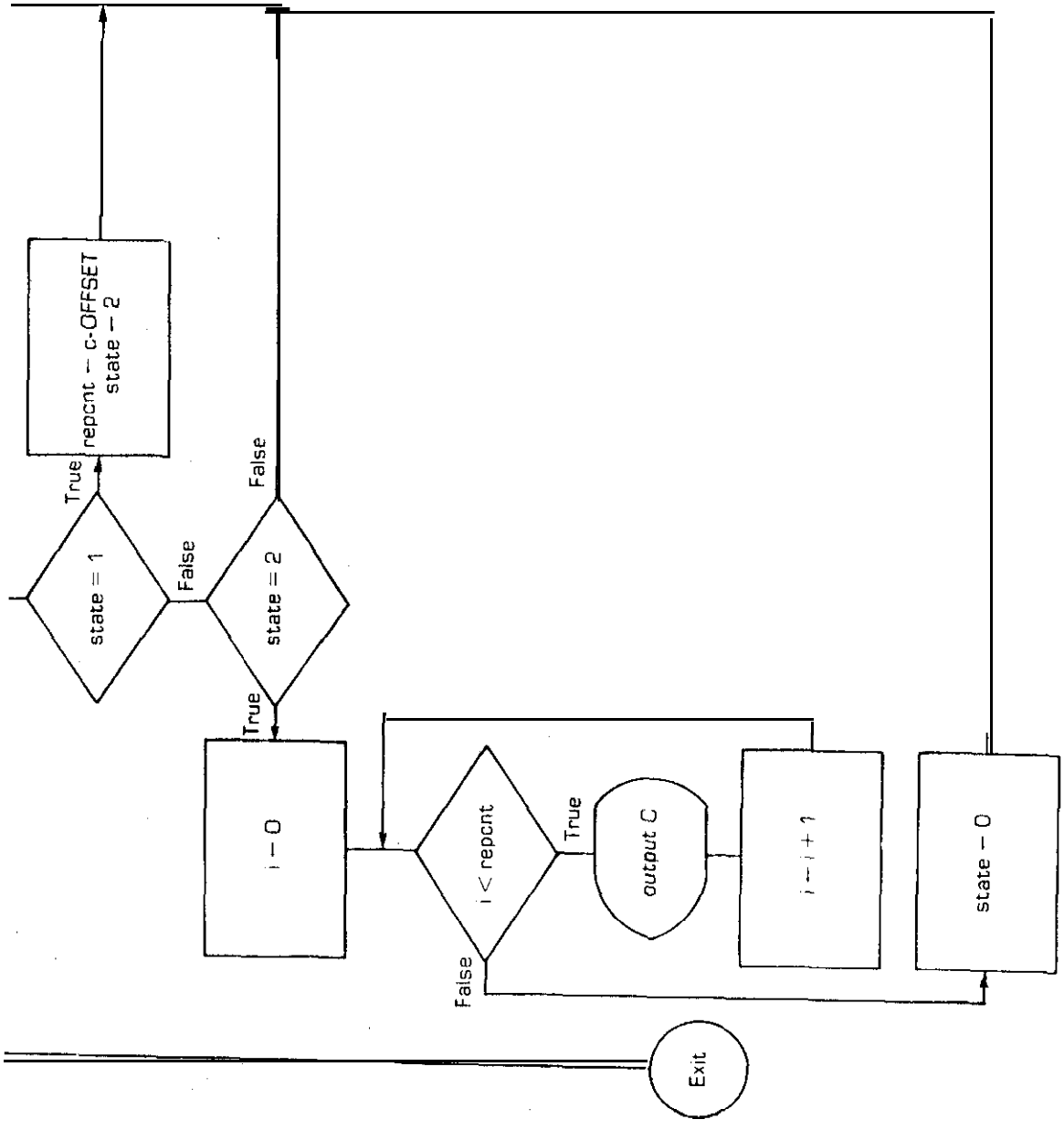
หลังจากนั้น เมื่อประสงค์จะนำข้อมูลที่เก็บ หรือส่งมา ให้ขยายคืนรูปเดิมด้วยโปรแกรม - decompress ดังนี้

DECOMPRESS

```
#include "stdio.h"
#define COMPCHAR '&' /* these #defines might be defined in a #include file that would be
   I" both programs */
#define OFFSET '0
main( )
/* decompresses input on stdin and puts it on stdout */
     {
     int state = 0;
     int c,i,repcnt;
     /* the test for '\n' is for terminal input */
     while ( ((c = getchar( ))! = EOF) && (c! = '\n'))
          {
          switch(state)
               {
          case 0:
               /* if compression character, don't output it*/
               if (c! = COMPCHAR) putchar(c);
               else state = 1;
               break;
          case 1:
               /* previous char was compression char current char is repetition count */
               repcnt = c - OFFSET;
               state = 2;
               break:
          case 2:
               /* this is the character to output repcnt time */
               for (i = 0;i<repcnt;i + +)
                    {
                    putchar(c);
                    }
               state = 0;
               break;
               }
          }
     exit(0);
     }
```

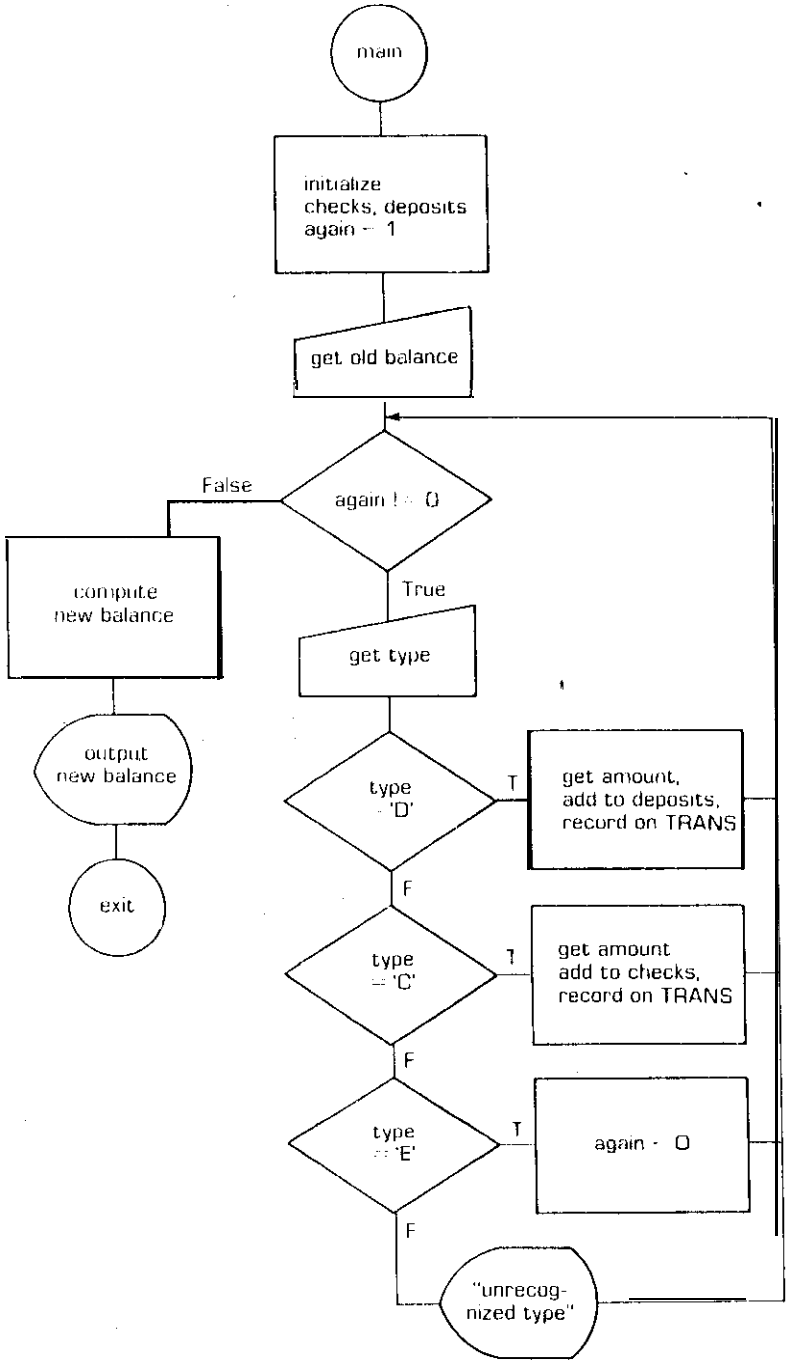
Flowchart A.2 DECOMPRESSION




repcnt ← c-OFFSET
state ← 2
True
state = 1
False
False
state = 2
True
i ← 0
i < repcnt
True
output C
i ← i + 1
state ← 0
False
Exit

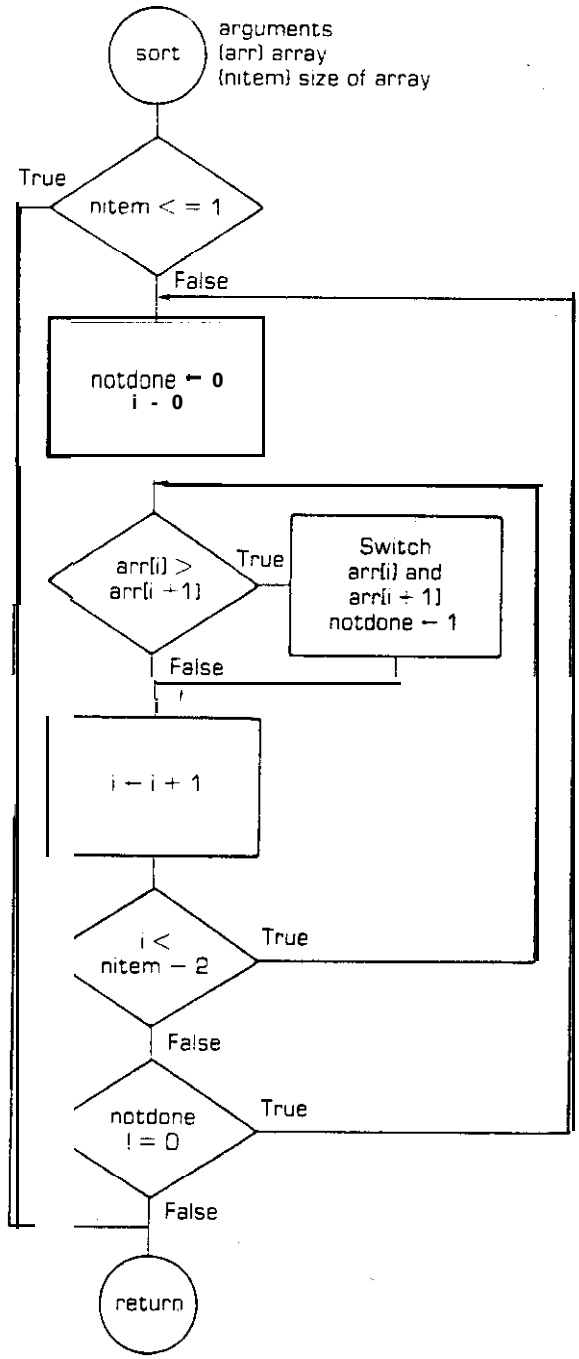
2. checkbook

โปรแกรมนี้จะรับเช็คและเงินฝากผ่านเข้าทางจอภาพแล้วเก็บข้อมูลไว้ในไฟล์และแจ้งผลออกมาเป็นคุลย์ใหม่ ไฟล์ชื่อ TRANSFIL เป็นไฟล์ที่สร้างขึ้น เพื่อรับเรคคอร์ดเกี่ยวกับเช็คและเงินฝากซึ่งอาจนำไปใช้ในโปรแกรมได้อีก

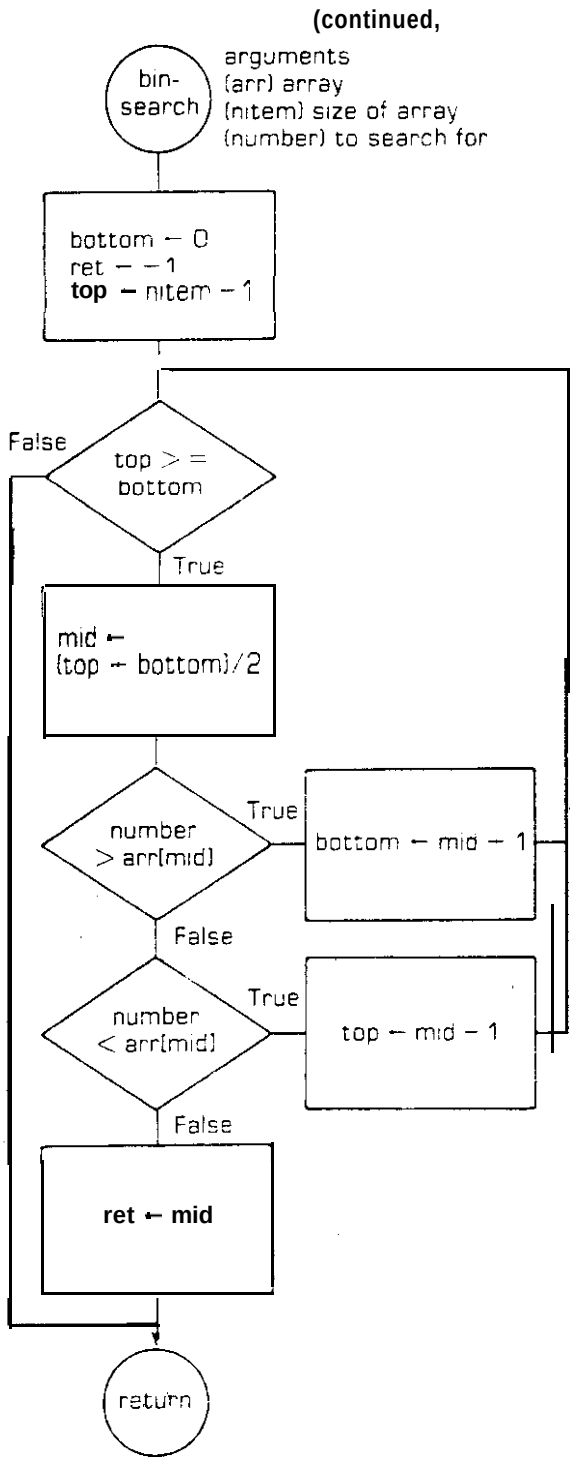


3. sort หรือ binary search

โปรแกรมต่อไปนี้แสดงการ sortตามวิธีbubble sortรวมทั้งการ search ตามวิธี binary search โดยจะเริ่ม sort ข้อมูลในลักษณะที่เข้า/ออกแบบอะเรย์แล้ว ถามว่าต้องการ search สิ่งใด ในที่นี้ใช้ 0 เป็นสัญญาณแสดงข้อมูลนำเข้าหน่วยสุดท้าย



(continued,


bin-
search
arguments
(arr) array
(nitem) size of array
(number) to search for
bottom ← 0
ret ← −1
top ← nitem − 1
False
top >=
bottom
True
mid ←
(top + bottom)/2
number
> arr[mid]
True
bottom ← mid − 1
False
number
< arr[mid]
True
top ← mid − 1
False
ret ← mid
return

```
#define MAXNUM 100
main( )
/* inputs, sorts, and searchs a string of integers */
        {
        int arr[MAXNUM];
        int size = 0;
        int ret, num, fnd;
        while (1)
              {
              /* input numbers till 0 or an error in scanf */
              printf("\ninput a number, 0 to start sort ");
              ret = scanf("%d",&num);
              if ( (num! = 0) && (ret= = 1) )
                  {
                  if (size<MAXNUM) arr[size++] = num;
                  else
                       {
                       printf("\nNo more room, starting sorting");
                       break:
                       }
                  }
              else break:
              } /* end while */
        printf("\nsize of array is %d",size);
        if (size = = 0)
              {
              printf("\nno input");
              }
        else
              {
              printf("\nUnsorted array");

              prtarr(arr,size);
              sort(arr,size);
              printf("\nSorted array");
              prtarr(arr,size);
              /* ask for input numbers */
              while (1)
                  {
                  printf("\nNumber to look for (0 to end) ");
                  ret = scanf("%d",&num);
                  if ((ret= = 1)&&(num! = 0))
                       {
                       fnd = binsrch(arr,size,num);
                       if (fnd> =0)
                            {
                            printf("\n%d is in array",num);
                            printf("\nindex is %d",fnd);
                            }
                       else printf("\n%d is not in array",num);
                       }
                  else break;
                  ) /* end while */
              } /* end else */
end:    exit(0);
        }
```

```
prtarr(arr,nitem)
/* prints nitems in integer array arr */
int arr[ ];
int nitem;
        {
        register int i;
        for (i = 0;i<nitem;i + +)
            {
            it (i%10 = = 0) printf("\n");
            printf(" %d ",arr[i]);
            t
        return;
        }
sort(arr,nitem)
/* sorts nitem in integer array arr */
int arr[ ];
int niiem;
        {
        register int i,notdone,temp;
        it (nitem< = 1) goto end;
        do
            {
            notdone=0;
            i = 0;
            do
                {
                if (arr[i]>arr[i + 1])
                    {
```

```
                    temp = arr[i];
                    arr[i] = arr[i + 1];
                    arr[i + 1] = temp;
                    notdone = 1;
                    }
                i + + ;
                }
                while (i<(nitem - 2));
            }
            while (notdone);
end:    return;
        }
binsrch(arr,nitem,number)
int arr[ ];
int nitem;
int number;

        int mid;
        in! bottom = 0;
        int top;
        int ret= -1;
        top = nitem - 1;
        while (top> = bottom)

            mid = (top + bottom) 2;
            if (number>arr[mid]) bottom = mid + 1;
            else if (number<arr[mid]) top=mid-1;
            /* found it */
            else {
                ret = mid;
                break;


        return ret;
```

4. linked list

โปรแกรมนี้ใช้สำหรับเชื่อมโยง list of record เข้าด้วยกันโดยใช้แอดเดรสของเรคคอร์ดเป็นเครื่องมือในการเชื่อมโยง เป้าหมายของ linked list มักมุ่งหมายไปที่การตามหา (search) เรคคอร์ดที่สนใจ

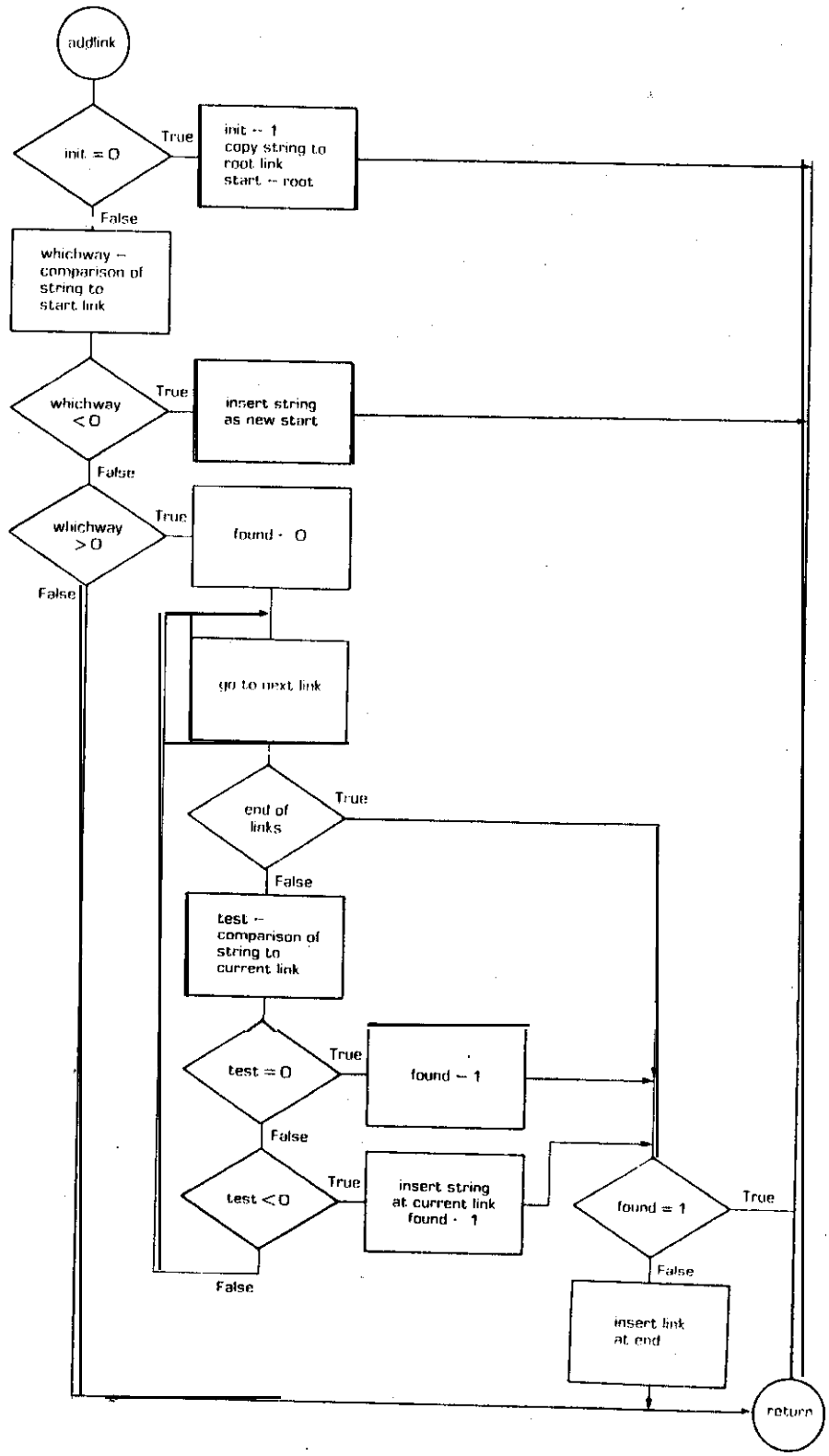
ตัวดำเนินการที่ใช้มากในโปรแกรมนี้คือ - > เราเรียกตัวดำเนินการนี้ว่า structure operator ตัวดำเนินการนี้ใช้กับ pointer ที่ชี้ไปที่ตัวแปรโครงสร้างเพื่ออ้างอิงสมาชิกตัวใดตัวหนึ่งของโครงสร้างมาใช้ เช่น

```
struct terminal * max ;
{
     char c ;
     if (*so-far! = max- > lines - 2)
     return (0)
```

เป็นการแจ้งให้คอมไพเลอร์ทราบว่าให้ไปใช้โครงสร้างแบบ (tag type) terminal ที่เคยนิยามไว้แล้วโดยตัวแปรชื่อ max เป็น pointer ที่ชี้ไปที่โครงสร้างนั้น

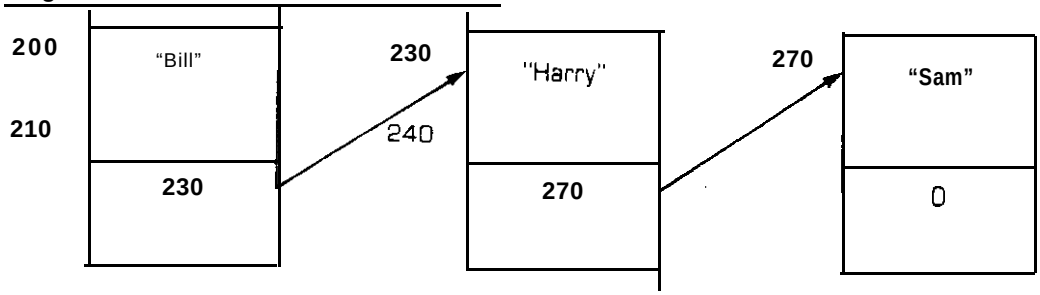
```
if (* so-far! = max -> lines - 2)
```

max - > line - 2 เป็นการสั่งให้ pointer ชื่อ max ที่ชี้ไปที่ตัวแปรโครงสร้างตามไปเพื่อเข้าถึงสมาชิกของโครงสร้างตัวหนึ่งสมาชิกตัวนั้นคือ lines


addlink
init = 0
True
init ← 1
copy string to
root link
start ← root
False
whichway ←
comparison of
string to
start link
whichway
< 0
True
insert string
as new start
False
whichway
> 0
True
found ← 0
False
go to next link
end of
links
True
False
test ←
comparison of
string to
current link
test = 0
True
found ← 1
False
test < 0
True
insert string
at current link
found ← 1
found = 1
True
False
insert link
at end
return
False

```
                    temp = arr[i];
                    arr[i] = arr[i + 1];
                    arr[i + 1] = temp;
                    notdone = 1;
                    }
                i++;
                }
                while (i<(nitem - 2));
            }
            while   (notdone);
end:    return;
        }
binsrch(arr,nitem,number)
int arr[];
int nitem;
int number;
        {
        int mid;
        int bottom = 0;
        int top;
        int ret = -1;
        top = nitem - 1;
        while (top> = bottom)
            {
            mid = (top + bottom)/2;
            if (number>arr[mid]) bottom = mid + 1;
            else if (number<arr[mid]) top = mid - 1;
            /* found it */
            else {
                ret = mid;
                break;
                }
            }
        return ret;
        }
```

Diagram A.1 LINKED LIST EXAMPLE




```
        char string[10];
        struct link *next;
        };
    main( )
    /* This program adds links and prints a list of links */
        {
        int chr = 0;
        char str[81];
        while (1)
            {
            if ( chr! = ' ' && chr ! = '\n')
                {
                printf("\nType A to add a link");
                printf("\nType P to print links");
                printf("\nType E to end\n");
                }
            chr = getchar( );
            if (chr = =EOF) break;
            chr = toupper(chr);
            if (chr = = 'E') break;
            if (chr = ='A')
                {
                printf("\nType string to add: ");
                scanf("%80s",str);
                addlink(str);
                }
            else if (chr = = 'P')
                {
                printf("\nLinks are:");
                printf("\nCount is: %d \n",prtlink( ));
                }
            }
        exit(0);
        }
    struct link root = {NULL,NULL};
    struct link *start = &root;
    int init = 0;
```

```
addlink(instr)
/* adds a link to singly linked list */
char *instr;
    {
    int found,whichway,test;
    struct link *new,*prev,*lalloc( ),*current;
    if (init = = 0)
       {
       /* this is the first link */
       init = 1;
       strcpy(start->string,instr);

    else

       whichway = strcmp(instr,start->string);
       /* whichway is 0 if strings match */
       /* is <0 if instr comes before start */
       /* is >0 if instr comes after start */

       if (whichway<0)

          new= lalloc( ):
          strcpy(new->string,instr);
          new->next = start;
          start = new;


       else if (whichway>0)
          {
          current = start;
          found = 0;
          /* search list for where to put string */
          while (current->next! = NULL)
              {
              prev = current;
              current = prev->next;
              test = strcmp(instr,current->string);

              /* test = 0 duplicate */
              if (test = = 0)
                 {
                 found = 1:
                 break;


               * test < 0 string comes before current *
              else if (test<0)
                 {
                 found = 1;
                 new = lalloc( );
```

5. random file maintainance

โปรแกรมนี้ทำหน้าที่เก็บรักษาไฟล์ชื่อ city โดยยอมให้เราเพิ่มเรคคอร์ดเข้าไปได้หรือจะสั่งพิมพ์เฉพาะเรคคอร์ดก็ได้

```
FILE UPDATE
#include "stdio.h"
#define SIZENAME 20
#define SIZECITY 20
#define SIZESTATE 10
#define ESCAPE 27 /* Used as character to end input */
#define NAMEFILE "TESTFILE" I* Name of file for records */
/* this is the record layout */
struct srecord {
      char name[SIZENAME];    /* Name */
      char city[SIZECITY];    /* City */
      char state[SIZESTATE]; /* State */
      };
#define SIZEREC sizeof(struct srecord)
         /* Size of record in bytes */
struct srecord record;
struct sfldes {
      char *field;    /* Points to field of record */
      char *label;    /* Label for that field */
      char size;     /* Size in bytes' of field */
      ) fldes[ ]=
               {
               {record.name,"Name",SIZENAME},
               {record.city,"City",SIZECITY},
               {record.state,"State",SIZESTATE}
               };
#define NUMFLD  sizeof(fldes)/sizeof(struct sfldes)
         /* Number of fields in record */
```

```
                    strcpy(new->string,instr);
                    prev->next = new:
                    new- >next = current;
                    break:
                    }
                /* test > 0 keep going */
                } /* end while */
            if (!found)
            /* ran past last link without setting up new link */
                {
                new = lalloc();
                current -> next = new;
                new ->next = NULL;
                strcpy(new- >string,instr);
                }
            ) * end else on whichway */
        } /* end else on init *I
    return 0;
    }
prtlink( )
/* prints the links in order */
/* returns number of links */
    {
    struct link *current;
    int count;
    count = 0;
    if (init = = 0)
        {
        printf("\n no links ");

    else
        \
        current = start;
        do
            {
            printf("\nlink %d is %s", count, current- >string);
            current = current -> next;
            count + + ;
            }
        while (current! = NULL);
        }
    return count;
    }
struct link *lalloc( )
        {
        static struct link m[100];
        static int count = 0;
```

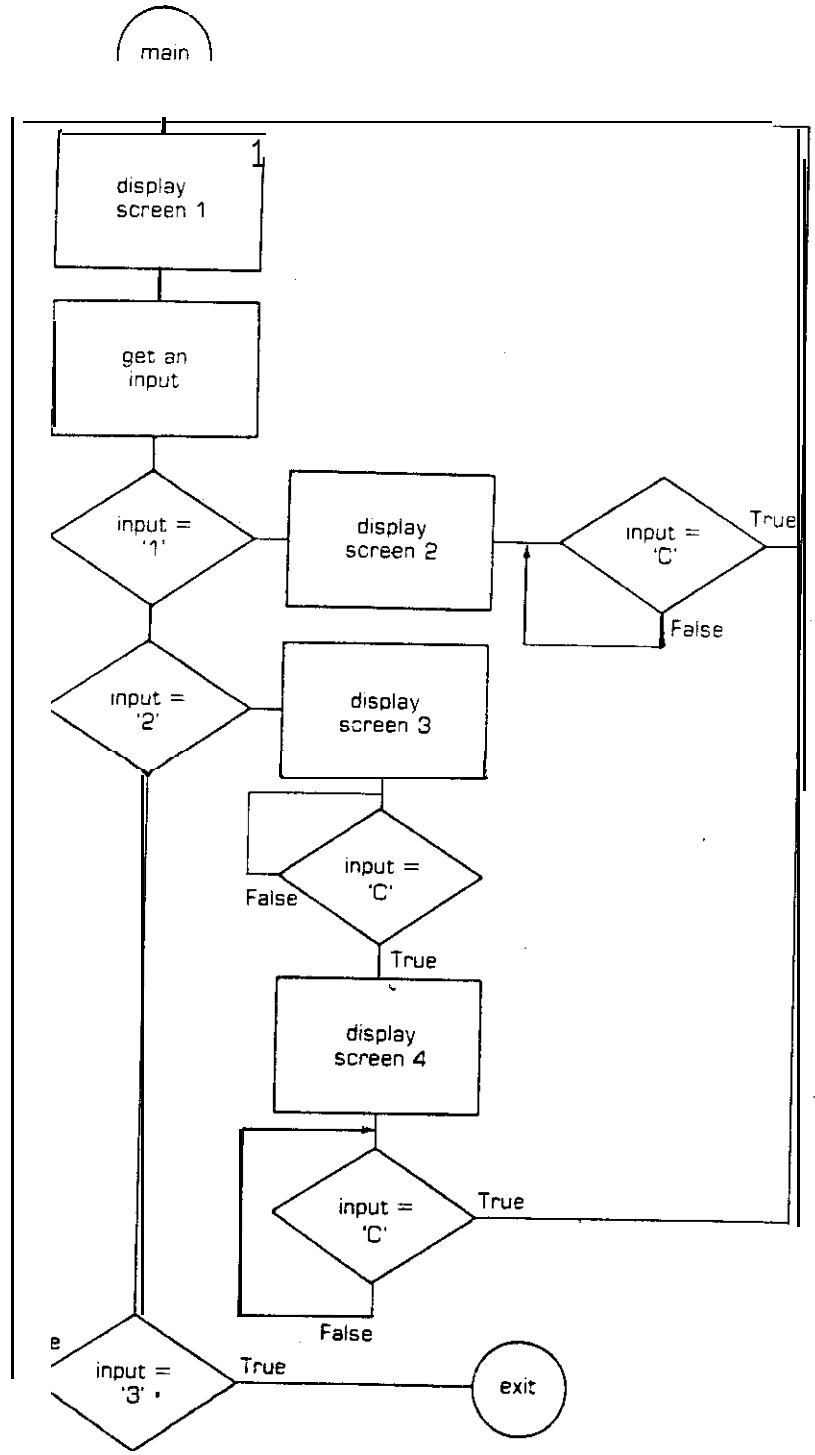
```
        if (ret<0) break:
        t
    return   ret;

getfld(buffer,n)
/* reads a field, returns -1 it ESC pressed else number of characters read */
char *buffer; /* Pointer to field */
int n;        /* size of field */
    {
    int c,ret;
    clear (buffer,n,0);
    ret = 0;
    while ( (n- -) && (c=getchar( ))!='\n' && c!=ESCAPE )
        {
        ret++:
        *bufter + + = c;
        t
    if (c= =ESCAPE) ret= -1;
    /* don't use rest of input till end of line */
    if (c!='\n')
        {
        while ( (c=getchar()) ! ='\n')
            {
            if (c= =ESCAPE) ret= -1;
            t
        t
    return   ret;
    t
prtall(filnam,reccnt,sizerec)
/* prints the records in filnam */
FILE *filnam; /* file pointer */
int reccnt:   /* number of records */
int sizerec;  /* size of record */
    {
    int i,j;
    printf("\nRecord count is %d\n",reccnt);
    for (i=1;i<=reccnt;i++)
        {
        rdrec(filnam,&record,i,sizerec);
        printf("\nRecord %d is:",i);
        for (j=0;j<NUMFLD;j++)
            {
            printf("\ n%s: ",fldes[j].label);
            prtfld(fldes[j].field,fldes[j].size);
            t
        t
    printf("\n");
```

```
        return sizerec;
        }
prtfld(field,size)
/* Prints 8 single field */
char *field;  /* Pointer to field */
int size;     /* Size of field */
        {
        while  (size- -)
            {
            putchar(*field + +);
            }
        return;
        }
```

6. screen menu display

โปรแกรมนี้ใช้แสดง main menu และ submenu รวม 2 เมนู



DISPLAY

```
#include "stdio.h"
#define ESCAPE 27
#define OFFSET 32
struct sline {
        int row;
        int col;
        char *string;
        };
static struct sline screen1[ ] = {
        {5,5,"Do you want to know about "},
        {7,10,"1. What this program does"],
        {9,10,"2. How this program works"),
        {11,10,"3. Exit this program"),
        {13,10."ENTER # AND RETURN"}
        };
Int sire1 = sizeof(screen1)/sizeof(struct sline);
static struct sline screen2[ ] ={
        {5,5,"It demonstrates menu selections"},
        {7,5,"Enter C to go back"},
        };
int size2 = sizeof(screen2)/sizeof(struct sline);
static struct sline screen3[ ] ={
        {5,5,"Enter C to go to next screen"),
        };
int size3 = sizeof(screen3)/sizeof(struct sline);
static struct sline screen4[ ] ={
```

```
        {5,5,"This program is written in C"},
        {7,5,"Enter C to go to main menu").
        };
int size4 = sizeof(screen4)/sizeof(struct sline);
main( )
        {
        int c;
        int d=0;
        while (1)
                {
                /* display main screen */
                if (d = = 0) dspscr(screen1,size1);
                cursor(20,20);
                c=getchar( );
                switch(c)
                        {
                        case '1':
                                dspscr(screen2,size2);
                                keyin('C');
                                d=0;
                                break;
                        case '2':
                                dspscr(screen3,size3);
                                keyin('C');
                                dspscr(screen4,size4);
                                keyin('C');
                                d=0;
                                break;
                        case '3':
                                exit(0);
                                break:
                        default:
                                d=1;
                                break;
                        }       /* end switch */
                }       /* end while */
        }       /* end main */

dspscr(screen,numline)
/* display a screen */
struct sline screen[ ]; /* array of lines */
int numline;                    /* number in array */
        {
        register int i:

        clrsc( );
        for (i=0;i<numline;i++)
                {
                dspline(&screen[i]);
```

```
        }
    return 0;
    }
/*spline(line)
 * prints line on terminal screen */
struct sline *line; /* line to print */
    {
    cursor(line->row,line->col);
    strout(line->string);
    return 0;
    }

cursor(row,col)
/* positions cursor at row,col
   no check made on validity of row and col
   This is terminal dependent          */
int row; /*  □□•  •  ☞
int col; /* column */
    {
    putchar(ESCAPE);
    putchar('=');
    putchar(OFFSET + row);
    putchar(OFFSET + col);
    return;
    }

strout(string)
/* outputs a string to terminal, not including null char •  /
/* no check is made for error on output */
char *string; /* string to output */
    {
    while (*string)
        {
        putchar(*string++);
        }
    return 0;
    }

clrsc( )
/* clears the screen */
/* this is terminal dependent */
    {
    putchar(ESCAPE);
    putchar('v');
    putchar(ESCAPE);
    putchar('T');
    putchar(ESCAPE);
```

```
        putchar('0');
        putchar('@');
        putchar(ESCAPE);
        putchar('9');
        putchar('P');
        return 0;
        }
keyin(chr)
/* waits till chr is pressed */
int chr; /* character to wait for */
        {
        while (1)
                {
                cursor(20.20);
                if (chr = = toupper(getchar( ))) break;
                }
        return chr;
        }
```

FUN WITH C ROUTINE

A program dedicated to the Beatles:

```
main( )
        {
while (9)
        {
        9;
        }
}
```

The usual program for demonstrating a C compiler is:

```
main( )
        {
        printf("\nhello world\n");
        }
```

An alternative is:

```
main( )
        {
        char *malloc( ) ,*b;
        printf("\ngoodbye world");
        b= malloc(1);
```

ภาคผนวก 3
C equivalent

ฟังก์ชันและคำสั่งต่าง ๆ ในภาษา C อาจเทียบได้กับคำสั่งในภาษาอื่น ๆ ดังนี้

APPENDIX C C EQUIVALENTS

BASIC	C Equivalen	Page
CALL assembly language routine	function	5
CLOSE	**fclose( )**	109
DATA—to set things to be READ (for initializing)	initializers	190
DEF—a function	function	43
DIM	[ ]	56
END	**exit( )**	4
FOR.TO.STEP/NEXT	**for**	33
GET	**getchar( )**	110
GOSUB	function	46
GOTO	**goto**	28
IF.THEN/ELSE	**if**	23
INPUT	**scanf( )**	109
LET(=)	=	17

BASIC	C Equivalent	Page
ON X GOTO	**switch**	39
ON X GOSUB	**switch**	39
OPEN	**fopen( )**	III
PEEK	pointers	74
POKE	pointers	74
PRINT	**printf( )**	107
PUT	**putchar( )**	105
READ (DATA)	**initializers**	190
REM	/* */	21
RETURN	**return**	190
STOP	**exit( )**	102
WRITE	**printf( )**	107
# (double)	**double**	8
$ (character)	**char**	8
% (integer)	**int**	8
Operators:		
+ - *	+ - * /	17
< > <= >= <>=	> < <= >= != = =	17
MOD	%	17
AND OR NOT	& | !	17

FORTRAN	C Equivalent	Page
CALL subroutine	function( )	46
COMMON	externals	51
COMPLEX	not supported	
CONTINUE	see DO or GOT0	
DATA-initialization	initializers	
DECODE	**sscanf( )**	109
DIMENSION	**[ ]**	56
DO	*for*	20
ELSE	*else*	24
ELSE IF	**else if**	24
ENTRY	not supported	
EQUIVALENCE	union	67
FORMAT	format string	107
FUNCTION	function	43
GO TO	**goto**	41
IF	if	23
IMPLICIT	not supported	
INCLUDE	#include	94

FORTRAN	C Equivalent	Page
INTEGER	**int**	8
LOGICAL	not necessary	
PAUSE	**printf( ) & getchar( )**	107
PRINT	**printf( )**	107
PROGRAM	**main( )**	49
READ	**scanf( )**	109
REAL	**float**	8
RETURN	return	42
REWIND	**fseek(,**	173
STOP	**exit( )**	102
SUBROUTINE	function	46
WRITE	**printf( )**	107
Statement function	function	46
Operators:		
EQ NE XOR OR AND NO?	**= = != && || !**	17
GTLTGELENEEQ	**> < >= <= != = =**	17
+ - *	+ - * /	17

PASCAL	C Equivalent	Page
ARRAY	[ ]	56
BEGIN	{	20
BOOLEAN ,	not needed	
CASE	switch	39
CHAR	**char**	8
CONST	**#define**	92
DO	see corresponding keyword	
DOWNTO	for	33
ELSE	**else**	24
END	see corresponding keyword	
FILE	file pointer	82
FOR	**for**	33
FUNCTION	function	43
GOTO	**goto**	28
IF	if	23
IN	enumeration	102
INTEGER	int	8
LABEL		17
MOD	**%**	17
NIL	NULL	75
NOT		17

PASCAL	C Equivalent	Page
OF	arrays of structures	56
OR	||	17
PACKED	not supported	
PROCEDURE	function	46
PROGRAM	**main( )**	49
READ	**scanf(,**	109
REAL	**float**	8
RECORD	**struct**	74
REPEAT	do-while	31
SET	enumeration	102
TEXT	**char[ ]**	8
TO	for	33
TYPE	**typedef**	60
UNTIL	do-while	31
VAR	declaration	56
WHILE	while	29
WITH	not supported	
WRITE	**printf( )**	107
:=	=	17
< <= = <> >= >	< <= == != >= ,	17
+ - * /	+ - * /	17
AND.OR.NOT	&&. || .!	17
MOD	%	17
DIV	/ (with integers)	17

PL/I*	C Equivalent	Page
ADDR	&	177
ALLOCATE	**malloc**	7:
BASED	pointers	74
BEGIN	{	20
CALL	function	46
CHARACTER	char	8
CLOSE	**fclose( )**	5
DECLARE	declarations	51
DECLARE 1.	structures	60
DEFINED	**union**	67
DO	{	20
DO variable =	for	33
Do WHILE	while	29

*Note that not all PL/I keywords are listed.

PL/I	C Equivalent	Page
Do UNTIL	do-while	31
END	}	20
EXTERNAL	externals	51
FIXED BINARY	int	8
FLOAT	float	8
GET EDIT. GET LIST	scanf( )	109
GO TO	goto	41
IF	if	23
INITIAL	initializers	62
OPEN	fopen( )	111
POINTER	pointers	74
PROCEDURE	functions	43
PROCEDURE OPTIONS(MAIN)	main( )	49
PUT EDIT, PUT LIST	printf( )	107
PUT SKIP.	printf("\n\n. . .)	107
RECORD	structures	74
R E T U R N	return	46
SELECT	witch	39
STATIC	static	62
WHILE	while	29
operators		
+ − * /	+ − * /	17
MOD	%	17
< > <= >= =	< > <= >= ==	17
& |	& |	17
	&& II	17
1	'	17

COBOL*	C Equivalent	Page
ACCEPT	getchar( ), scanf( )	107
AT END	EOF'	105
CALL	function	46
CLOSE	fclose( )	5
COMPUTE	=	29
DEPENDING ON	switch	39
ELSE	else	25
END	exit( )	102
FILE	file pointer	5
FOR	for	33
GOT0	goto	41

*Note that not all COBOL Keywords are listed

COBOL*	C Equivalent	Page
IF	**if**	23
INDEXED	**[ ]**	55
INITIAL	initializers	62
OCCURS	[ ]	55
OPEN	**fopen( )**	5
READ	**read( )**	5
REDEFINES	**union**	67
REWIND	**fseek( )**	173
SPACE		108
PICTURE 999V999	**float**	8
PICTURE 999	**int**	8
PICTURE XXXX	**char [ ]**	8
PICTURE	format control	8
PERFORM VARYING	**for**	20
PERFORM UNTIL	**while**	20
RECORD	**struct**	59
STOP RUN	**exit( )**	5
UNTIL	**while**	29
WRITE	**printf( )**	107
Operators:		
LESS THAN (<)	<	17
GREATER THAN (>)	>	17
EQUAL TO (=)	==	17
AND OR NOT	**&& || !**	17
+ - * /	+ - * /	17

ภาคผนวก 4
I/O routine

ฟังก์ชันและคำสั่งต่อไปนี้อาจมีใช้ในคอมไพเลอร์ของท่านหรือไม่ก็ได้ขอให้ตรวจสอบคู่มือคอมไพเลอร์ของท่านเสียก่อน คำสั่งและฟังก์ชันในที่นี้แยกเป็น 2 พวกคือ buffered roution หรือ high-level routine กับ unbuffered roution หรือ low-level routine ขอให้สังเกตรูปไวยากรณ์ให้ถี่ถ้วน ส่วนวิธีใช้หรือตัวอย่างการนำไปใช้ได้เคยแสดงไว้แล้วในบทที่ 9-10 ขอให้ศึกษาเพิ่มเติมจากตัวอย่างโปรแกรมในภาคผนวกที่ 2 ด้วย

ภาคผนวก 5
คอมไพเลอร์ภาษา C

ต่อไปนี้เป็นรายชื่อบริษัทผู้จำหน่ายคอมไพเลอร์ภาษา C ซึ่งจำหน่วยคอมไพ-เลอร์ที่มุ่งใช้กับ CPU บางรุ่นบางแบบ โดยที่ทั่วไปมุ่งสร้างเพื่อใช้กับไมโครโปรเซสเซอร์หมายเลข 8080, 8088 หรือ Z80 และ CP/M คำว่า small C หมายถึงคอมไพ-เลอร์ที่อาจไม่รับตัวดำเนินการบางตัวหรือข้อมูล (data type) บางแบบ โดยมากจะยอมรับ char และ int แต่อาจไม่รับ float และ double

"**r**" for reading only
"**w**" for writing only—this discards any data if the file already exists
"**a**" for appending to the end of an existing file

The return value is a file pointer for the file or **NULL** if the file cannot be opened. The file pointer is used in all other operations with this file. Since files can only be opened for reading or writing, many compilers include additional modes that allow simultaneous reading and writing, such as might be performed on a random access file.

```
FILE *freopen(name,mode,filepnt)
char *name;
char *mode:
FILE *filepnt;
```

Attaches a new file to previously opened file pointed at by **filepnt**. The old file is closed. The **name** and **mode** are the same as **fopen**. The return value is also the same. This function is usually used to reassign **stdin**, **stdout**, or **stderr**, because these files are automatically opened when a program is executed.

```
fread(buffer,itmsize,numitm,filepnt)
char *buffer;
int itmsize;
int numitm;
FILE *filepnt;
```

This reads **numitm** items of size **itmsize** from the file pointed at by **filepnt** and puts them starting at the address pointed to by **buffer**. The return value is the number of items read. This value may be less than **numitm** if an error or end of file occurred.

```
fwrite(buffer,itmsize,numitm,filepnt)
char *buffer:
int itmsize;
int numitm;
FILE *filepnt;
```

This writes **numitm** items of size **itmsize** to the file pointed at by **filepnt** from the address pointed to by **buffer**. The return value is the number of items written. This value may be less than **numitm** if an error occurred.

```
fclose(filepnt)
FILE *filepnt;
```

This closes the file pointed to by **filepnt**. It flushes any data in the buffer and closes the file. The return value is **EOF** if there was an error, or 0 otherwise.

Format	Corresponding pointer	Notes
d	integer	
o	integer	input may or may not have leading 0
x	integer	input may or may not have leading 0x
u	integer	unsigned integer
h	short integer	
c	character	skip over white space is not done. Next input character is returned. If next non-white character is wanted, use %1s.
s	character	pointer should be to variable array big enough to hold the string plus null character that is added.
f or **e**	float	input may have optional sign, a optional decimal point, and optional exponent field starting with E or e followed by signed or unsigned integer.
l	long integer	
lx	long integer	hexidecimal input
lo	long integer	octal input
lf or **le**	long float	double precision input

Input for numbers stops at the first nonvalid character if the field length has not been reached.

Note that the arguments to the scan routines must be pointers. A field is a string of non-white space characters. It extends either to the next white space character or until the field width, if any is specified, is completed. White space is blanks, tabs, newlines, and comments.

There are potential problems with **scanf**. If an error occurs in the input string, the scanning stops there. If **scanf** is immediately called again, the scanning will restart there. For example, if:

```
scanf("%d",&intvar)
```

and the input typed in from **stdin** is:

```
ABC
```

then **scanf** will return a 0. If it is called again with the same format, it will continue to return a 0.

FORMATTED OUTPUT

printf(format, *arguments...*)
char *format;
----- *arguments* (various types)

This prints values of *arguments* on **stdout** according to the **format** control string. The return value is the number of characters written or **EOF** if an error occurred.

fprintf(filepnt,format, *arguments...*)
FILE *filepnt;
charc*format;
----- *arguments*

This prints values of *arguments* on the file pointed to by **filepnt** according to the **format** control string. The return **value is** the **same as printf.**

sprintf(string,format, *arguments...)*
char *string;
char *format;
----- *arguments*

This puts the values of *arguments* into the string pointed at by **string** according to the **format** control string. The return **value is** the same **as printf.**

Formatting of output

The format specifiers for output are of the type:

$$\%-w.plx$$

- (optional) indicates the field is left justified

w is the minimum field width. More will be used if necessary.

separates the numbers *w* and *f* if *p* is used

p is the maximum field width for a string or the precision for a floating point number (number of digits after the decimal point)

I indicates the corresponding argument is a long type

x is one of the following:
d-decimal signed integer
u-decimal unsigned integer
x—hexidecimal integer (no leading 0x)
o-octal integer (no leading 0)
s-character string
c-single character
f-fixed decimal floating point number
e-exponential floating point number
g--either **f** or **e**, whichever is shorter

Characters that are not in format specifiers are output directly. If the character following a % is nc format specifier. then it is printed.

If the converted item has fewer characters than the minimum field length, then:

If **-** was present, left justify (pad on right)

Otherwise right justify (pad on left)

If field length *w* began with a 0, then use 0 to pad, otherwise use space (' ').

Format Specifier	Argument	Notes
s	string pointer	printed until null or maximum field width (*w*)

Format Specifier	Argument	Notes
e	float or double	output is of form: (−)x.xxxxxxE(+/−)xx where the precision (*p*) specifies the number of digits after the decimal point. Default precision is 6.
f	float or double	output is of form: (−)xxx.xxxxxx where the precision (*p*) specifies the number of digits after the decimal point. Default is 6.
g	float or double	e or f is used, whichever is shorter.

BUFFERED FILE OPERATIONS

```
fseek(filepnt,offset,mode)
FILE *file;
long onset;
int  mode;
```

This positions the file pointed to by **filepnt** to a new position so the next character read from that file is from that position,

mode = 0 position to **offset** bytes from beginning of file
= 1 position to **offset** bytes from current position
= 2 position to **offset** bytes relative to end of file

The return value is EOF if error occurs. Example of calls are:

Function call	Result
fseek(filepnt,0L,0)	file positioned at beginning (equals a rewind)
fseek(filepnt,0L,2)	file positioned at the end (to append to it)
fseek(filepnt,5L,1)	file positioned 5 bytes past current position.

```
long ftell(filepnt)
FILE *filepnt;
```

The return value is the current position in the file referenced by **filepnt** or EOF if an error occurred.

```
rewind(filepnt)
FILE *filepnt;
```

This rewinds the file referenced hy a **filepnt**. (Performs an **fseek(filepnt,0L,0)**).

```
ferror(filepnt)
FILE *filepnt;
```

The return value is the file descriptor if successful, otherwise a negative value. This file descriptor is used in the other operations on this file.

```
creat(filename,mode)
char *file;
int mode;
```

This opens a file with the system name **filename** for writing. If the file exists, it is deleted. The **mode** bits are system dependent. The return value is the file descriptor if successful, otherwise a negative value.

```
unlink(filename)
char *filename;
```

This removes the **filename** file from the system. The return value is 0 if successful, otherwise a negative number.

```
read(filedes,buffer,count)
int filedes;
char *buffer;
int count;
```

This reads count bytes from the file referenced by filedes to the address starting at buffer. The return value is 0 if end-of-file occurred. less than 0 if an error occurred. or the number of bytes read.

```
write(filedes,buffer,count)
int filedes;
char *buffer;
int count;
```

This writes count number of bytes starting at buffer to the file referenced by filedes. The return value is less than 0 if an error occurred or the number of bytes written.

```
long lseek(filedes,offset,mode)
int filedes;
long offset
int mode;
```

This positions the file referenced by filedes to the position specified by offset and mode.

mode = 0 offset relative to beginning of file
= 1 offset relative to current file position
= 2 offset relative to end of file

The return value is less than 0 if an error occurred or the new file position. Examples of calling this function are:

Function call	Result
lseek(filedes,0L,0)	file positioned at beginning (equals a rewind)
lseek(filedes,0L,2)	file positioned at the end—(to append to it)
lseek(filedes,5L,1)	file positioned 5 bytes past current position.

```
close(filedes)
int filedes;
```

This closes the file referenced by **filedes**. This return value is less than 0 if an error occurred or 0 if successful.

```
exit(retcode)
int retcode;
```

This flushes and closes all open files. It returns to the operating system with a return code of **retcode**.

```
_exit(retcode)
int retcode;
```

This terminates the program immediately without closing or flushing files. It returns the value of **retcode** to the operating system.

The return value is non-zero if an error occurred while reading or writing the file pointed to by **filepnt**. Otherwise zero.

clearerr(filepnt) or clrerr(filepnt)
FILE ***filepnt;**

This resets the error indication on the **filepnt** file. Until this function file is called. once an error condition occurs on a file, **EOF** wilt he the returned value on all functions concerning it.

fileno(filepnt)
FILE ***filepnt;**

The return value is the file descriptor used by the system input/output routines associated with the file filepnt.

fflush(filepnt)
FILE *filepnt;

This forces the internal buffer of the filepnt file to he written; that is, it flushes the buffer. The return value is **EOF** if any errors occurred or 0 otherwise.

Exhibit H.I ASCII COOL

DECIMAL	OCTAL	HEXADECIMAL'	BINARY	CHARACTER	NOTE
0	000	00	0000000	NUL	null
1	001	01	0000001	SOH	
2	002	02	0000010	STX	
3	003	03	0000011	ETX	
4	004	04	0000100	EOT	
5	005	05	0000101	ENQ	
6	006	06	0000110	ACK	
7	007	07	0000111	BEL	Produces beep or bell
8	010	08	0001000	BS	on terminals
9	011	09	0001001	HT	Backspace (\b)
10	012	0A	0001010	LF	Horizontal tab (\t)
11	013	0B	0001011	VT	Line feed (\n)
12	014	0C	0001100	FF	Vertical tab
13	015	0D	0001101	CR	Form feed (\f)
14	016	0E	0001110	SO	Carriage return (\r)
15	017	0F	0001111	SI	
16	020	10	0010000	DLE	
17	021	11	0010001	DC1	
18	022	12	0010010	DC2	
19	023	13	0010011	DC3	
20	024	14	0010100	DC4	
21	025	15	0010101	NAK	
22	026	16	0010110	SYN	
23	027	17	0010111	ETB	
24	030	18	0011000	CAN	
25	031	19	0011001	EM	
26	032	1A	0011010	SUB	
27	033	1B	0011011	ESC	
28	034	1C	0011100	FS	Escape

Exhibit H.I ASCII CODE (Continued)

DECIMAL	OCTAL	HEXADECIMAL	BINARY	CHARACTER	NOTE
29	035	1D	0011101	GS	
30	036	1E	0011110	RS	
31	037	1F	0011111	VS	
32	040	20	0100000	SP	
33	041	21	0100001	!	Space
34	042	22	0100010	"	
35	043	23	0100011	#	
36	044	24	0100100	$	
37	045	25	0100101	%	
38	046	26	0100110	&	
39	047	27	0100111	'	
40	050	28	0101000	(	Single quote
41	051	29	0101001	)	
42	052	2A	0101010	*	
43	053	2B	0101011	+	
44	054	2C	0101100	,	
45	055	2D	0101101	-	Comma
46	056	2E	0101110	.	Hyphen
47	057	2F	0101111	/	Period
48	060	30	0110000	0	
49	061	31	0110001	1	
50	062	32	0110010	2	
51	063	33	0110011	3	
52	064	34	0110100	4	
53	065	35	0110101	5	
54	066	36	0110110	6	
55	067	37	0110111	7	
56	070	38	0111000	8	
57	071	39	0111001	9	
58	072	3A	0111010	:	
59	073	3B	0111011	;	Colon
60	074	3C	0111100	<	Semicolon
61	075	3D	0111101	=	
62	076	3E	0111110	>	
63	077	3F	0111111	?	
64	100	40	1000000	@	
65	101	41	1000001	A	
66	102	42	1000010	B	
67	103	43	1000011	C	
68	104	44	1000100	D	
69	105	45	1000101	E	
70	106	46	1000110	F	
71	107	47	1000111	G	

Exhibit H.1 ASCII CODE (Continued)

DECIMAL	OCTAL	HEXADECIMAL	BINARY	CHARACTER	NOTE
72	110	48	1001000	H	
73	111	49	1001001	I	
74	112	4A	1001010	J	
75	113	4B	1001011	K	
76	114	4C	1001100	L	
77	115	4D	1001101	M	
78	116	4E	1001110	N	
79	117	4F	1001111	O	
80	120	50	1010000	P	
81	121	51	1010001	Q	
82	122	52	1010010	R	
83	123	53	1010011	S	
84	124	54	1010100	T	
85	125	55	1010101	U	
86	126	56	1010110	V	
87	127	57	1010111	W	
88	130	58	1011000	X	
89	131	59	1011001	Y	
90	132	5A	1011010	Z	
91	133	5B	1011011	[	
92	134	5C	1011100	\	
93	135	5D	1011101	]	
94	136	5E	1011110	^	
95	137	5F	1011111	_	Underline
96	140	60	1100000	`	Back quote
97	141	61	1100001	a	
98	142	62	1100010	b	
99	143	63	1100011	c	
100	144	64	1100100	d	
101	145	65	1100101	e	
102	146	66	1100110	f	
103	147	67	1100111	g	
104	150	68	1101000	h	
105	151	69	1101001	i	
106	152	6A	1101010	j	
107	153	6B	1101011	k	
108	154	6C	1101100	l	
109	155	6D	1101101	m	
110	156	6E	1101110	n	
111	157	6F	1101111	o	
112	160	70	1110000	p	
113	161	71	1110001	q	
114	162	72	1110010	r	

Exhibit H.I ASCII CODE (Continued)

DECIMAL	OCTAL	HEXADECIMAL	BINARY	CHARACTER	NOTE
115	163	73	1110011	s	
116	164	74	1110100	t	
117	165	75	1110101	u	
118	166	76	1110110	v	
119	167	77	1110111	w	
120	170	78	1111000	x	
121	171	79	1111001	y	
122	172	7A	1111010	z	
123	173	7B	1111011	{	
124	174	7C	1111100	|	
125	175	7D	1111101	}	
126	176	7E	1111110	~	
127	177	7F	1111111	(DEL)	Rubout

พิมพ์ที่... สำนักพิมพ์มหาวิทยาลัยรามคำแหง
Ramkhamhaeng University Press.

ภาษาที่
IT25842189
49.00 B