Strutture dati dinamiche in C

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

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

Prerequisiti ed obiettivi

- Uso elementare di puntatori, vettori e strutture
- Conoscenza delle grammatiche a produzioni

- Costruzione di programmi che fanno uso di strutture dati dinamiche
  - Tali programmi richiedono meccanismi di *allocazione* dinamica e *deallocazione* esplicita della memoria
- Realizzazione dei tipi *lista* ed *albero* e loro utilizzo nella costruzione di compilatori
  - cfr. cap. 12 di Deitel & Deitel

Definizione di nuovi tipi

typedef *tipo* Nome**;**

- Semplici sinonimi di tipi esistenti
  - Possibilità di dare un nome a tipi complessi
- Interpretata dal compilatore (`#define` no)
  - Parametrizzazione di programmi (portabilità)
  - Significatività dei nomi (documentabilità)

Forzatura di tipi

(*tipo***)** *espressione*

- Operatore di “Cast”
- Forzatura di un valore ad un tipo
  - Il tipo deve essere compatibile col valore

Allocazione dinamica

Pascal	C
var puntatore**: ^***tipo***;**	*tipo* *****puntatore**;**
new(puntatore**)**	puntatore **= (***tipo* ***)** **malloc(***dim***);**
dispose(puntatore**)**	**free(**puntatore**);**
nil	**null**

Allocazione dinamica

void *malloc(size_t n_bytes**)**

- Definita in `<stdlib.h>`
- Valori di ritorno:
  - Indirizzo per un oggetto del tipo richiesto
  - **null** se lo spazio non è disponibile
- Cast obbligatorio per il tipo richiesto
- Richiede il numero di byte da allocare
  - Calcolabile mediante l'operatore **sizeof()**

Allocazione dinamica

size_t sizeof(*oggetto***)**

size_t sizeof(*tipo***)**

- Restituisce la dimensione di oggetto o tipo

  typedef size_t = unsigned int

  - Definito in `<stddef.h>`

void free(void *puntatore)

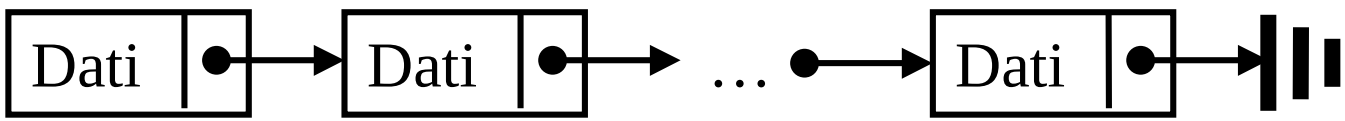
- Libera la memoria dell'oggetto in puntatore
  - Deve essere stata allocata dinamicamente

Strutture dinamiche

- Elementi omogenei (= dello stesso tipo)
- Residenti in memoria centrale
- Numero di elementi non fissato
  - Aggiunta secondo necessità
    - Allocazione dinamica della memoria
  - Accesso attraverso elementi precedenti
    - Elemento =
      Dati + Puntatori agli elementi seguenti

Liste concatenate

- Sequenze lineari di elementi
  - Accesso tramite scansione sequenziale
    - Ciascun elemento punta al successivo
    - L'ultimo elemento ha un terminatore



Liste concatenate

```
struct el_lista {
… /* dati */
struct el_lista *el_succ;
} *lista = null;

struct el_lista *new_el =
    (struct el_lista *)malloc(sizeof(struct el_lista));
(*elem).el_succ = new_el;
(*new_el).el_succ = null;
```

Acquisizione di una *lista di interi*

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

struct el_lista {
   int dato;
   struct el_lista *el_succ;
   };
typedef struct el_lista *Elem;

/* prototipi di funzione */
int stampa_lista(Elem);
Elem add_el(void);
```

Acquisizione di una *lista di interi* (II)

```
main() {
   Elem lista =  NULL, new_el, last_el;

   do {
      if ((new_el = add_el()) != NULL) {
         if (lista == NULL) lista = new_el;
         else  (*last_el).el_succ = new_el;
         last_el = new_el;
         }
   } while (new_el != NULL);
   stampa_lista(lista);
   system("PAUSE"); return 0; }
```

Acquisizione di una *lista di interi* (III)

```
Elem add_el(void) {
   int i=0; Elem new_el;

   printf("Inserisci un intero positivo: ");
   scanf("%d",&i); printf("\n");
   if (i > 0) {
      new_el =
     (Elem)malloc(sizeof(struct el_lista));
      (*new_el).dato = i;
      (*new_el).el_succ = NULL;
      return new_el;}
   else return NULL;}
```

Acquisizione di una *lista di interi* (IV)

```
int stampa_lista(Elem last_el) {
  printf("\nLa lista inserita è:\n");
  while(last_el != NULL) {
      printf("%i\n",(*last_el).dato);
      last_el = (*last_el).el_succ;
  }
}
```

L'acquisizione di una grammatica: il problema

Si vuole acquisire una grammatica G. Per semplicità, assumiamo che G consista di al più 10 produzioni nella forma BNF.

Scelta delle strutture dati:

- l'insieme delle produzioni è rappresentato mediante un vettore di 10 elementi
- ogni produzione è strutturata in una parte sinistra ed una parte destra.
- ciascuna delle due parti di una produzione è rappresentata come stringa di caratteri a lunghezza variabile (puntatore a caratteri)

L'acquisizione di una grammatica: una bozza di soluzione

```
/* programma acq-gramm.c */
#include <stdio.h>
#include <stdlib.h>

struct StructProd {
  char *sx; /* parte sinistra */
  char *dx; /* parte destra */
};
typedef struct StructProd TipoProd;
```

L'acquisizione di una grammatica: una bozza di soluzione (II)

```
int main(void) {
  TipoProd gramm[10]; int i=0, more=1;
  char risp;

  while (i<10) and more {
     gramm[i]= acqProd;
     printf("Altra produzione? (s/n)", ..);
     scanf(.., &risp);
     if (risp=='n') more = 0;
     else i=i+1;
  }
return 0; }
```

Esercizi sull'`acquisizione` di una grammatica

- Completare la bozza di soluzione, specificando in particolare prototipo e definizione della funzione `acqProd`
- Generalizzare `acqProd` in modo tale che acquisisca uno per uno i simboli che compaiono nelle parti sinistra e destra di ciascuna produzione.

  Suggerimenti:

  – rappresentare ciascuna delle due parti di una produzione come lista di stringhe (una stringa per ogni simbolo)

  – inserire le stringhe in coda alla lista