I puntatori e l'allocazione dinamica di memoria

Esercizi risolti

1 Esercizio

Si vuole realizzare un tipo struct, utilizzato per informazioni su operazioni di vendita, avente i seguenti campi:

- codice: numero intero indicante il codice di riferimento dell'articolo venduto
- nome: stringa di lunghezza inferiore a 20 caratteri
- prezzo: numero reale (float) corrispondente al prezzo unitario dell'articolo
- npezzi: numero (intero) di pezzi venduti

Con tale tipo si vogliono realizzare liste dinamiche. La struct deve quindi essere realizzata come struttura ricorsiva, con puntatore a un dato dello stesso tipo.

Si definiscano in C il tipo struct, utilizzando due diversi schemi: (a) una struct a un solo livello, contenente tutti i campi, puntatore ricorsivo compreso; (b) una struct a due livelli, nella quale a primo livello si accede a un puntatore ricorsivo e ad una sotto-struttura contenente il resto dei dati.

Soluzione

Schema (a-1): tipo `operazione_t`, dichiarato mediante `typedef`, contenente tutti i campi.

```
typedef struct operazione
{
   int codice;
   char nome[20];
   float prezzo;
   int npezzi;
   struct operazione *link;
} operazione_t;
```

Schema (a-2): tipo `operazione_p` (puntatore) dichiarato prima di definire la struct (eccezione ammessa in C). Tipo `operazione_t`, dichiarato mediante `typedef`, contenente tutti i campi

```
typedef struct operazione *operazione_p;
typedef struct operazione
{
   int codice;
   char nome[20];
   float prezzo;
   int npezzi;
   operazione_p link;
} operazione_t;
```

Schema (b): tipo `info_t` per i dati, il resto secondo la strategia (a-2)

```
typedef struct
{
   int codice;
   char nome[20];
   float prezzo;
   int npezzi;
} info_t;
typedef struct operazione *operazione_p;
typedef struct operazione
```



```
{
   info_t dati;
   operazione_p link;
} operazione_t;
```

2 Esercizio

Si dichiarino in C 3 variabili reali (**float**), di nome x, y, z. Si dichiarino successivamente 3 variabili puntatore, di nome px, py, pz, da inizializzare (direttamente nella dichiarazione), con i valori dei puntatori alle 3 variabili precedenti.

Soluzione

```
int x, y, z;
int *px=&x, py=&y, pz=&z;
```

3 Esercizio

Si realizzi una funzione di nome `scambiaInt`, che scambi il contenuto di due variabili intere, ricevute come parametro “by reference” (non essendo previsto in C il passaggio “by reference”, si passano “by value” i puntatori. Si faccia un esempio di chiamata della funzione, per scambiare iol contenuto delle variabili intere x e y.

Soluzione

```
void scambiaInt(int *p0, int *p1)
{
  int tmp;

  tmp = *p0;
  *p0 = *p1;
  *p1 = tmp;
};
...
scambiaInt(&x,&y);
```

4 Esercizio

Si scriva una funzione in grado di confrontare due stringhe, ritornando un risultato compatibile con la funzione `strcmp`, salvo il fatto che si devono ignorare le differenze tra caratteri maiuscoli e minuscoli. Si utilizzi una scansione delle stringhe basata su puntatori. Si noti che non è ammesso convertire le stringhe in maiuscolo (o minuscolo), né utilizzare altre stringhe (in cui copiare le stringhe (convertendolo in maiuscolo/minuscolo).

Soluzione

La soluzione proposta utilizza la funzione di libreria `toupper` (dato un carattere, se è alfabetico, ne ritorna la versione maiuscola). Si potrebbe utilizzare, in modo analogo, la `tolower`.

```
void confrontaStr(char *s0, char *s1)
{
  char *p0, *p1;
```

```
  p0 = s0; p1 = s1;
  while (toupper(*p0)!=toupper(*p1) && *p0!='\0' && *p1!='\0')
  {
    p0++; p1++;
  }
  return(toupper(p0)-toupper(p1));
};
```

5 Esercizio

Si scriva una funzione in grado di creare (e ritornare) una stringa data dalla concatenazione di due stringhe, ricevute come parametri. La stringa generata va allocata dinamicamente.

Soluzione

La soluzione proposta utilizza le funzioni di libreria `strcopy` e `strcat`, in grado, rispettivamente, ci copiare una stringa e di appendere una stringa in fondo ad un'altra.

```
char *concatenaStr(char *s0, char *s1)
{
  char *s01;

  s01 = malloc((strlen(s0)+strlen(s1)+1)*sizeof(char));
  strcopy(s01,s0);
  strcat(s01,s1);
  return(s01);
};
```

6 Esercizio

Si scriva una funzione `malloc2d`, in grado di allocare una matrice rettangolare di numeri reali (tipo `float`), le cui dimensioni sono ricevute come parametri. La matrice viene inizializzata azzerando tutte le caselle.

Soluzione

La soluzione proposta alloca un vettore di puntatori a righe, ognuna delle quali viene successivamente allocata e inizializzata.

```
float **malloc2d(int nr, int nc)
{
  float **m;
  int i,j;

  m = malloc(nr*sizeof(float *));
  for (i=0; i<nr; i++) {
    m[i] = malloc(nc*sizeof(float));
    for (i=0; i<nr; i++) {
      m[i][j] = 0.0;
    }
  }

  return(m);
};
```


7 Esercizio

Si ridefinisca la struct dell'esercizio 1, utilizzando per il campo `nome` una stringa dinamica. Si scriva poi una funzione in grado di copiare il contenuto di una struttura in un'altra, duplicando la stringa `nome`. La struttura destinazione viene ricevuta "by reference", mentre la sorgente viene ricevuta "by value".

Soluzione

La soluzione proposta corrisponde allo schema (a-1) dell'esercizio 1. Si utilizza la funzione strdup.

```
typedef struct operazione
{
   int codice;
   char *nome;
   float prezzo;
   int npezzi;
   struct operazione *link;
} operazione_t;
...
void copiaOperazione(operazione_t *dst, operazione_t src)
{
  *dst = src;
  dst->nome = strdup(src.nome);
}
```

8 Esercizio

Utilizzando la struct definita nell'esercizio 7, si scriva una funzione in grado di ritornare (il puntatore a) un duplicato di una struct ricevuta come parametro. La struttura da copiare viene ricevuta "by value".

Soluzione

La soluzione proposta alloca una struttura, vi copia il contenuto della stringa `src`, e duplica la stringa `nome`.

```
operazione_t *duplicaOperazione(operazione_t src)
{
  operazione_t *dst = malloc(sizeof(operazione_t));
  *dst = src;
  dst->nome = strdup(src.nome);
  return (dst);
}
```


9 Esercizio

Si ripeta l'esercizio 8, supponendo che la struttura da copiare venga ricevuta "by reference".

Soluzione

La soluzione proposta alloca una struttura, vi copia il contenuto della stringa `src`, e duplica la stringa `nome`.

```
operazione_t *duplicaOperazione(operazione_t *src)
{
  operazione_t *dst = malloc(sizeof(operazione_t));
  *dst = *src;
  dst->nome = strdup(src->nome);
  return (dst);
}
```

10 Esercizio

Sia data una lista di struct quali quelle utilizzate negli esercizi 7,8,9. Si scriva una funzione che, ricevuto come parametro il puntatore al primo elemento della lista, conti gli elementi in lista e ritorni tale valore come risultato.

Soluzione

La funzione riceve come parametro il puntatore al primo elemento in lista. La variabile locale `p` viene utilizzata per percorrere tutta la lista.

```
int contaOperazioni(operazione_t *header)
{
  int n;
  operazione_t *p;

  for (n=0, p=header; p!=NULL; n++) {
    p = p->link;
  }
  return (n);
}
```

11 Esercizio

Sia data una lista di struct quali quelle utilizzate negli esercizi 7,8,9. Si scriva una funzione che, ricevuto come parametro il puntatore al primo elemento della lista e una stringa di caratteri, cerchi nella lista l'elemento, se esiste, avente nome uguale a quello ricevuto. La funzione ritorna il puntatore alla struct trovata (eventualmente NULL).

Soluzione

La funzione riceve come parametro il puntatore al primo elemento in lista. La variabile locale `p` viene utilizzata per percorrere la lista.

```
int contaOperazioni(operazione_t *header, char *cerca)
{
  operazione_t *p;

  for (p=header; p!=NULL; p=p->link) {
    if (strcmp(p->nome,cerca)==0)
```

```
        return (p);
    }
    return (NULL);
}
```


