PROGETTO INTERMEDIO DI FONDAMENTI DI INFORMATICA 2020.21

CALCOLO DI EPRESSIONI ALGEBRICHE

Si sviluppi un programma che calcoli il valore di espressioni algebriche del tipo

2.3 – 6.0 * (3.2 – (1.5 / 0.5 * 2.0) * (4.5 * 0.2 – 1.0) + 3.6 / (7.5 / (2.5 – 1.25)) – 1.2) + 9.0

Un'espressione algebrica deve essere rappresentata da una lista dinamica di elementi, ciascuno dei quali può essere un numero, un operatore ('+, '-', '*' oppure '/') oppure una parentesi aperta o una parentesi chiusa:

```
typedef enum {numero, operatore, paraperta, parchiusa} TipoElemento;
typedef struct EL {TipoElemento tipo;
                   float val;
                   char op;
                   struct EL *prox;} Elem;
typedef Elem *EspressioneD;
```

Il programma deve contenere i seguenti sottoprogrammi:

```
float CalcolaEspressioneD (Elem *Iniz, Elem *Fine);
/* Iniz dopo la (, fine sulla ) */
```

che riceve due puntatori Iniz e Fine a elementi di un'espressione senza parentesi (con soli numeri e operatori alternati) e calcola il valore della sottoespressione compresa tra l'elemento puntato da Iniz (**incluso**) e l'elemento puntato da Fine (**escluso**). Scrivere il sottoprogramma basandosi sul codice che implementa la funzione per il caso di Lista Sequenziale. Questo codice è riportato alla fine per comodità.

```
void Sostituisci(EspressioneD ED, Elem *punt);
```

che riceve un'intera espressione e un puntatore a un suo elemento, e prova a sostituire un valore numerico alla sottoespressione che comincia con l'elemento puntato. Il sottoprogramma opera solo nei casi seguenti:

- l'elemento puntato è una parentesi aperta e la successiva parentesi nella lista è una parentesi chiusa: in tal caso il sottoprogramma sostituisce alla sottolista compresa tra le due parentesi (comprese) un solo elemento il cui valore numerico è ottenuto valutando l'espressione senza parentesi inclusa tra le due parentesi mediante una chiamata a `CalcolaEspressioneD`; ad esempio se punt punta alla parentesi aperta in (4.5 * 0.2 – 1.0) ai sette elementi viene sostituito l'elemento -0.1;
- L'elemento puntato è il primo dell'intera espressione ed è di tipo numero, e non vi sono più parentesi nell'espressione: in questo caso l'intera lista viene sostituita da un unico elemento calcolato tramite `CalcolaEspressioneD`

```
float ValutaEspressioneLD(EspressioneD ED);
```

che riceve un'espressione e la valuta avvalendosi di ripetute chiamate alla procedura `Sostituisci`.

N.B. Il sottoprogramma Sostituisci deve deallocare la memoria occupata dagli elementi che vengono cancellati

Cercare di minimizzare gli sforzi per la parte di acquisizione della lista rappresentante un'espressione: tale parte avrebbe bisogno della gestione dei **file**, che verranno studiati in seguito.

Le ripetute chiamate alla procedura Sostiuisci devono produrre le seguenti sostituzioni a partire dall'espressione iniziale

2.3 – 6.0 * (3.2 – (1.5 / 0.5 * 2.0) * (4.5 * 0.2 – 1.0) + 3.6 / (7.5 / (2.5 – 1.25)) – 1.2) + 9.0

1° scansione

2.3 – 6.0 * (3.2 - 6.0 * (4.5 * 0.2 - 1.0) + 3.6 / (7.5 / (2.5 – 1.25)) – 1.2) + 9.0

2.3 – 6.0 * (3.2 - 6.0 * -0.1 + 3.6 / (7.5 / (2.5 – 1.25)) – 1.2) + 9.0

2.3 – 6.0 * (3.2 - 6.0 * -0.1 + 3.6 / (7.5 / 1.25) – 1.2) + 9.0

2° scansione

2.3 – 6.0 * (3.2 - 6.0 * -0.1 + 3.6 / 6.0 – 1.2) + 9.0

3° scansione

2.3 – 6.0 * 23.2 + 9.0

4° scansione (finale)

150.5

SI SUGGERISCE DI ISPIRARSI AI SEGUENTI SOTTOPROGRAMMI, CHE ERANO STATI PENSATI PER ESPRESSIONI RAPPRESENTATE TRAMITE LISTE SEQUENZIALI

```
void    Sostituisci (Espressione *pesp, int i)
{
int j;
float risultato;

if (i >= pesp->N) || (pesp->sequenza[i].tipoelem != Par) ||
    (pesp->sequenza[i].qualepar != aperta) return;
j=i+1;
while (j< pesp->N) && (pesp->sequenza[j].tipoelem != Par)
    j++;
if (pesp->sequenza[j].qualepar != chiusa) return;
risultato = Calcola(pesp->sequenza[i+1].qualenum,
pesp-> sequenza [i+2].qualeop, pesp->sequenza[i+3].qualenum);
/* compatta la lista */
pesp->sequenza[i].tipoelem = Num;
pesp->sequenza[i].qualenum = risultato;
pesp->N = pesp->N - (j-i);
i++;
while (i < pesp->N)
    {
    pesp->sequenza[i] = pesp->sequenza[j+1];
    i++;
    j++;
    }
return;
}
```

```
float   ValutaEspressione (Espressione esp)
{
int i;

while (esp.N > 1)
    {
    i = 0;
    while (i < esp.N)
        {
        Sostituisci (&esp, i)
        i++;
        }
    }
return (esp.sequenza[0].qualenum);
}
```

```
float CalcolaEspressioneLseq (Espressione E)
{
  float risultato;
  float termine;
  float fattore;
  char operdiviso, opiumeno;
  int i = 0;

  fattore = E.seq[i].val;
  termine = fattore;
  while(i+1 < E.N &&(E.seq[i+1].op == '*'||E.seq[i+1].op == '/'))
  {
    i++;
    operdiviso = E.seq[i].op;
    i++;
    fattore = E.seq[i].val;
    if(operdiviso == '*') termine = termine * fattore;
    if(operdiviso == '/') termine = termine / fattore;
  }
  risultato = termine;
```

```
  while(i+1 < E.N && (E.seq[i+1].op == '+' || E.seq[i+1].op == '-'))
  {
    i++;
    opiumeno = E.seq[i].op;
    i++;
    fattore = E.seq[i].val;
    termine = fattore;
    while(i+1 < E.N &&(E.seq[i+1].op == '*' || E.seq[i+1].op == '/'))
    {
      i++;
        operdiviso = E.seq[i].op;
        i++;
      fattore = E.seq[i].val;
        if(operdiviso == '*') termine = termine * fattore;
        if(operdiviso == '/') termine = termine / fattore;
    }
    if(opiumeno == '+') risultato = risultato + termine;
    if(opiumeno == '-') risultato = risultato - termine;
  }
  return risultato;
}
```