第三章 C語言的輸入輸出及結構

- 執行語彙分析時，很重要的一個工作就是從輸入資料流每次讀取一個字元，然後對這些字元加於分析歸類成指定的符記（token）。分析歸類是語彙分析程式的主要工作，本章的重點在從輸入資料流讀取資料，將資料存入適當的結構，並將指定的資料輸出至檔案。
- 本章將說明在標準函式庫裡的某些輸入輸出函式。
- 函式 scanf() 及 gets() 從鍵盤輸入。
- 函式 printf() 及 puts() 輸出至螢幕。
- 函式 fscanf() 及 fgets() 從指定的檔案輸入。
- 函式 fprintf() 及 fputs() 輸出至指定的檔案。

C語言的結構

- 結構（structure）是指資料結構而言，對於結構化程式設計或物件導向程式設計都是很重要的，尤其對複雜的資料結構更提供特別的助益。
- C 語言的陣列雖然結合了多項資料，可是均為相同的資料型態，例如同為字元、或同為整數，***C語言的結構（struct）可將不同型態的資料結合起來，使用一個結構變數或指標稱呼該結構，方便資料的處理工作。***

3.1 輸出函式

- 輸出函式 printf() 有兩個很好的高階性能。第一，任何數目的引數都可列印出來。第二，輸出透過簡單的格式就可控制。
- **格式一**：
- **printf (字串) ;**
- **字串**：
- 雙引號及其內所含的字元稱為字串，如 “GCC”
- 為字串，其雙引號內含字元為 GCC 三個大寫
- 英文字母。
- **printf("請輸入一個整數：");**
- 在螢幕上顯示「請輸入一個整數：」字串。

- **格式二：**
- **printf (格式字串，變數);**
- **格式字串：**
- 格式字串較常用的有 "%d"、"%f"、"%lf"、"%c"、"%s" 等五種格式字串，其餘的格式請參閱附錄E printf() 函式。
-
- **整數格式以 "%d" 表示。**
- **浮點數格式以 "%f" 表示。**
- **倍精確浮點數格式以 "%lf" 表示。**
- **字元格式以 "%c" 表示。**
- **字串格式以 "%s" 表示。**
-
- 一個格式字串中可同時表示多種格式，例如 "%d %lf %s" 表示第一個變數使用 %d 整數格式，第二個變數使用 %lf 倍精確浮點數格式，第三個變數使用 %s 字串格式表示。

- **變數：**
-
- **變數指變數名稱而言。在 C 語言中所用到的變數都要事先宣 告，宣告為整數變數 int、倍精確浮點數變數 double、字元 變數 char、或字串變數 char s[80]，等等。**
- **如下例中 i 為整數變數、d 為倍精確浮點數變數、ch 為字元 變數、s 為字串變數。**
-
- int i; /*宣告i為整數變數*/
- float f; /*宣告f為浮點數變數*/
- double d; /*宣告d為倍精確浮點數變數*/
- char ch; /*宣告ch為字元變數*/
- char s[80]; /*宣告s為字串變數，共80個字元*/

- **如下例：**
-
- printf("%d", i);　　/*格式字串 "%d",變數 i*/
- printf("%f", f);　　/*格式字串 "%f",變數 f*/
- printf("%lf", d);　　/*格式字串 "%lf",變數 d*/
- printf("%c", ch);　　/*格式字串 "%c",變數 ch*/
- printf("%s", s);　　/*格式字串 "%s",變數 s*/
- printf("%d %lf₩n",i,d);　/*格式字串 "%d %lf₩n"*/
- /*變數 i 對應 %d 格式*/
- /*變數 d 對應 %lf 格式*/
- printf("變數i的值=%d",i); /*格式字串 "變數i的值=%d"*/
- /*變數 i 對應 %d 格式*/

表 3.1 printf() 函式常用的轉換字元

- **轉換字元　輸 出 格 式**
- --------　----------
- c　　**將整數轉換為字元，適用** char **型態**
- d, i　　**十進位整數，適用** int **型態**
- f　　**浮點數，適用** float **型態**
- lf　　**倍精確浮點數，適用** double **型態**
- s　　**字串指標，輸出字串，適用** char[] **型態**

- 【程式yeardays.c】
- /********** yeardays.c **********/
- #include <stdio.h>
- int main()
- {
- float days = 365.25;
- int months = 12;
- double daysPerMonth = days / months;
- char title[13] = "每月平均天數";
- printf("%s\n一年天數=%6.2f 月數=%d 每月平均天數=%lf\n",
- title, days, months, daysPerMonth);
- return 0;
- }

- 【程式outputfile.c】
- /********** outputfile.c **********/
- #include <stdio.h>
- #include <stdlib.h>
- int main(int argc, char *argv[])
- {
- FILE *f = fopen("randfile.txt", "w");
- int i;
- for (i=0; i<5; i++)
- fprintf(f, "%d %9d\n", i, rand());
- fclose(f);
- system("TYPE randfile.txt");
- return 0;
- }

3.2 輸入函式

- **輸入函式 scanf() 有兩個很好的高階性能。第一，可輸入至任何數目的引數。第二，輸入透過簡單的格式就可控制。**
- **scanf() 函式的格式：**
- ***scanf (格式字串，&變數)；***
- **格式字串：**
- **含有格式的字串，常用的五種格式如下，其餘請參閱「附錄 E scanf() 函式」。**
- **整數格式以 "%d" 表示**
- **浮點數格式以 "%f" 表示**
- **倍精確浮點數格式以 "%lf" 表示**
- **字元格式以 "%c" 表示**
- **字串格式以 "%s" 表示**
- **格式字串中可同時表示多種格式，例如 "%d %lf %s" 表示第一個變數使用 %d 整數格式，第二個變數使用 %lf 倍精確浮點數格式，第三個變數使用 %s 字串格式表示。**

- **&變數：**
-
- 變數指變數名稱而言。在 C 語言中所用到的變數都要事先宣告，宣告為整數變數 int、浮點數變數 float、倍精確浮點數變數 double、字元變數 char、或字串變數 char s[80]，等 等。
- 如下例中 i 為整數變數、f 為浮點數變數、d 為倍精確浮點 數變數，ch 為字元變數、s 為字串變數。變數前的「&」符號表示該變數的內含值為在電腦記憶體中的位址。
-
- int i; **/*宣告i為整數變數*/**
- float f; **/*宣告f為浮點數變數*/**
- double d; **/*宣告d為倍精確浮點數變數*/**
- char ch; **/*宣告ch為字元變數*/**
- char s[80]; **/*宣告s為字串變數*/**

- 如下例：
-
- scanf("%d", &i);　　/*格式字串 "%d",變數 i*/
- scanf("%f", &f);　　/*格式字串 "%f",變數 f*/
- scanf("%lf", &d);　　/*格式字串 "%lf",變數 d*/
- scanf("%c", &ch);　　/*格式字串 "%c",變數 ch*/
- scanf("%s", &d);　　/*格式字串 "%s",變數 d*/
- scanf("%d %lf", &i,&d); /*格式字串 "%d %lf",變數i及d*/
- 　　/*變數i對應%d,變數d對應%lf*/

表 3.2 scanf() 函式常用轉換字元

轉換字元	輸入型態	變數資料型態
c	字元	字元指標（char * ）
d	整數	整數指標（int * ）
f	浮點數	浮點數指標（float * ）
lf	倍精確浮點數	倍精確浮點數指標（double * ）
i	整數	整數指標（int * ）
s	字串	字串指標（char * ）

```
/******************** scandays.c ******************/
#include <stdio.h>
int main()
{
  float days;
  int months;
  double daysPerMonth;
  printf("請輸入一年天數: ");
  scanf("%f", &days);
  printf("請輸入月數: ");
  scanf("%d", &months);
  daysPerMonth = days / months;
  printf("一年天數=%6.2f 月數=%d 每月平均天數=%lf\n",
        days, months, daysPerMonth);
  return 0;
}
```

- 【程式getstr.c】
-
- /*************** getstr.c ************/
- #include <stdio.h>
- int main()
- {
- char s[80];
- printf("請輸入一個字串: ");
- ***gets(s);***
- printf("您剛輸入的值= %s \n", s);
- return 0;
- }
-

- 【程式danger.c】
-
- /*************** danger.c ****************/
- #include <stdio.h>
- int main()
- {
- char ok[3]="OK";
- char s[5];
- printf("請輸入一個字串s: ");
- ***gets(s);***
- printf("ok=\"%s\" s=\"%s\"\n", ok,s);
- return 0;
- }
-

- 【程式safe.c】
-
- ```
  /****************** safe.c ****************/
  #include <stdio.h>
  int main()
  {
    char ok[3]="OK";
    char s[5];
    printf("請輸入一個字串s: ");
    fgets(s, sizeof(s), stdin);
    printf("ok=\"%s\"  s=\"%s\"\n", ok,s);
    return 0;
  }
  ```
-
  ```

【程式getfile.c】

```
/*************** getfile.c **************/
#include <stdio.h>
int main()
{
  FILE *f=fopen("randfile.txt", "r");
  char buf[80];
  while ( fgets(buf,sizeof(buf),f) != NULL )
  {  printf("%s", buf);  }
  fclose(f);
  return 0;
}
```

【程式inputfile.c】

```
/*************** inputfile.c **************/
#include <stdio.h>
int main()
{
  FILE *f=fopen("randfile.txt", "r");
  int i, num;
  while ( fscanf(f, "%d %d", &i, &num) != EOF)
  { printf("i=%d  num=%d\n", i,num); }
  fclose(f);
  return 0;
}
```

3.3 函式 sprintf() 及 sscanf()

- 函式 sprintf() 及 sscanf() 為 printf() 及 scanf() 的字串版本。格式如下：
- ***sprintf(字串, 格式字串, 變數);***
- 將「變數」依「格式字串」編排後的結果存入「字串」變數裡頭。
- ***sscanf(字串, 格式字串, 變數);***
- 從「字串」依「格式字串」的格式讀入「變數」裡。

【程式sscan.c】

```
/******************** sscan.c ****************/
#include <stdio.h>
int main(void)
{
  char str[ ]="12 3.4";
  int i;
  double d;
  sscanf(str, "%d %lf", &i, &d);
  printf("整數i=%d 倍精確浮點數d=%lf\n", i,d);
  return 0;
}
```

【程式sprint.c】

```
/************ sprint.c ***********/
#include <stdio.h>
int main(void)
{
  char str[80];
  int i=98;
  double d=7.65;
  sprintf(str, "%d %lf", i, d);
  printf("str=\"%s\"\n", str);
  return 0;
}
```

3.4 函式 gets() 及 puts()

- 從鍵盤輸入一個字串時呼叫 scanf() 函式，使用「%s」格式輸入，那是一般所使用的方法，但受限於輸入的字串當中不能含有空白字元，若含有空白字元就得另想他法了。
- ***C 語言提供一個 gets(buffer) 方法，允許輸入的字串當中含有空白字元，它將空白字元當成資料的一部份，並將 Enter 鍵所產生的「\r\n」中的「\r」去掉，只含「\n」部份而已。***
- 呼叫 gets(buffer) 成功時傳回 buffer，失敗或檔尾傳回 NULL，但 gets() 隱藏危機，仍以 fgets() 函式取代為宜。

【程式gets.c】

```
/****************** gets.c **************/
#include <stdio.h>
int main(void)
{
  char buffer[100];
  printf("請輸入一字串: ");
  fgets(buffer, sizeof(buffer), stdin);
  printf("您剛輸入的字串= ");
  puts(buffer);
  return 0;
}
```

- 【程式scanfromstr.c】
- /*************** scanfromstr.c ************/
- #include <stdio.h>
- int main()
- {
- int a, b, sum;
- char buffer[80];
- printf("請輸入一個字串(1234 567): ");
- ***fgets(buffer, sizeof(buffer), stdin);***
- ***sscanf(buffer, "%4d%4d", &a, &b);***
- sum = a + b;
- printf("a=%d b=%d sum=%d\n", a,b,sum);
- return 0;
- }

【程式token.c】

```
/******************** token.c ******************/
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[])
{
  char buf[256], *token;
  printf("請逐次輸入字串，直接按Ctrl-Z鍵結束\n");
  while ( fgets(buf,sizeof(buf),stdin) != NULL )
  {
    token=strtok(buf, " \t\n");
    while (token != NULL)
    {
      puts(token);
      token=strtok(NULL, " \t\n");
    }
  }
  return 0;
}
```

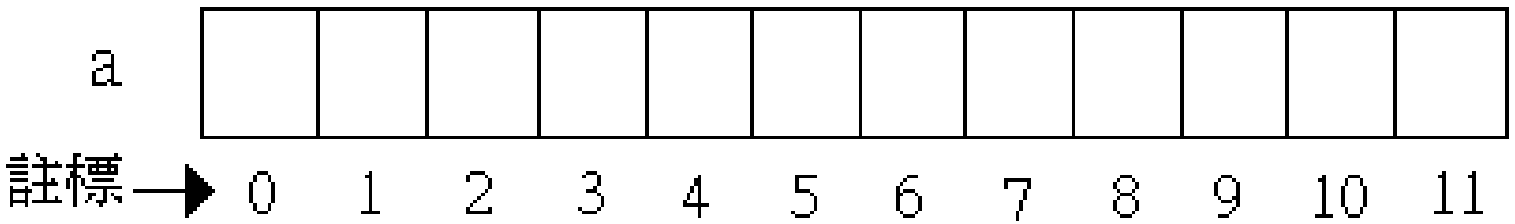
3.5 列舉型態

- 列舉型態 enum 是一種資料型態，它的整數值限於一個指定的常數集合。它的語法如下：
- ***enum 列舉識別字 { 指定的常數集合 } ;***
- 如下例：
- ***enum opinionTag { YES, NO, MAYBE } ;***
- 列舉識別字為 opinionTag，指定的常數集合為三個成員，YES 整數常數值為 0，NO 整數常數值為 1，MAYBE 整數常數值為 2。
- 您可以宣告列舉型態的變數 guess 如下：
- ***enum opinionTag guess;***
- ***...***
- ***guess = YES;***
- 設定 guess 的值為 YES 整數常數，相當於 1。

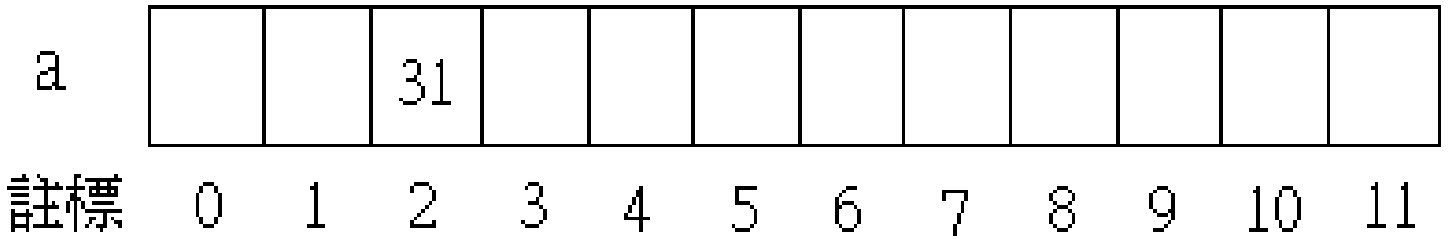
3.6 陣列

- 陣列（array）是由一群具有相同資料型態的元素所組成的，利用迴圈並配合陣列的註標值，可輕易的處理較大量的資料，它是一系列同一型態的資料項，可利用迴圈並配合陣列的索引值存取，所配置的是連續的記憶體。利用索引存取陣列裡的元素，這個索引稱為註標。C 語言裡的所有型態都可用來建立一個陣列。陣列裡的元素本身也可以是陣列，也就是陣列裡可以包含陣列。字串只是字元的陣列而已。
- 典型的陣列宣告，所配置記憶體從一個基底位址開始，然後配置陣列元素所需的連續記憶體。陣列名稱事實上就是這個基底位址的指標常數。

- **陣列宣告**說明陣列的資料型態、變數名稱、元素個數等等。**方括號"[ ]" 為陣列符號，每一組方括號表示一維，方括號**內可選用「常數運算式」，其值為常數，用於表示陣列該維的元素個數。陣列之元素儲存於記憶體的連續位址，其元素個數由 0 數起。
- 如下例：
- int a[12];
- 表示陣列的名稱為 a，共有 12 個元素，每個元素均為整數，其佔用記憶體的排列如下：



- 要指定陣列中的元素可使用如下的格式：
- 陣列名稱 [註標]
- 如下例：
- ***a[2]=31;　/*a陣列中第2個元素之值設為31*/***
- 將 a 陣列中第 2 個元素之值設為 31，請注意其元素之註標從 0 算起。如下圖所示。



【程式wordstack.c】

```
/******************** wordstack.c ***************/
#include <stdio.h>
#include <stdlib.h>
#define WORDMAX 512
char words[WORDMAX][36];
int top=0;
void push(char word[])
{
  strcpy(words[top], word);
  top++;
}
char * pop()
{
  static char word[36];
  top--;
  strcpy(word, words[top]);
  return word;
}
```

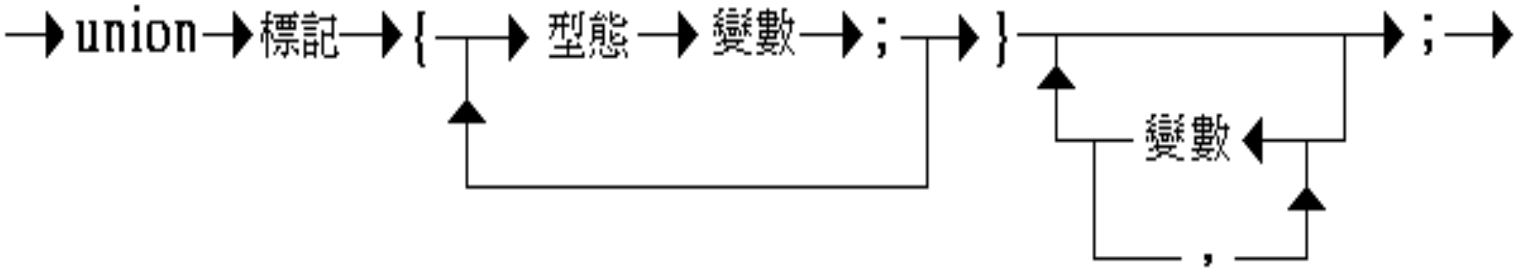
```
int isEmpty()
{
  if (top==0)
    return 1;
  else
    return 0;
}
int main()
{
  char word[36];
  printf("請逐次輸入字串疊入堆疊，Ctrl-Z結束\n");
  while (fgets(word,sizeof(word),stdin)!=NULL)
  {
    push(word);
  }
  printf("逐次從堆疊疊出後輸出\n");
  while (! isEmpty())
  {
    printf("%s", pop());
  }
  return 0;
}
```

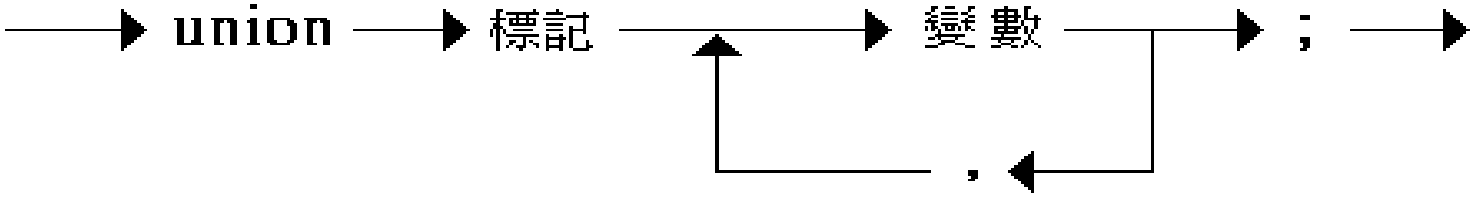
```
/****************** wordqueue.c ***************/
#include <stdio.h>
#include <stdlib.h>
#define WORDMAX 512
char words[WORDMAX][36];
int head=-1, tail=-1;
int isEmpty()
{
  if (head==-1 && tail==-1)
    return 1;
  else
    return 0;
}
void queuein(char word[])
{
  if (isEmpty())
  {
    strcpy(words[++tail], word);
    head = tail;
  }
  else
     strcpy(words[++tail], word);
 }
```

```
char * queueout()
{
  if (isEmpty() || head>tail)
    return NULL;
  else
    return words[head++];
}
int main()
{
  char word[36], *wp;
  printf("請逐次輸入字串佇入佇列，Ctrl-Z結束\n");
  while (fgets(word,sizeof(word),stdin)!=NULL)
  {
    queuein(word);
  }
  printf("逐次從佇列佇出後輸出\n");
  while ((wp=queueout())!=NULL)
  {
    printf("%s", wp);
  }
  return 0;
}
```

3.7 宣告同位

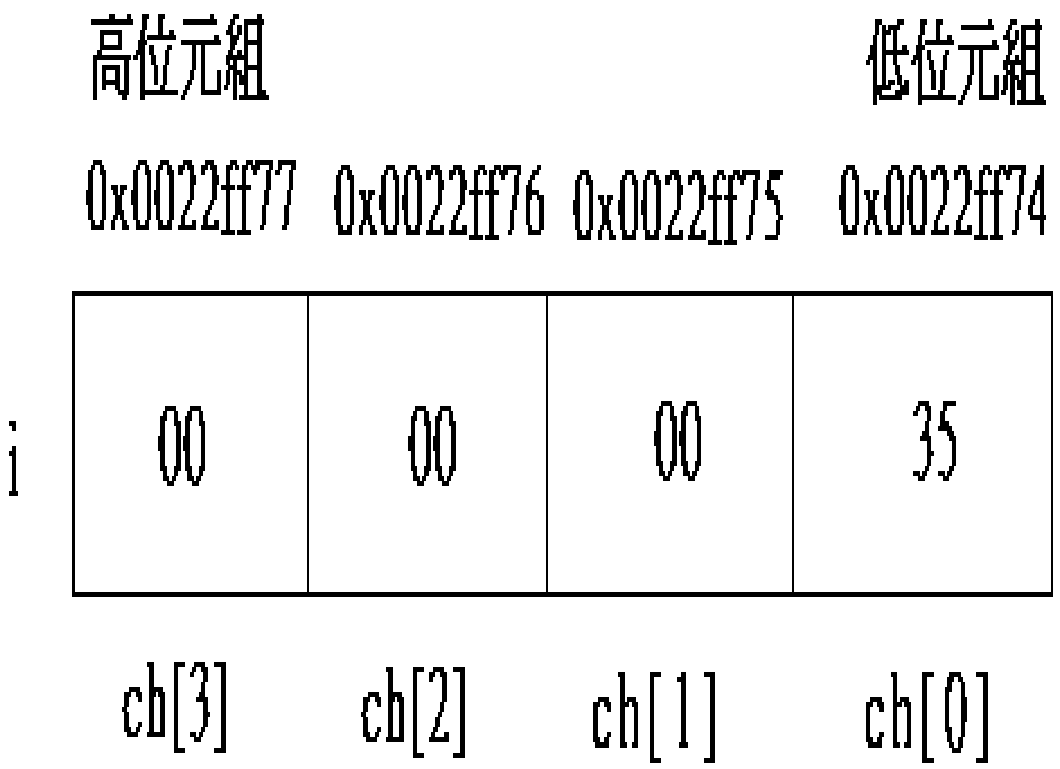
- 以關鍵字 union 宣告一個同位型態的結構，並以標記命名，也就是標記的名稱，其後以大括號括起來的是同位的成員，每一個成員皆為指定型態的變數，其語法如下圖所示。





同位變數在記憶體裡以同位成員中佔用最大記憶體的變數為準。若使用佔用較小記憶體的變數，則剩餘記憶體就浪費掉了。例如同位宣告之變數 u，其佔用記憶體情形如下圖所示。

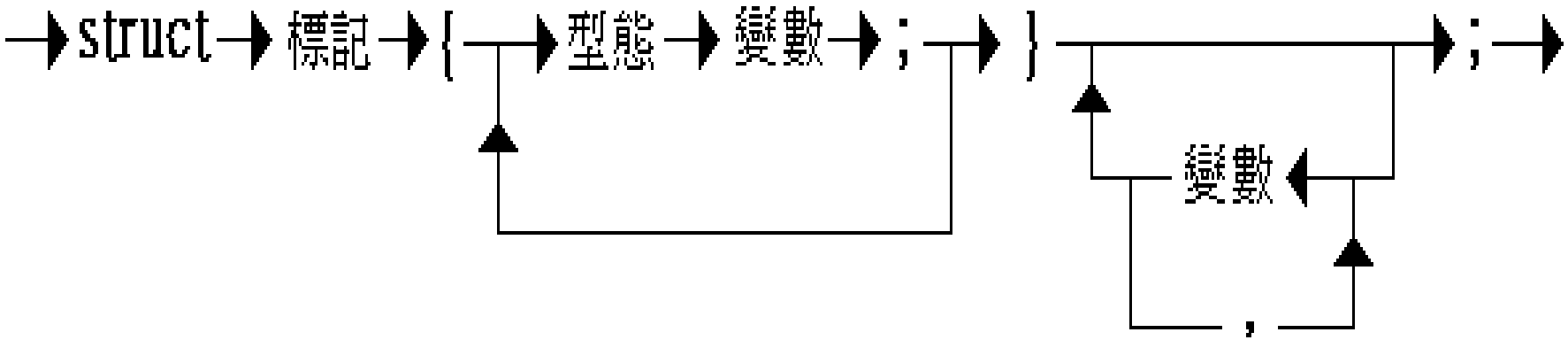
```
union utag
{
  int i;
  char ch[4];
} u;
```



- 同位變數 u 中含有二個成員：
-
- ***整數　　i　　佔四個位元組***
- ***字元陣列 ch[ ]　佔四個位元組***
-
- 但在記憶體中均使用同一塊記憶體。
- 　　u.ch[0]='5';
- 最低位元組內含值為字元 '5'，所構成的 i 整數值為 0x00000035，字元 ch[0] 內含值為 0x35。

3.8 宣告結構

- 結構的宣告以關鍵字 struct 開始，跟著為結構的標記名稱，然後以大括號括起來的是一個區塊（block），結構包含結構成員，每一個成員皆為指定型態的變數。　結構語法如下圖所示。



- 如下例，結構標記（tag）名為 wordTag，共有兩個結構變數成員，一個為字元型態 char 的陣列 word，最長 36 個字元，另一個為結構指標變數 next。
- ***struct wordTag***
- ***{***
- ***char word[36];***
- ***struct wordTag *next;***
- ***};***
- 宣告結構變數 wordnode 如下。
- ***struct wordTag wordnode;***
- 宣告結構指標變數 top 如下。
- ***struct wordTag *top=NULL;***

3.8 建立連結堆疊

- 陣列可以當堆疊及佇列使用，但宣告陣列變數時必須指明陣列的大小，其大小及所佔用的記憶體在宣告時就已決定了，屬於靜態的資料結構，但我們的許多應用程式要求用到時才配置記憶體給指定的結構或變數，必須屬於動態的才行。
- 我們先來看一看動態的堆疊如何實作。

```
int  isEmpty()
{
  if  ( top == NULL )
    return 1;
  else
    return 0;
}
```

- 這個函式 isEmpty() 判斷堆疊是否空白，頂端指標 top 的初始值為保留字 NULL，因此若 top 值為 NULL 即表示堆疊空白，傳回整數值 1，否則傳回 0 表示非空白。

```
void push(char *word)
{
  struct wordTag *p;
  p = malloc(sizeof(struct wordTag));
  strcpy(p->word, word);
  p->next = top;
  top = p;
}
```

- 這個函式 push() 將識別字 word 疊入堆疊頂端。首先取得一塊記憶體，命名為 p，將參數 word 拷貝至 p 所指的 word 欄，將 next 指到頂端，修訂新的頂端為 p。

```
struct wordTag * pop()
{
  struct wordTag *p;
  if  ( isEmpty() )
    return NULL;
  p = top;
  top = top->next;
  return p;
}
```

- 這個函式 pop() 傳回堆疊頂端元素的指標，並修訂新的頂端為它的下一個。堆疊空白時傳回 NULL 值。

- 【程式stack.c】

```
/********************* stack.c ******************/
#include <stdio.h>
#include <stdlib.h>
struct wordTag
{
  char word[36];
  struct wordTag *next;
};
struct wordTag *top=NULL;
int isEmpty()     { …}
void push(char *word)    { …}
struct wordTag * pop() { … }
int main()
{
  char word[36];
  printf("請逐次輸入字串疊入堆疊，  Ctrl-Z結束\n");
  while ( fgets(word,sizeof(word),stdin) != NULL ) { push(word); }
  printf("逐次從堆疊疊出後輸出\n");
  while ( ! isEmpty() ) {  printf("%s", pop());  }
  return 0;
}
```

3.9 建立連結佇列

- 堆疊的疊入及疊出都發生在頂端，屬於先進後出的操作，但佇列不同，它從後端佇入，從前端佇出，因此必需有前端指標及後端指標，屬於先進先出的操作。
- ```
  int  isEmpty()
  {
    if  ( head == NULL )
      return 1;
    else
      return 0;
  }
  ```
- 當前端指標 head 的值為 NULL 時表示佇列空白，傳回整數 1，否則傳回整數 0。
  ```

【程式queue.c】

```
/********************* queue.c ********************/
#include <stdio.h>
#include <stdlib.h>
struct wordTag
{
  char word[36];
  struct wordTag *next;
};
struct wordTag *head=NULL, *tail;
int isEmpty()
{
  if (head==NULL)
    return 1;
  else
    return 0;
}
```

```
void  queuein(char *word)
{
  struct wordTag *p;
  p = malloc(sizeof(struct wordTag));
  strcpy(p->word, word);
  if (isEmpty())
  {
    head = p;
    tail = p;
    p->next = NULL;
  }
  else
  {
    p->next = tail->next;
    tail->next = p;
  }
}
```

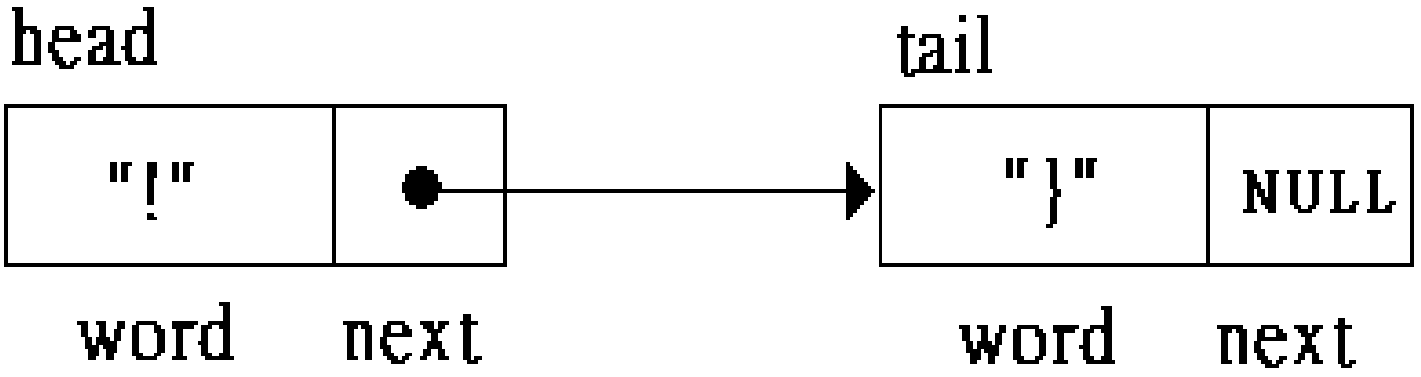
```
struct wordTag * queueout()
{
  struct wordTag *p;
  if (isEmpty())
    return NULL;
  p = head;
  head = head->next;
  return p;
}
```

```
int main()
{
  char word[36];
  struct wordTag *wp;
  printf("請逐次輸入字串佇入佇列，Ctrl-Z結束\n");
  while (fgets(word,sizeof(word),stdin)!=NULL)
  {
    queuein(word);
  }
  printf("逐次從佇列佇出後輸出\n");
  while ((wp=queueout())!=NULL)
  {
    printf("%s", wp->word);
  }
  return 0;
}
```

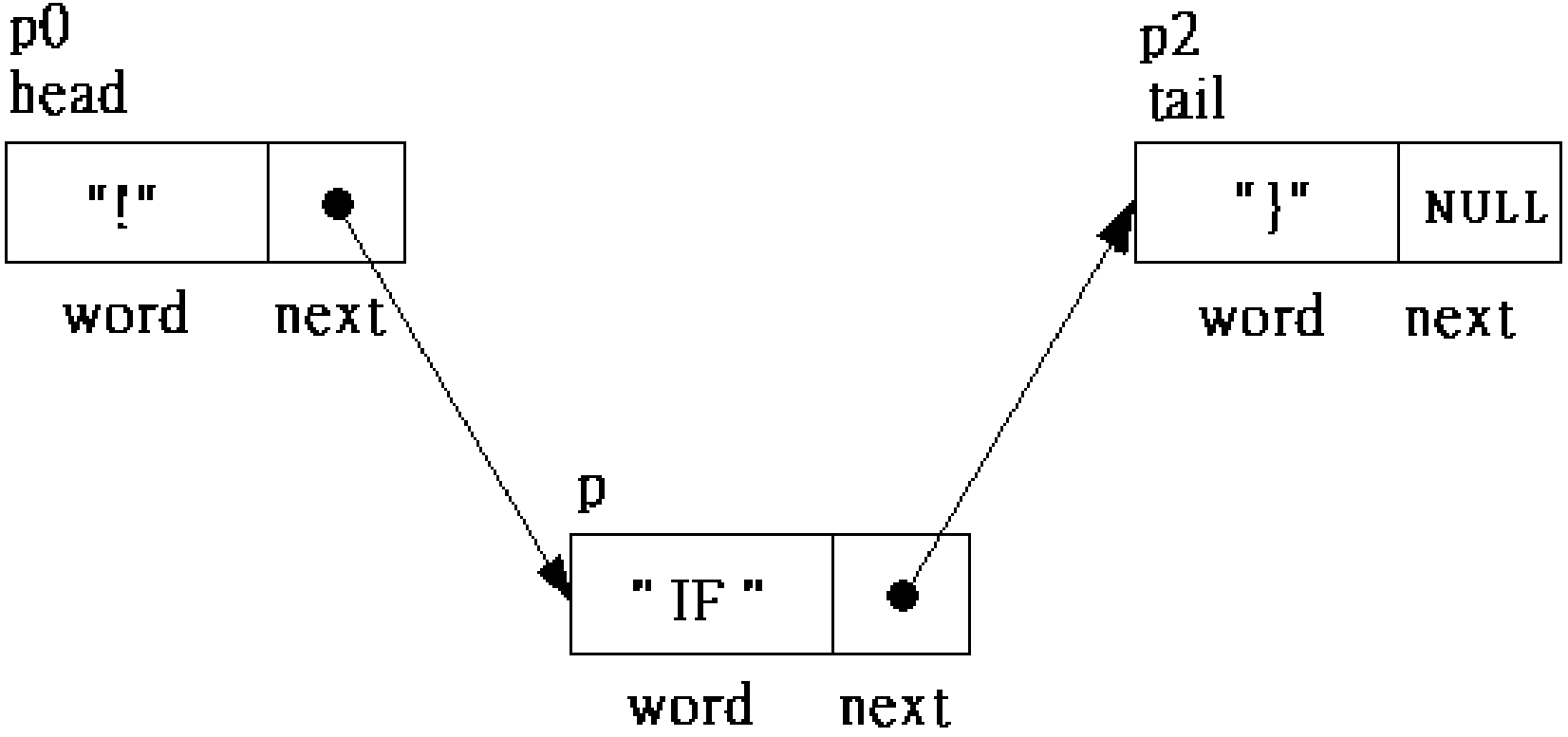
3.10 建立連結串列

- 堆疊及佇列其進出都有一定的規則可循，但連結串列的節點要在那一個位置插入卻要依題目的需求而定。例如連結串列裡存放的都是識別字字串，要依英文字母由小到大排列，那麼字母小的就要插入在前面，字母大的就要插入在後面。若要求改變了，要依英文字母由大到小排列，那麼字母小的就要插入在後面，字母大的反而要插入在前面了。

- 因為我們的識別字第一個字元為英文字母，我們事先建立一個前端節點 head，其識別字存放一個比英文字母小的字元，例如 '!'。事先建立一個後端節點 tail，其識別字存放一個比英文字母大的字元，例如 '}'。



- 如此我們所輸入的識別字會在這兩個節點間的適當位置依指定的規則插入，例如輸入識別字 "IF"，'I' 比 '!' 大，比 '}' 小，插入其間如下圖所示。



- 【程式list.c】
-
- /***************** list.c ***************/
- #include <stdio.h>
- #include <stdlib.h>
- struct wordTag
- {
- char word[36];
- struct wordTag *next;
- };
- struct wordTag *head, *tail;

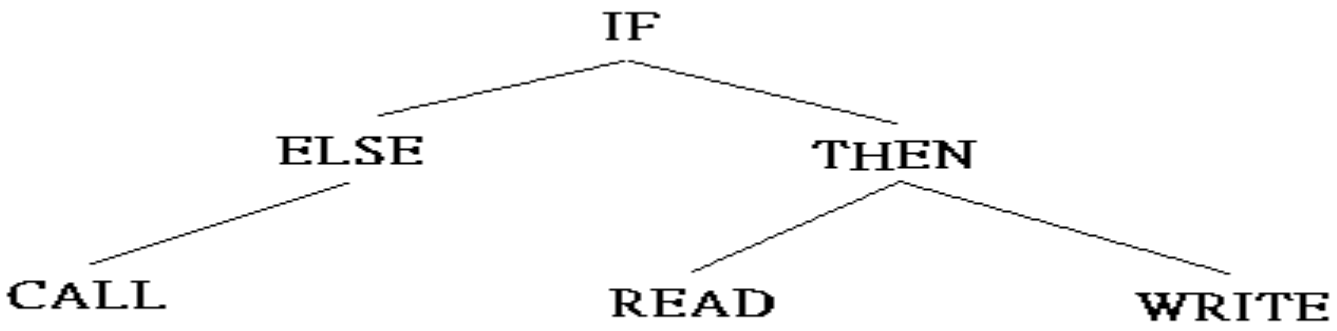
```
void insertNode(char *word)
{
  struct wordTag *p, *p0, *p2;
  p = malloc(sizeof(struct wordTag));
  strcpy(p->word, word);
  p0 = head;
  p2 = p0->next;
  while (1)
  {
    if (strcmp(p->word, p2->word)<0)
    {
      p->next = p2;
      p0->next = p;
      break;
    }
    else
    {
      p0 = p2;
      p2 = p2->next;
    }
  }
}
```

```
void listprint()
{
  struct wordTag *p=head->next;
  while (p!=NULL)
  {
    if (p==tail) break;
    printf("p=0x%p next=0x%p word=\"%s\"\n",
          p, p->next, p->word);
    p = p->next;
  }
}
```

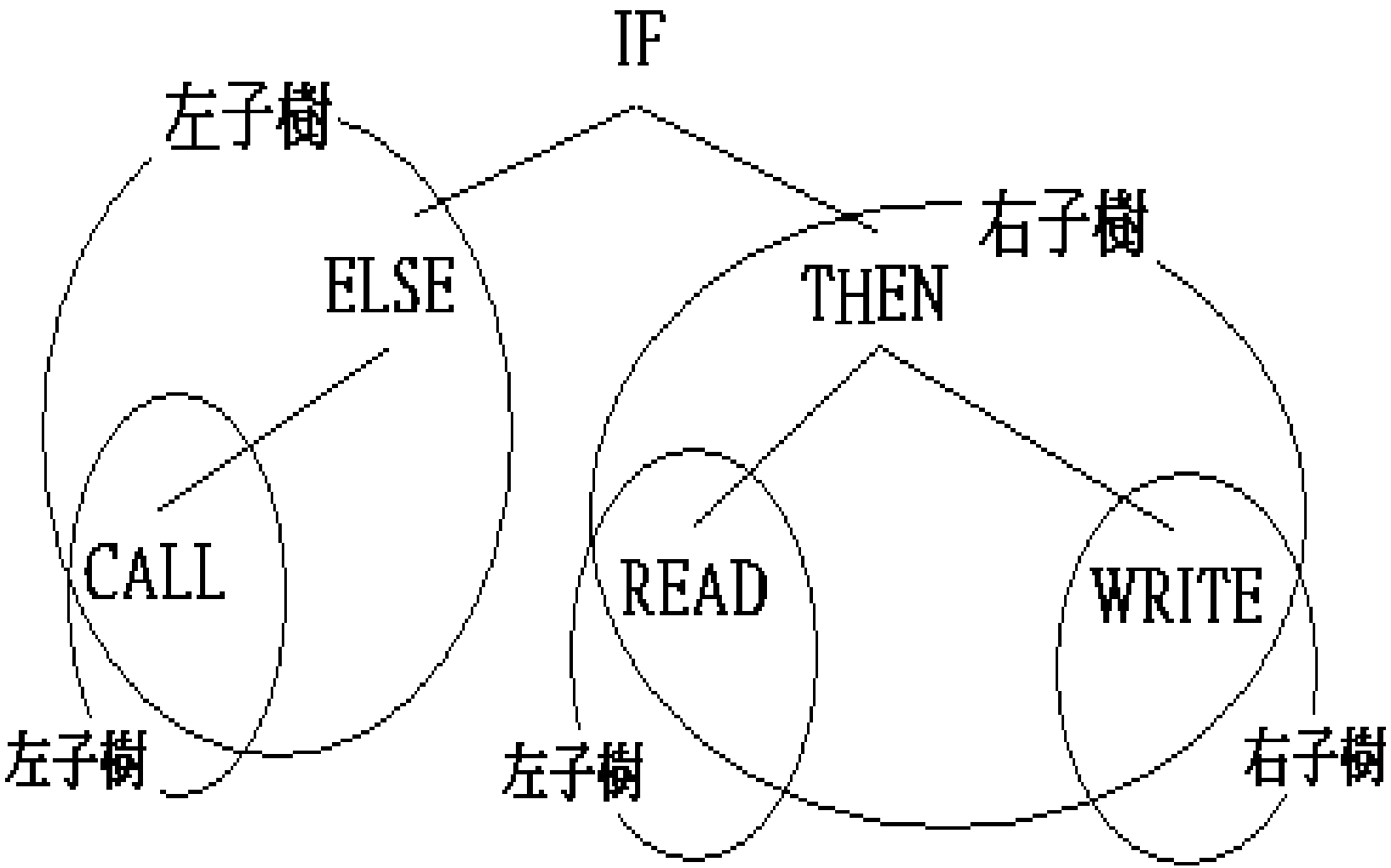
```
int main()
{
  char word[36];
  tail = malloc(sizeof(struct wordTag));
  strcpy(tail->word, "}");              /*比英文字母大*/
  tail->next = NULL;
  head=malloc(sizeof(struct wordTag));
  strcpy(head->word, "!");              /*比英文字母小*/
  head->next = tail;
  printf("請逐次輸入字串插入連結串列，Ctrl-Z結束\n");
  while (fgets(word,sizeof(word),stdin)!=NULL)
  {
    word[strlen(word)-1]='\0';          /*清除\n為\0*/
    insertNode(word);
    printf("%s\n", word);
  }
  printf("逐次從連結串列前端取出後輸出\n");
  listprint();
  return 0;
}
```

3.11 樹結構

- 樹結構（tree）一般常用的為二元樹（binary tree），每一個節點最多有兩個子樹（subtree），左邊其比較值較小的稱為左子樹（left subtree），右邊其比較值較大的稱為右子樹（right subtree），樹葉（leaf）節點沒有左右子樹。下列 tree.c 執行時分別輸入下列的識別字字串：
- IF、THEN、ELSE、CALL、READ、WRITE
- 構成如下的樹結構：



- 要取出樹結構裡的每一個節點，有許多方法，其中一種稱為中序法（in order），先取左子樹，再取節點，再取右子樹。以上圖的樹結構為例，說明如下：
- 對於 IF 節點而言，其左子樹為 ELSE、CALL，其右子樹為 THEN、READ、WRITE。對於 ELSE 節點而言，其左子樹為 CALL，沒有右子樹。對於 THEN 節點而言，其左子樹為 READ，其右子樹 WRITE。對於 CALL、READ、WRITE 等三個節點而言，都沒有左子樹，也沒有右子樹，均屬於樹葉節點 。以中序法取出樹結構每一個節點的順序如下：
-
- **(左子樹) IF (右子樹)**
-
- **(左子樹 ELSE 右子樹) IF (左子樹 THEN 右子樹)**
-
- **(CALL ELSE) IF (READ THEN WRITE)**
-
- **CALL ELSE IF READ THEN WRITE**


IF
左子樹
右子樹
ELSE
THEN
CALL
READ
WRITE
左子樹
左子樹
右子樹

```
/********************** tree.c **********************/
#include <stdio.h>
#include <stdlib.h>
struct wordTag
{
  char word[36];
  struct wordTag *left, *right;
};
struct wordTag *root;
void createRoot(char word[])
{
  root=malloc(sizeof(struct wordTag));
  root->left=NULL;
  root->right=NULL;
  strcpy(root->word, word);
}
```

```
struct wordTag *insertNode(char word[])
{
  struct wordTag *p = root;
  struct wordTag *q = root;
  while (p!=NULL)
  {
    q = p;
    if (strcmp(word, p->word) < 0)
      p = p->left;
    else if (strcmp(word, p->word) > 0)
      p = p->right;
    else
      return p;
  }
  p = malloc(sizeof(struct wordTag));
  p->left = NULL;
  p->right = NULL;
  strcpy(p->word, word);
  if (strcmp(word, q->word) < 0)
    q->left = p;
  else
    q->right = p;
}
```

```
void inOrder(struct wordTag *p)
{
  if (p != NULL)
  {
    inOrder(p->left);
    printf("0x%p\t0x%p\t0x%p\t\"%s\"\n",
          p, p->left, p->right, p->word);
    inOrder(p->right);
  }
}
```

```
int main()
{
  char word[36];
  root = NULL;
  printf("請逐次輸入字串插入樹結構，Ctrl-Z結束\n");
  while (fgets(word,sizeof(word),stdin)!=NULL)
  {
    word[strlen(word)-1] = '\0';
    if (root==NULL)
      createRoot(word);
    else
      insertNode(word);
  }
  printf("\t樹根root=0x%p 中序(in order)取出\n", root);
  printf("\n  指標 \t\t左鏈 \t\t右鏈 \t\t識別字\n");
  inOrder(root);
  return 0;
}
```