Strutture dati dinamiche in C (III)

Laboratorio di Linguaggi di Programmazione
a.a. 2001/2002

dott.ssa Francesca A. Lisi
lisi@di.uniba.it

Sommario

- Alcune *utilities* per la gestione di stringhe
- Gli alberi binari di ricerca
  - Realizzazione di dizionari
- Le tabelle hash
  - Realizzazione di tabelle dei simboli per compilatori

Riferimenti

- cap. 12 di Deitel & Deitel
- par. 6.5-6.6 di Kernighan & Ritchie
- cap.1 di Semeraro

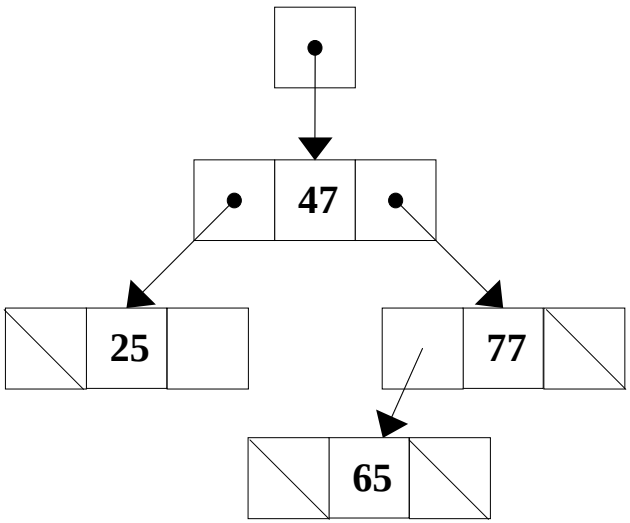
Utilities per stringhe di caratteri

```
char *strdup(char *s) {
  char *p;
  p = (char*)malloc(strlen(s) + 1);
  if (p != NULL) strcpy(p,s);
  return p;
}

int getword(char *word, int lim) {
   scanf("%s",word);
   word[lim] = '\0';
   return (word[0] != '*');
}
```

Alberi binari di ricerca

- Assenza di nodi duplicati
- Esistenza di una relazione di ordinamento sui nodi t.c.:
  - I valori contenuti nei nodi di ogni sottoalbero sinistro sono **minori** del valore contenuto nel rispettivo nodo padre
  - I valori contenuti nei nodi di ogni sottoalbero sinistro sono **maggiori** del valore contenuto nel rispettivo nodo padre
- Es. dizionario



Realizzazione di un dizionario

```
/* bozza di programma */
#define MAXWORD 100

struct tnode {
   char *word;
   struct tnode *left, *right;
};

struct tnode *talloc(void) {
   return (struct tnode *)
          malloc(sizeof(struct tnode));
}
```

Realizzazione di un dizionario (II)

```
struct tnode *addEl(struct tnode *,char *);
void treeprint(struct tnode *);

int main() {
    struct tnode *root = NULL;
    char word[MAXWORD];
    while (getword(word,MAXWORD) != 0)
          root = addEl(root,word);
    treeprint(root);
    system("PAUSE");
    return 0;}
```

Realizzazione di un dizionario (III)

```
struct tnode *addEl(struct tnode *p,char *w){
   int cond;
   if (p == NULL) {
      p = talloc();
      p->word = strdup(w);
      p->left = p->right = NULL;}
   else if ((cond = strcmp(w,p->word)) == 0)
      printf("Gia' esistente!\n");
   else if (cond < 0)
      p->left = addEl(p->left,w);
   else p->right = addEl(p->right,w);
   return p;}
```

Realizzazione di un dizionario (IV)

```
void treeprint(struct tnode *p) {
    if (p != NULL) {
        treeprint(p->left);
        printf("%s\n",p->word);
        treeprint(p->right);
        }
}
```

Esercizi

- Scrivere una funzione C che esegua la ricerca di una parola all'interno del dizionario e risponda si/no
- Scrivere una funzione C che esegua la ricerca di una parola all'interno del dizionario e ne restituisca la posizione, se la trova, altrimenti NULL
- Scrivere un programma C che memorizzi in ogni voce del dizionario il numero d'ordine di inserimento della parola

Tabelle hash

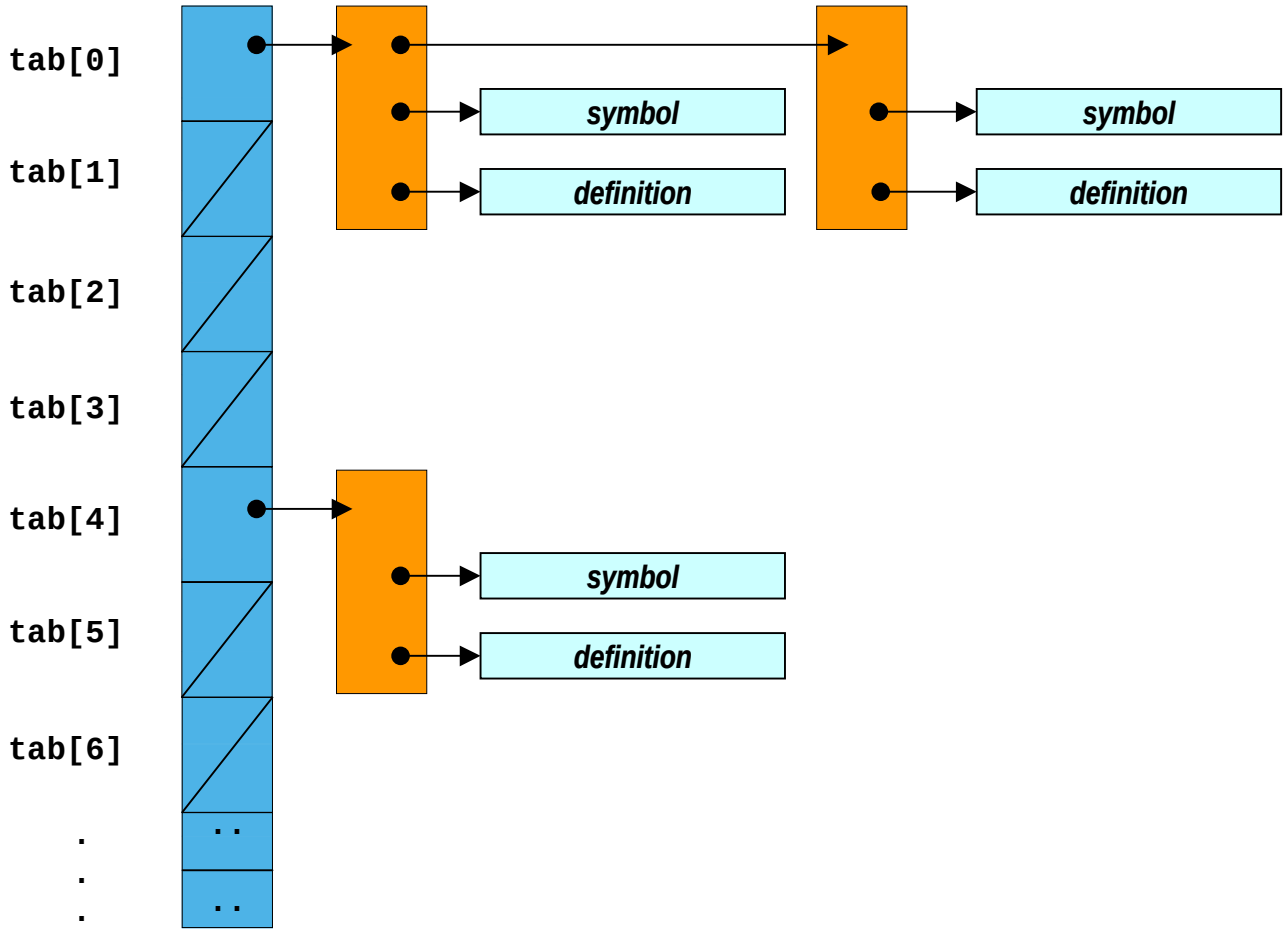
- Elementi memorizzati senza ordine prefissato
- Accesso mediante una *funzione hash* che, data la chiave dell'elemento cercato, restituisce un indirizzo di memoria

$$\textbf{\textit{hash:chiave} } \rightarrow \textbf{\textit{ indirizzo}}$$

ottenuto mediante calcoli sui caratteri della chiave

- Se la funzione hash non è biunivoca, le *collisioni* sono tipicamente gestite prevedendo un'*area di trabocco* per ogni entry della tabella.

Esempio di *symbol table*



Realizzazione di una *symbol table*

```
#include <stdio.h>
#include <stdlib.h>

#define HASHSIZE 101

struct nlist {
   struct nlist  *next;
   char *name, *defn;
};

static struct nlist  *hashtab[HASHSIZE];
```

Realizzazione di una *symbol table* (II)

```
unsigned hash(char *s) {
/* calcola il valore HASH di s */
   unsigned hashval;

   for (hashval = 0; *s != '\0'; s++)
      hashval = *s + 31 * hashval;
   return hashval % HASHSIZE;
}
```

Realizzazione di una *symbol table* (III)

```
struct nlist  *lookup(char *s) {
/* cerca s nella tabella HASH */
   struct nlist  *np;

   for (np = hashtab[hash(s)]; np != NULL;
  np = np->next)
      if (strcmp(s,np->name) == 0)
         return np;
   return NULL;
}
```

Realizzazione di una *symbol table* (IV)

```
struct nlist  *install(char *name, char *defn) {
/* inserisce un elemento */
   struct nlist  *np; unsigned hashval;

   if ((np = lookup(name)) == NULL) {
      np = (struct nlist  *)malloc(sizeof(*np));
      if (np == NULL || (np->name = strdup(name)) == NULL)
         return NULL;
      hashval = hash(name);
      np->next = hashtab[hashval];
      hashtab[hashval] = np;
      }
   else free((void *)np->defn);
   if ((np->defn = strdup(defn)) == NULL) return NULL;
   return np;
}
```

Esercizi

- Scrivere una funzione C che stampi il contenuto di una tabella hash mettendone in evidenza le aree di trabocco
- Scrivere un programma C che acquisisca simboli e le loro definizioni e le inserisca in una *symbol table*
- Realizzare una *symbol table* che gestisca le variabili, memorizzandone le seguenti proprietà:
  - nome simbolico (stringa di caratteri)
  - tipo (‘I’ per **int**, ‘C’ per **char**, ‘F’ per **float**, etc.)
  - indirizzo (un intero da 0 a 99)