Programmazione I - Laboratorio

Esercitazione 4 - Puntatori, vettori e stringhe

Gianluca Mezzetti[1] Paolo Milazzo[2]

1. Dipartimento di Informatica, Università di Pisa
http://www.di.unipi.it/∼mezzetti
mezzetti@di.unipi.it

2. Dipartimento di Informatica, Università di Pisa
http://www.di.unipi.it/∼milazzo
milazzo@di.unipi.it

Corso di Laurea in Informatica
A.A. 2012/2013

Puntatori (1)

Un puntatore è una variabile che contiene l'indirizzo di memoria di un'altra variabile.

I puntatori consentono di realizzare strutture dati dinamiche (e.g. liste)

Gli operatori chiave per l'utilizzo dei puntatori sono:

- l'operatore di dereferenziazione $*$ che permette di accedere alla locazione di memoria il cui indirizzo è memorizzato dal puntatore
- l'operatore & che consente di ottenere l'indirizzo di memoria di una data variabile

Puntatori (2)

Un puntatore si dichiara usando l'operatore di dereferenziazione come segue:

```
int *p1, *p2;
char *s;
```

dove `int` e `char` sono il tipo della variabile che sarà riferita dal puntatore
Per assegnare a un puntatore l'indirizzo di una variabile nota si procede come segue:

```
int x=0;
p1 = &x;
```

Per leggere o modificare il dato nella locazione di memoria riferita da un puntatore si può usare l'operatore di dereferenziazione in una espressione o in un assegnamento:

```
x = *p1 + 1; /* x prende valore 1 */
*p1 = x + 1  /* x prende valore 2 (tramite il puntatore) */
```

Puntatori ed argomenti di funzione (1)

Poichè il C passa alle funzioni gli argomenti per valore, la funzione chiamata non può modificare una variabile della funzione chiamante.

Per esempio una funzione `swap` che scambia i valori di due variabili definita come segue non ha in realtà nessun effetto:

```
void swap(int x, int y)        /* SBAGLIATO */
{
  int temp;

  temp = x;
  x = y;
  y = temp;
}
```

Puntatori ed argomenti di funzione (2)

Il modo per ottenere l'effetto desiderato (emula il passaggio dei parametri per riferimento) è utilizzando puntatori come argomenti della funzione:

```
void swap(int *x, int *y)       /* CORRETTO */
{
  int temp;

  temp = *x;
  *x = *y;
  *y = temp;
}
```

A questo punto, se vogliamo scambiare i valori delle variabili a e b, invocheremo swap come segue:

```
swap(&a, &b);
```

Puntatori ed argomenti di funzione (3)

Anche il passaggio di un array ad una funzione ha un comportamento simile al passaggio per riferimento, ad esempio

```
#include <stdio.h>

int modifica_ultimo(int a[],int size);

int main()
{
  int a[5] = {1,2,3,4,5};
  modifica_ultimo(a,5);
  printf("%d\n",a[4]); /* stampa 10 */
}

int modifica_ultimo(int a[], int size)
{
  a[size-1] = a[size-1]+5;
}
```

Note:

- a[] denota un parametro formale che è un array di cui si ignora la dimensione
- Solitamente la dimensione dell'array si passa con un altro parametro intero (size in questo caso)

Puntatori e array (1)

La relazione esistente tra puntatori e array è molto stretta (sostanzialmente sono la stessa cosa)

Qualsiasi operazione effettuabile indicizzando un array può essere fatta anche tramite un puntatore

Ad esempio, il seguente codice definisce un array a e un puntatore pa che punta all'inizio dell'array:

```
int a[10];
int *pa;
pa = &a[0];
```

Puntatori e array (2)

A questo punto possiamo accedere agli elementi dell'array in modo tradizionale usando `a[0]`,`a[1]`,`...` oppure usando `pa` e l'aritmetica dei puntatori.

In sostanza:

- Sommare un intero `n` a un puntatore corrisponde a incrementare l'indirizzo di memoria contenuto nel puntatore di `n*sizeof(tipo_puntato)`
- Sottrarre due puntatori da come risultato la differenza numerica tra gli indirizzi divisa per `sizeof(tipo_puntato)`

dove `sizeof(tipo_puntato)` è la quantità di memoria occupata da un valore di tipo `tipo_puntato`

Quindi: `*(pa+i)` è un modo alternativo per accedere ad `a[i]`

Stringhe

In C non esiste un tipo stringa, piuttosto una stringa è un puntatore ad una sequenza di *char* terminata dal carattere nullo $'\backslash 0'$. Le funzioni di libreria sono conformi a questa visione.
La libreria *string.h* contiene una serie di funzioni per manipolare stringhe:

strncat(s,t,n)	concatena n caratteri di t alla fine di s
strcmp(s,t)	restituisce un numero negativo, 0 o positivo se lessicograficamente $s < t, s == t, s > t$
strncmp(s,t,n)	come strcmp ma solamente i primi n caratteri
strcpy(s,t)	copia t in s
strncpy(s,t,n)	copia t in s, copia al più n caratteri
strlen(s)	numero di caratteri di s
strchr(s,c)	puntatore alla prima occorrenza di c in s
strrchr(s,c)	puntatore all' ultima occorrenza di c in s

s e t sono $char*$ e c ed n sono *int*.

Utilizzi non sicuri delle funzioni di input

Attenzione!!

Non usare `scanf(...)` con un input `%s` o `gets()`

Meglio usare `getchar()` e leggere un carattere per volta

Esempio: Stringhe I

```
#include <stdio.h>
#include <string.h>
#define SSIZE 20
#define INPUTSIZE 10

int min(int,int);
int main(void)
{
        char *s0;
        char *s1;
        char s2[SSIZE];
        char s3[SSIZE] = {'a','r','r','a','y','\0'};
        char s4[SSIZE];
        int i=0;
        int ch;


        printf("%p\n","ciao"); /*ciao e' una stringa (puntatore a
            una sequenza di caratteri terminata da \0) che viene
            piazzata nella memoria gia' inizializzata del programma
            */

        /*Gli array sono puntatori, se sono terminati da un
            carattere nullo sono quindi anche stringhe*/
        s0 = s3;
        printf("%p - %p || %s - %s \n",s0,s3,s0,s3);
```

Esempio: Stringhe II

```
        /*Possiamo prendere il puntatore al 3 carattere*/
        s0 = &s3[2];
        printf("%p - %p || %s - %s \n",s0,s3,s0,s3);
        /*Che e' la stessa cosa di */
        printf("%d\n",&s3[2] == s3+2);

        s0 = "prova";
        s1 = "ciaociao";
        printf("%p - %p\n",s0,s1); /* L'indirizzo del prova/ciao
            alla riga sopra e' ora in s0/s1*/

        /* Il mio compilatore le mette una di seguito all'altra,*/
        printf("%ld\n",s1-s0);/*Stampa 6 perche' len(prova)=5 ma
            viene aggiunto il carattere nullo al termine.*/

        /* Prendiamo due stringhe da input usando solo la funzione
            getchar notare che getchar prende anche gli a capo come
             caratteri */
        /*Usare Ctrl-D per mandare EOF su stdin */
        while((ch = getchar()) != EOF && i < SSIZE-INPUTSIZE-1){
        s2[i] = ch;
        /* *(s2+i) = ch */ /* alternativa */
        i++;
    }
    s2[SSIZE-INPUTSIZE-1]=0;
        printf("\n");
```

Esempio: Stringhe III

```
        /* s2 e' ora una stringa perche' s2[SSIZE-INPUTSIZE-1] e'
            il carattere nullo*/
        printf("Hai inserito : %s\n",s2);

        /*Concateniamo s2 ed s3 nello spazio che abbiamo*/
        strncat(s2,s3, min(SSIZE-strlen(s2)-1,strlen(s3)));
        printf("%s \n",s2);

        /*Compariamo s2 ed s3*/
        printf("Risultato comparazione di %s ed %s: %d \n",s2,s3,
            strcmp(s2,s3));

        printf("%lu\n",strlen(s4)); /* Questo valore non si puo'
            predire, infatti le variabili locali non vengono
            inizializzate e quindi il loro contenuto puo' essere
            zero o meno, posso anche ottenere un segmentation fault
             perche' se non trova nessuno zero che la fermi la
            funzione strlen puo' andare a leggere una porzione di
            memoria che non e' concesso leggere*/

        return 0;
}

int min(int x,int y){
        if(x<y)
                return x;
        else
```

Esempio: Stringhe IV

```
                return y;
}
```

Esercizio - dollarize.c

Scrivere una procedura che ricevuto un array di caratteri modifichi il vettore in modo che ogni vocale venga sostituita dal simbolo '$'.

```
void dollarize(char arr[], int dim);
```

Esercizo - dollarize.c

```
#include <stdio.h>

void dollarize(char arr[], int dim);

int main()
{
    char arr[11] = {'h','e','l','l','o',' ','w','o','r','l','d'};
    dollarize(arr,11);  printf("%s\n",arr);
}

void dollarize(char arr[], int dim)
{
  int i;
  for (i=0; i<dim; i++)
    switch (arr[i])
    {
      case 'a': case 'e': case 'i': case 'o': case 'u':
      case 'A': case 'E': case 'I': case 'O': case 'C':
        arr[i]='$';
    }

    /* ALTERNATIVA:
    If ((arr[i]=='a')||(arr[i]=='e')||(arr[i]=='i')||(arr[i]=='o')
        ||(arr[i]=='u')||(arr[i]=='A')||(arr[i]=='E')
        ||(arr[i]=='I')||(arr[i]=='O')||(arr[i]=='U'))
      arr[i]='$';
    */
}
```

Esercizio - ricerca_dicotomica.c (1)

La ricerca dicotomica (o ricerca binaria) è un algoritmo che consente di cercare un elemento all'interno di un array ordinato.

L'algoritmo è simile al metodo usato per trovare una parola sul dizionario: sapendo che il vocabolario è ordinato alfabeticamente, l'idea è quella di iniziare la ricerca non dal primo elemento, ma da quello centrale, cioè a metà del dizionario. Si confronta questo elemento con quello cercato:

- se corrisponde, la ricerca termina indicando la posizione in cui l'elemento è stato trovato;
- se è inferiore, la ricerca viene ripetuta sugli elementi precedenti (ovvero sulla prima metà del dizionario), scartando quelli successivi;
- se invece è superiore, la ricerca viene ripetuta sugli elementi successivi (ovvero sulla seconda metà del dizionario), scartando quelli precedenti.

Quando tutti gli elementi sono stati scartati, la ricerca termina indicando che il valore non è stato trovato.

Esercizio - ricerca_dicotomica.c (2)

Scrivere una funzione ricorsiva che implementi l'algoritmo di ricerca dicotomica.

```
int ricercadicotomica(int a[], int n, int sx, int dx)
```

con `n` elemento da cercare in `a`, e `sx` e `dx` indici degli estremi della porzione di array correntemente considerata

La funzione deve restituire la posizione dell'elemento cercato nell'array se presente o `-1` altrimenti

Esercizo - ricerca_dicotomica.c I

```
#include <stdio.h>

int ricercadicotomica(int a[], int n, int sx, int dx);

int main()
{
  int a[] = {1,3,5,7,9};
  int i;
  int res;

  for (i=0; i<10; i++)
  {
    res = ricercadicotomica(a,i,0,4);
    if (res>=0)
      printf("Valore %d presente in posizione %d\n",i,res);
    else
      printf("Valore %d non presente\n",i);
  }
}

int ricercadicotomica(int a[], int n, int sx, int dx)
{
  int mid;

  if (sx==dx)
    if (a[sx]==n) return sx;
    else return -1;
```

Esercizo - ricerca_dicotomica.c II

```
  else
  {
    mid = (sx+dx)/2;
    if (a[mid]==n) return mid;
    else if (a[mid]>n) return ricercadicotomica(a,n,sx,mid);
    else return ricercadicotomica(a,n,mid+1,dx);
  }
}
```

Esercizio: inversione

Scrivere un programma che prenda in input una stringa (lunga al più 20) con `getchar`, la copi invertita in una variabile e stampi tale variabile.

Esercizo - inversione.c

```
#include <stdio.h>

#define INPUT_SIZE 21

int main()
{
  int ch;
  char s1[INPUT_SIZE];
  char s2[INPUT_SIZE];
  int i,j;

  while (((ch=getchar()) != EOF) && (i < INPUT_SIZE-1))
  {
    s1[i] = ch;
    i++;
  }
  s1[i]='\0';

  for (j=0; j<i; j++)
  {
    s2[j] = s1[i-1-j];
  }
  s2[i]='\0';

  printf("\n%s\n",s2);
}
```

Esercizio: scambio

Scrivere una funzione con la seguente interfaccia
`void Scambia(char s[])`, che scambi il primo carattere della stringa s con l'ultimo.

Esercizio: scambio

Scrivere una funzione con la seguente interfaccia
`void Scambia(char s[])`, che scambi il primo carattere della stringa s con l'ultimo.

```
void Scambia(char s[])  {
  int i;
  char c;

  i = strlen(s);

  c = s[0];
  s[0] = s[i-1];
  s[i-1] = c;
}
```

Esercizio: concatenazione

Scrivere una funzione con l'interfaccia: void concatena(char s1[], char s2[]) che funzioni come strcat.