seconda prova scritta

20. 12. 2000

Correzione

Puntatori

- Che cos'è un valore riferimento?
- Un valore riferimento (o valore puntatore) è **NULL** oppure l'indirizzo di una variabile (detta variabile referente).

- Che cos'è una variabile riferimento?
- Una variabile di tipo riferimento (puntatore) è una variabile il cui valore è un valore riferimento.

Operatori sui puntatori

- A quali espressioni si applica l'operatore unario &, e cosa restituisce?
- Si applica ad un qualsiasi designatore e restituisce l'indirizzo di memoria della variabile da esso designata.

- A quali espressioni si applica l'operatore unario di dereferenziazione *, e cosa restituisce?
- Si applica ad una espressione di tipo puntatore il cui valore sia diverso da NULL, e restituisce la variabile referente.

Puntatori

- Le seguenti espressioni di inizializzazione sono corrette? Perché?

```
char *p = &((char*) p);
double *p = &((double*) p);
```

- Entrambe le espressioni non sono corrette. Nella prima, alla variabile p che è di tipo **`char*`** viene assegnato un valore di tipo **`char**`** (l'indirizzo di una variabile di tipo **`char*`**). Nella seconda, alla variabile p che è di tipo **`double*`** viene assegnato un valore di tipo **`double**`**.

Array e Stringhe

- Se la variabile **tab** è dichiarata come **char* tab[] = {"a","i","u","t","o"};**
  - L'espressione **tab[2]** è un designatore?
    Si
  - qual è il suo tipo?
    char *
  - qual è il suo valore?
    l'indirizzo di una variabile dinamica costituita da un array di 2 caratteri ('u' e \0).

Array e Stringhe

- Se la variabile **tab** è dichiarata come **char* arr[] = {"p","a","n","i","c","o"};**
  - L'espressione **arr[3]**è un designatore?
    Si
  - qual è il suo tipo?
    char *
  - qual è il suo valore?
    l'indirizzo di una variabile dinamica costituita da un array di 2 caratteri ('i' e \0).

Stringhe

- Come si può verificare se due variabili di tipo stringa rappresentano lo stesso valore?
- Usando la funzione **`strcmp`** definita nella libreria **`string.h`**

- E' possibile confrontare il contenuto di due stringhe con l'operatore relazionale ==? Perché?
- No, perché == opera il confronto tra i due valori di tipo **`char*`**, che sono indirizzi.

Aritmetica dei puntatori

- E' possibile usare l'aritmetica dei puntatori sulle liste? Perché?
- No, perché essa è definita solo sugli array

- E' possibile usare l'aritmetica dei puntatori sulle stringhe? Perché?
- Si, perché le stringhe sono array.

Tipi enum e struct

- Dichiarare un tipo enumerazione **Seme** che rappresenta i 4 semi delle carte da poker (cuori, fiori, ecc.).

```
typedef enum Seme {cuori,quadri,fiori,picche} Seme;
```

- Utilizzando il tipo **Seme** dichiarato nell'esercizio precedente, dichiarare un tipo record **Carta**, che rappresenta le carte da gioco, con due campi: **seme** e **numero**.

```
typedef struct {
      Seme seme;
      int numero;
} Carta;
```

Tipi dinamici

- Utilizzando il tipo **Carta** dell'esercizio precedente, dichiarare un tipo dinamico **ListaCarte** che rappresenta un lista semplice di carte, con due campi: **carta** e **next**.

```
typedef struct ListaCarte {
      Carta carta;
      struct ListaCarte* next;
} ListaCarte;
```

Valori di tipo lista semplice

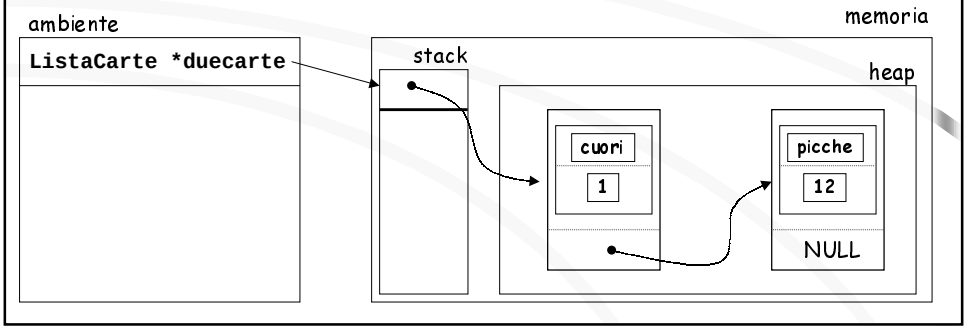
- Utilizzando i tipi definiti precedentemente, inizializzare una variabile **duecarte** di tipo puntatore a **ListaCarte** in modo che punti ad una lista di due carte: l'asso di cuori e la donna di picche.

-
```
ListaCarte *duecarte;

duecarte = (ListaCarte*) malloc(sizeof(ListaCarte));
(duecarte->carta).numero = 1;
(duecarte->carta).seme = cuori;
duecarte->next =
    (ListaCarte*)malloc(sizeof(ListaCarte));
(duecarte->next->carta).numero = 12;
(duecarte->next->carta).seme = picche;
duecarte->next->next = NULL;
```

Modello ambiente-memoria

- Utilizzando il modello ambiente-memoria, descrivere graficamente la situazione al termine dei comandi necessari a realizzare quanto richiesto dall'esercizio precedente.



- Utilizzando i tipi definiti negli esercizi precedenti, scrivere la definizione di una funzione che, prendendo come parametro attuale una lista di carte verifica se hanno tutte allo stesso seme (e restituisce 1 se questo è vero, e 0 altrimenti).

```
int colore(ListaCarte *lista){
      Seme seme_prima_carta;
      int risultato = 1;

      if (lista!= NULL)
            seme_prima_carta = (lista->carta).seme;
      while (lista != NULL){
            if ((lista->carta).seme != seme_prima_carta){
                  risultato = 0;
                  break;
            }
            else
                  lista = lista->next;
      }
      return risultato;
}
```

Tipi enum

- Dichiarare un tipo enumerazione **Colore** che rappresenta i 2 colori delle pedine degli scacchi (bianco e nero), ed un tipo enumerazione **Personaggio** che rappresenta i diversi tipi di pedina (re, regina, alfiere, cavallo, torre e pedone).

- ```
  typedef enum Colore {bianco, nero} Colore;

  typedef enum Personaggio {re, regina, alfiere,
    cavallo, torre, pedone} Personaggio;
  ```
  ```

Tipi struct

- Dichiarare un tipo record **Pedina**, che rappresenta una pedina degli scacchi, con due campi: **colore** e **personaggio**, di tipo **Colore** e **Personaggio**, rispettivamente.

- ```
  typedef struct {
    Colore      colore;
    Personaggio personaggio;
  } Pedina;
  ```

Tipi struct

- Dichiarare un tipo record Casella che rappresenta una casella della scacchiera, ed è un record di due campi: **colore** (il colore della casella, di tipo **Colore**) e **pezzo**, un riferimento ad una variabile di tipo **Pedina** (il riferimento sarà **NULL** se la casella non contiene nessuna pedina, altrimenti conterrà l'indirizzo di una variabile di tipo Pedina).

- ```
  typedef struct {
    Colore colore;
    Pedina *pezzo;
  } Casella;
  ```

Array bidimensionali

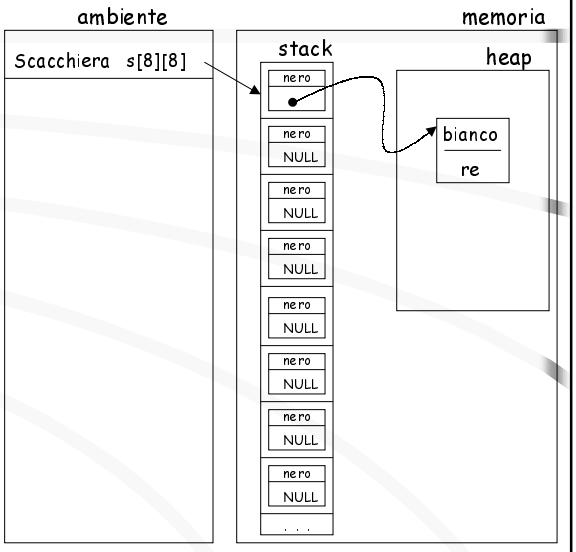
- Dichiarare il tipo **Scacchiera**, i cui elementi sono tabelle bidimensionali di dimensione 8 per 8 aventi come di tipo base il tipo **Casella**. Un valore di tipo **Scacchiera** rappresenta una possibile configurazione della scacchiera.
- **typedef Casella Scacchiera[8][8];**

Array bisimensionali

- Utilizzando i tipi definiti precedentemente, inizializzare una variabile **s** di tipo **Scacchiera** in modo che rappresenti una scacchiera in cui tutte le caselle sono nere e nella quale è presente solo il re bianco (in una cella qualsiasi).
-
```
Scacchiera s;
int i,j;
Pedina *king=(Pedina*) malloc(sizeof(Pedina));
king->colore = bianco;
king->personaggio = re;
for (i=0; i<8; i++)
   for (j=0; j<8; j++){
     s[i][j].colore = nero;
     s[i][j].pezzo = NULL;
   }
s[0][0].pezzo = king;
```

Modello ambiente-memoria

- Utilizzando il modello ambiente-memoria, descrivere graficamente la situazione al termine dei comandi necessari a realizzare quanto richiesto dall'esercizio precedente.



- Utilizzando i tipi definiti negli esercizi precedenti, scrivere la definizione di una funzione che, prendendo come parametro attuale un valore s di tipo **Scacchiera** verifica se la torre nera è presente nella scacchiera **s**. La funzione restituisce 1 se la torre nera è presente, altrimenti restituisce 0.

```
int cercaTorreNera(Scacchiera s){
  int i,j,trovato = 0;

  for (i=0; i<8; i++)
    for (j=0; j<8; j++){
      if ((s[i][j].pezzo != NULL) &&
            ((s[i][j].pezzo)->colore == nero) &&
            ((s[i][j].pezzo)->personaggio == torre) ){
         trovato = 1;
      }
    }
  return trovato;
}
```

Variabili statiche e dinamiche

- E' vero o no che ogni variabile (statica o dinamica) ha un indirizzo che la identifica univocamente?
- Si, è vero.

Variabili dinamiche

- In cosa differiscono le variabili dinamiche da quelle statiche?
- Una variabile dinamica, a differenza delle variabili statiche
  - può essere creata come risultato di una computazione
  - non è legata ad un nome
  - ha una durata che non dipende dal contesto di creazione

Allocazione dinamica

- Cosa restituisce la funzione **`malloc`** ?
- Alla funzione **`malloc`** deve essere passata come parametro la dimensione del tipo della variabile che si vuole creare (variabile referente). La funzione **`malloc`** alloca memoria per tale nuova variabile e ne restituisce l'indirizzo.

Allocazione dinamica

- Cosa restituisce l'espressione **`(double*) malloc(sizeof(double))`** ?
- L'indirizzo di una nuova variabile di tipo **`double`** allocata nello heap.

Espressioni booleane

- Si consideri **`int *x;`** Per quali valori di **x** l'espressione booleana **`((x!=NULL) || (x!=&y))`** è vera?
- Per ogni valore di **x.** Infatti i valori di una variabile **`int*`** possono essere o **NULL** o l'indirizzo di una variabile.
  Se **`x!=NULL`** allora il primo disgiunto è vero, e l'intera espressione è vera.
  Se **`x==NULL`** allora la sottoespressione **`x!=&y`** è vera, e quindi anche l'intera espressione è vera.

Espressioni booleane

- Si consideri **`int *x;`**
  Per quali valori di **x** l'espressione booleana **`!((x==NULL) && (x==&y))`** è falsa?
- Per nessun valore di **x** (non è possibile che sia uguale sia a **NULL** che all'indirizzo di una variabile!)

Aritmetica dei puntatori

- Qual è il valore delle variabili **`*v, w[0] e w[1]`** dopo l'esecuzione dei comandi seguenti?

```
int *v, w[2];
w[1] = 1;
w[0] = 0;
v = w;
*v++ = 2;
```

- **`w[0]=2, w[1]=1, *v=1`**

Aritmetica dei puntatori

- Qual è il valore delle variabili *x, y[0], y[1] dopo l'esecuzione dei comandi seguenti?

```
int *x, y[2];
y[0] = 0;
y[1] = 1;
x = y;
++*x;
```

- **`y[0]=1, y[1]=1, *x=1`**

Procedure

- In cosa consistono la dichiarazione e la definizione di una procedura?
- Nella dichiarazione viene espressa la firma della procedura, mentre nella definizione ne viene specificato anche il corpo.

- In cosa possono differire la dichiarazione e la definizione di una stessa procedura?
- Nel nome dei parametri formali (che nella dichiarazione possono anche essere omessi). Ovviamente, la dichiarazione ha solo la firma (non il corpo).

Procedure

- Si considerino le seguenti definizioni di costante di procedura:

```
int sempre_uno(int x){ return 1; }
int sempre_due(int x){ return 2; }
```

Si definisca il tipo **Sempre_N** al quale appartengono sia **sempre_uno** che **sempre_due**.

- **typedef int (*Sempre_N)(int x);**

Array di Procedure

- Utilizzando il tipo **Sempre_N** dell'esercizio precedente, scrivere i comandi necessari per dichiarare una variabile di tipo array di due elementi di tipo **Sempre_N**, ed inizializzare l'array con le due procedure **sempre_uno** e **sempre_due**.

- **Sempre_N f[2]= {sempre_uno,sempre_due};**

Procedure

- Si considerino le seguenti definizioni di costante di procedura:

```
char sempre_a(char x){ return 'a'; }
char sempre_b(char x){ return 'b'; }
```

Si definisca il tipo **Sempre_k** al quale appartengono sia **sempre_a** che **sempre_b**.

- **typedef char (*Sempre_k)(char x);**

Variabili di tipo procedura

- Utilizzando il tipo **Sempre_k** dell'esercizio precedente, scrivere i comandi necessari per dichiarare due variabili di tipo **Sempre_k**, ed assegnare loro, rispettivamente, la costante di procedura **sempre_a** e **sempre_b**.

- ```
  Sempre_k f = sempre_a;
  Sempre_k g = sempre_b;
  ```

Terminazione

- Per quali valori del parametro attuale la seguente procedura termina correttamente, assumendo che Lista sia un tipo lista di interi con campi val e next?

```
int foo(Lista *lista){
  int i;
  if ((lista == NULL) || (lista->val == 0))
    return 1;
  else{
    i = lista->val;
    while (i != 0)
       i = foo(lista->next);
    return i;
  }
}
```

- Solo per lista == NULL oppure lista != NULL e lista->val == 0.
```

Terminazione

- Per quali valori del parametri attuale la seguente procedura termina correttamente, assumendo che **Lista** sia un tipo lista di interi con campi **val** e **next**?

```
int foo(Lista *lista){
  int i;
  if ((lista != NULL) && (lista->val != 0)) {
       i = lista->val;
       while (i != 0)
              i = foo(lista->next);
       return i;
  }
  else
       return 1;
}
```

- Solo per lista==NULL o per lista!=NULL e lista->val==0.