1. Esercizi risolti

Esempio 1
Data una lista formata da celle definite come segue:

```
struct cella
{
   int info;
   cella* next;
};
```

la stampa degli elementi della lista può essere utilmente prodotta mediante una funzione ricorsiva:.

```
void StampaListaRicorsiva(cella *lis)
{
   cella* t = lis;
   if (t!=NULL)
   {
      cout<<t->info;
      StampaListaRicorsiva(t->next);      // passo induttivo
   }
}
```

Esempio 2

Supponendo di avere un linguaggio, di tipo non proprio recente, che non preveda il tipo stringa, si può immaginare di implementarlo come struttura astratta. In questo caso, si può implementare il tipo in esame o mediante un array o mediante puntatori. Ciascuna delle implementazioni presenterà, ovviamente, svantaggi e vantaggi che il lettore conosce relativamente ad esse.
Volendo implementare il tipo stringa mediante l'array si potrebbe porre la seguente dichiarazione:

```
const int MAX=10;
typedef char stringa[MAX];
```

e definire alcune operazioni quali ad esempio:

```
int strlen(stringa s);                  // restituisce la lunghezza della stringa
char charAt(stringa s, int p);          // restituisce il carattere in posizione pos
char *subString(stringa s, int from, int to);   // restituisce sottostringa di s tra le posizioni from e to
```

L'implementazione potrebbe essere la seguente:

```
/* Implementazione tipo stringa */
/* Data: 03/11/2008 */
/* File: ImplementazioneStringa.cpp  */
#include <iostream>
using namespace std;
const int MAX=10;
```

```
typedef char stringa[MAX];
int strlen(stringa s);
char charAt(stringa s, int p);
char *subString(stringa s, int from, int to);
int main()
{
   stringa s={"ciao"};
   cout<<strlen(s)<<endl;
   cout<<"charAt(\"ciao\", 3)="<<charAt(s, 3)<<""<<endl;
   cout<<"subString(\"ciao\", 2, 5)=\""<<subString(s, 2,5)<<"\""<<endl;
   system("Pause");
   return EXIT_SUCCESS;
}
int strlen(stringa s)
{
   char *p=&s[0];
   int c=0;
   while((*p)!='\0')
   { c++;
     p++;
   }
   return c;
}
char charAt(stringa s, int p)
{
   if (p>strlen(s) || p<0)
     return '\0';
   else
     return s[p-1];
}

char *subString(stringa s, int from, int to)
{
   char *c, *inizio;
   if (from>strlen(s))
     return '\0';
   else
     {
       c=inizio=&s[from];
       for (int i=from; i<=to; i++)
       {
         *c=s[i];
         c++;
       }
       *c='\0';
       return inizio;
     }
}
```

Esempio 3

Quando in un problema si ha a che fare con numeri molto grandi, i compilatori dei linguaggi attuali hanno potenza e accuratezza tali da effettuare calcoli senza errori di troncamento o arrotondamento.

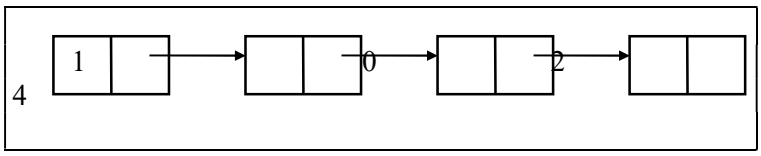
Tuttavia, in alcuni casi, il problema richiede un numero di cifre che va ben al di là della capacità di rappresentazione dei numeri all'interno della memoria del sistema. Allora, il ricorso alle liste diventa indispensabile. Si potrebbe pensare, ad esempio, un numero decimale come una sequenza di cifre, ciascuna associata ad una posizione in base alla potenza di 10 ad essa relativa. In questo modo si possono rappresentare numeri la cui lunghezza supera la capacità di rappresentazione della memoria del sistema. L'implementazione tipica è in questo caso mediante liste e puntatori, per cui la dichiarazione potrebbe essere la seguente:

```
typedef  struct cifra
{
     int c;
     struct cifra *next;
}
```

per cui un numero viene definito come:

```
cifra *numero;
```

Ad esempio, il numero 1024 viene rappresentato con la lista illustrata a fianco.



Le operazioni da implementare, in questo caso, potrebbero essere:

a. Addizione(N1,N2,N3)
b. Sottrazione(n1,N2,N3)
c. Moltiplicazione(N1,N2,N3)
d. Divisione(N1,N2,N3)
e. Potenza(N1,e,N2)

Esempio 4

L'introduzione del tipo puntatore, unitamente al concetto di libreria, ci consente di costruire una libreria di operazioni su una lista di valori di un dato tipo t. Ci proponiamo di costruire una libreria che consenta ad un utente di operare su una lista L. Le operazioni che comunemente questi utilizzerà, supponiamo siano le seguenti:

a. Primo(L), che restituisce il valore del primo elemento di L;
b. Resto(L), che restituisce la lista L privata della testa;
c. Lunghezza(L), che restituisce il numero degli elementi della lista L;
d. Ultimo(L), che restituisce il valore dell'ultimo elemento di L;
e. Menultimo(L), che restituisce la lista L, privata dell'ultimo elemento;
f. Vuota(L), che restituisce True o False, a seconda che L sia vuota o meno;
g. InPrimo(L,x), che inserisce l'elemento x in testa alla lista L;
h. InUltimo(L,x), che inserisce l'elemento x in coda alla lista L;
i. Uguali(L1,L2), che restituisce True o False, a seconda che L1 ed L2 siano uguali o meno.

Esempio 5

Anche l'algebra polinomi può essere con successo implementata tramite liste e puntatori, mediante la seguente dichiarazione:

```
typedef  struct monomio
{
    int coeff;
    int grado;
    struct monomio *next;
```

};

monomio *p1, *p2;

La libreria da costruire su questo tipo astratto, prevedrebbe le seguenti operazioni:
a. Addizione(p1,p2,p3) che costruisce il polinomio p3, somma dei polinomi p1 e p2;
b. Sottrazione(p1,p2,p3), che costruisce il polinomio differenza tra p1 e p2;
c. Moltiplicazione(p1,p2,p3), che costruisce il polinomio prodotto tra p1 e p2;
d. Divisione(p1,p2,p3,p4) che costruisce il polinomio p3 (quoziente tra p1 e p2) ed il polinomio p4 (resto della divisione);
e. Acquisizione di un polinomio;
f. Stampa di un polinomio

Esempio 6

Scrivere un programma che carichi in una lista una serie di valori positivi e calcoli e stampi il numero di valori pari e di quelli dispari (lo zero sia assunto come valore pari). Il problema è di per sé abbastanza semplice e consiste nello scrivere una funzione per il caricamento di valori interi in una lista, fino alla fine dell'input (Ctrl-Z premuto dall'utente). Supponiamo, sempre per semplicità, di trattare con dati di tipo **int**, per cui la definizione del dato *cella* è:

```
typedef struct cella
{
  int info;
  struct cella *next;
} cella;
```

La funzione di inserimento ha prototipo:

```
void inserisci(cella* &p)
```

che riceve per indirizzo il puntatore alla lista. Successivamente, occorre implementare le due funzioni:

```
int contapari (cella *l);
int contadispari(cella *l);
```

che calcolano, rispettivamente il numero di valori pari e dispari presenti nella lista. Forniamo, inoltre, la funzione

```
void  elimina(cella* &l);
```

che dealloca la lista creata, e la funzione
```
void stampa(cella *l);
```

che visualizza gli elementi presenti. A scopo didattico, riportiamo la versione ricorsiva delle funzioni *stampa(), contapari(), contadispari()* ed *elimina()*.

ContaPariDispari.cpp

```
#include <iostream>
#include <cstdlib>
using namespace std;
typedef struct cella                    /* tipo del dato */
{ int info;
```

```
  struct cella *next;
} cella;
void inserisci(cella* &l);                          /* inserimento valori */
void stampa(cella *l);                              /* stampa lista */
void elimina(cella* &l);                            /* eliminazione lista */
int contapari (cella *l);                       /* conta i valori pari */
int contadispari(cella *l);                         /* conta i valori dispari */
int main()
{  cella *p;
   inserisci(p);
   cout<<"La lista contiene "<<contapari(p)<<" valori pari"<<endl;
   cout<<"La lista contiene "<<contadispari(p)<<" valori dispari"<<endl;
   elimina(p);
   stampa(p);
   system("pause");
   return 0;
}

void inserisci(cella* &l)                           /* inserisce elementi letti da
input …*/
{   int x;
    cella *q;
    l=NULL;
    while(cin>>x)                                   /*…fino all'immissione di 0 */
    {  q=new cella;
       (*q).info=x;
       (*q).next=l;
       l=q;
       cin>>x;
    }
}
void stampa(cella *l)                           /* funzione di stampa ricorsiva */
{   cella *q;
     q=l;
     if (q!=NULL)
     {  cout<<(*q).info;
        stampa((*q).next);                      /* richiama sul resto della lista */
     }
}
void elimina (cella* &l)                            /* eliminazione ricorsiva della
lista */
{ cella *q;
   if (l==NULL)
      cout<<"Lista eliminata!"<<endl;
   else
   {   q=l;
       l=l->next;
       delete []q ;
       elimina(l);                              /* richiama sul resto della lista */
   }
}
int contapari(cella *l)
{   cella *q=l;
     if (q==NULL)
        return 0;
```

```
    else
       if ((q->info%2)==0)
            return 1+contapari(q->next);
      else  return contapari((*q).next);
}
int contadispari(cella *l)
{   cella *q=l;
    if (q==NULL)
        return 0;
    else
    if ((q->info%2)!=0)
        return 1+contadispari(q->next);
    else
        return contadispari((*q).next);
}
```

2. Esercizi applicativi

Risolvere i seguenti esercizi

1. Data la dichiarazione seguente:

```
typedef struct nodo
{  char info;
   struct nodo *next;
} cella;
```

determinare l'effetto delle seguenti funzioni,

a.
```
int f1 (cella *l)
   {  return (l==NULL);
   }
```

b.
```
char f2 (cella *l)
   {  if (l!=NULL)
         return l->info;
   }
```

c.
```
void f3 (cella *l, int &p)
   {  p=0;
      while (l!=NULL)
         if (l->inf > 0)   p++;
   }
```

d.
```
cella  *f4 (cella *l)
   {   cella *p;
       if (l!=NULL)   p=l->next;
       return p;
   }
```

2. Data la dichiarazione indicata in Esercizio 1, determinare l'effetto delle seguenti funzioni, dopo aver individuato la presenza di eventuali errori:

a.
```
cella f1 (int n)
   {  cella *tmp= new cella;
      info=n;
      return tmp;
   }
```

b.
```
int f2 (cella *l)
   {  cella *q=l;
      if (q)
          return 0;
      else
          return 1 + f2 (q.next);
   }
```

c.
```
cella *f3 (cella *l)
   {
     cella *q;
     if (l==NULL)     return l;
     else
      if ((l->info%2)==0)
      { q=l;
        l=l->next;
        delete [] q;
        return f3 (l);
      }
      else l->next=f3 (l->next);
}
```

3. Problemi proposti

1. Considerando l'Esempio 2 relativo agli Esercizi risolti, implementare il tipo stringa mediante una lista concatenata, sfruttando la dichiarazione seguente:

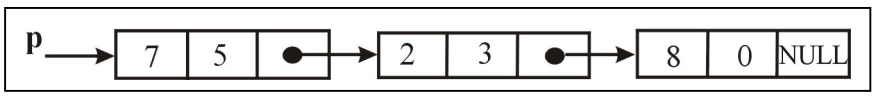
```
typedef struct stringa
{   char *c;
    int  lunghezza;
};
```

ed implementare le seguenti operazioni:
   a. Len(S), che restituisce la lunghezza di S;
   b. Left(S,i) che restituisce la sottostringa di S, formata dai primi i caratteri;
   c. Right(S,k,i) che restituisce la sottostringa di S formata da k caratteri a partire dalla posizione i-esima;
   d. Vuota(S), che restituisce in S la stringa vuota;
   e. Maiuscola(S), che trasforma la stringa S in maiuscolo;
   f. Minuscola(S) che trasforma la stringa S in minuscolo;
   g. Primo(S), che fornisce il primo carattere di S;
   h. Ultimo(S), che fornisce l'ultimo carattere di S;
   i. Conc(S1,S2,S3) che pone in S3 la concatenazione di S1 con S2;
2. Scrivere un'applicazione completa che preveda tutte le operazioni su una lista di interi, in particolare:
   a. instesta (cella* &ptr, int a) /* inserimento in testa */
   b. inscoda (cella* &ptr, int a) /* inserimento in coda */
   c. insordine (cella* &ptr, int a) /* inserimento ordinato */
   d. elimtesta (cella* &ptr) /* eliminazione in testa */
   e. elimcoda (cella* &ptr) /* eliminazione in coda */
   f. elimpos (cella* &ptr, int a) /* eliminazione in data posizione */
3. Scrivere un programma che:
   a. consenta di inserire da tastiera due polinomi (solo coefficienti ed esponenti);
   b. preveda le operazioni di *somma* e *differenza* tra i due polinomi;
   c. *stampi* il polinomio risultante.

   Sugg.: si può usare un array di **struct** oppure, soluzione più efficiente, una lista per rappresentare i soli coefficienti diversi da zero (rappresentazione *sparsa*), per cui il polinomio

$$P(X) = 7X^5 + 2X^3 + 8$$

   può essere rappresentato come:



4. Scrivere un programma che simuli le funzioni di una *rubrica telefonica* di un cellulare, offrendo le funzioni di:
   a. *inserimento* da tastiera di nomi e numeri di telefono;
   b. *visualizzazione* delle informazioni *ordinate* per nome;
   c. ricerca del nome dato il numero;
   d. ricerca del numero dato il nome.