Relazione progetto di laboratorio

Elvis Ciotti
Matr. 200244

7 febbraio 2003

Programmazione degli elaboratori
C.d.L. in Informatica applicata

Prof. Marco Bernardo

Fasi dello sviluppo del software secondo la metodologia in the small

Fase 1: Specifica del problema

Una espressione aritmetica semplice in ANSI C è costituita da identificatori di variabili di tipo int o double, costanti numeriche di tipo int o double, i due operatori aritmetici unari, i cinque operatori aritmetici binari (due additivi e tre moltiplicativi), e le parentesi tonde.

Scrivere un programma ANSI C che, acquisita una espressione aritmetica semplice, stampi una espressione aritmetica semplice ottenuta da quella acquisita inserendo delle parentesi che riflettono la precedenza e l'associatività degli operatori.
Il programma deve inoltre stampare il tipo della espressione acquisita (assumere che le variabili che iniziano con una lettera tra a ed l siano di tipo int, le altre di tipo double).

***Fase 2 : A*nalisi del problema**

Input del problema: l'input del problema consiste in una espressione aritmetica semplice in input composta da:

- identificatori di variabili, cioè sequenze di caratteri alfanumerici e sottotratti che non iniziano con un cifra decimale. Sono di tipo int quelle che iniziano con un lettera fra la 'a' e la 'l' (o equivalentemente sono int quelle che iniziano con un sottotratto o con una lettera fra la 'm' e la 'z')
- costanti numeriche (cifre decimali anche con punti per quelle di tipo double)
- parentesi tonde che racchiudono sottoespressioni
- operatori unari + e –
- operatori binari moltiplicativi * / e %
- operatori binari additivi + e –

Nel caso l'utente inserisca errori in input, questi devono essere notificati e l'esecuzione del programma deve terminare poiché un eventuale inserimento delle parentesi sarebbe relativo ad una espressione non corretta e di conseguenza il risultato non può essere corretto.
I tipi di errori possibili sono ampiamente descritti nella fase 3 dello sviluppo software.

Output del problema:

- Inserire delle parentesi riflettenti precedenza e associatività degli operatori nella espressione aritmetica

La precedenza massima dei vari operatori è quella delle parentesi: occorre perciò risolvere (cioè inserire parentesi riflettenti precedenza e associatività degli operatori) prima di tutto le sottoespressioni contenute fra le parentesi, indicare che la sottoespressione è stata risolta e procedere con la prossima fino a risolvere tutta la espressione (con un controllo in input del numero di parentesi).
In ognuna di queste occorre procedere da destra associando (cioè mettendo delle parentesi) prima gli operatori unari alla costante o variabile o sottoespressione risolta immediatamente successiva, poi procedere da sinistra prima con l'associazione degli operatori binari moltiplicativi e infine con l'associazione dei binari moltiplicativi. Per i binari l'associazione avviene inserendo una parentesi aperta prima della costante (o variabile o gruppo precedentemente risolto) immediatamente precedente e inserendone una chiusa dopo la variabile (o costante o gruppo precedentemente risolto) immediatamente successiva all'operatore.

- Calcolare il valore int o double dell'espressione

Il calcolo del tipo int o double dell'espressione si riduce alla ricerca all'interno dell'espressione di variabili double (che iniziano con una lettera fra la m e la z o con un sottotratto), di costante double (cioè che contenga il punto).
Da notare che un identificatore di variabile che iniziano con una cifra decimale sono da considerare errori e non di tipo int.
Se l'espressione contiene almeno un'identificatore di variabile o costante di tipo double, tutta l'espressione è di tipo double.
Il risultato infatti di una operazione binaria o unaria con gli operatori richiesti è di tipo double se e solo se almeno un operando è di tipo double.
Se l'utente inserisce per sbaglio un operatore di modulo % che agisce su almeno un operando (costante o identificatore di variabile) double, le parentesi devono essere inserite ugualmente ma il calcolo del tipo non può essere eseguito poiché in una operazione del tipo *a%b* (dove *a* o b *o* entrambi sono double) non è accettata in linguaggio ANSI C ed da considerare errore.

Fase 3 : Progettazione dell'algoritmo

I passi che il programma esegue (con specifica delle funzioni relative nella implementazione della fase successiva) per risolvere il problema sono i seguenti:

Acquisizione della espressione da file in memoria con controllo dell'input

Dovendo allocare preventivamente memoria sufficiente a contenere l'espressione su cui lavorare, è necessario contare il numero di caratteri dell'espressione in base al quale si può ricavare la quantità di memoria necessaria totale conoscendo la memoria necessaria per un carattere.
Occorre controllare anche che il file sia presente e non vuoto.
La funzione *int n_carat(FILE *file_f)* di occupa di questo restituendo 0 se il file non esiste o se non contiene nessun carattere, altrimenti restituisce il numero di caratteri.
Nel caso la funzione restituisca 0 il programma (nella funzione main) notifica l'errore e termina.

Se il numero di caratteri è invece maggiore o uguale a 1 viene assegnata memoria ai due puntatori (utilizzati come vettori) sui quali viene acquisita e calcolata l'espressione. Se il numero di caratteri acquisito è n, occorre dimensionare il primo vettore (vettore espressione contenente l'espressione) a n+2 caratteri (devono infatti essere anche racchiusa fra parentesi per semplificazione dei calcoli di risoluzione) e il secondo (vettore parentesi) a n+2 interi (deve contenere il numero di parentesi che vanno messe in stampa prima o dopo i caratteri dell'espressione corrispondenti). Il vettore parentesi va inoltre azzerato (nella implementazione in linguaggio ansi c la funzione calloc alloca memoria e la azzera anche). Il programma deve controllare anche che il sistema operativo rilasci la memoria richiesta, altrimenti deve notificarlo e terminare l'esecuzione.

Una volta assegnata la memoria necessaria occorre di riversare il contenuto del file nel vettore espressione (rinchiusa fra parentesi). Gli spazi vanno tralasciati e una espressione del tipo a * b + c viene acquisita come a*b+c.
La funzione *void scansione_file(char *a, FILE *file_f, int n* si occupa di questo.

Controllo dell'input

Tenendo conto della specifica e analisi del problema descritte nelle fasi 1 e 2 occorre controllare che l'espressione acquisita non contenga errori del tipo:

- identificatori di variabili contenenti caratteri diversi da alfanumerici o sottotratti: procedendo da sinistra nel vettore, quando viene trovato una lettera o un sottotratto bisogna analizzare tutti i caratteri fino al prossimo operatore o parentesi chiusa (che indica la fine dell'identificatore) e controllare che questi siano solo caratteri alfanumerici o sottotratti. In caso negativo viene restituito errore e il numero di carattere non valido nell'identiicatore. In questo modo ad esempio una espressione del tipo *pippo&pluto*ciao* è un errore al 6° carattere, infatti dal carattere *p* all'operatore * successivo esiste il carattere & che non è né alfanumerico né un sottotratto.
  Una volta arrivati all'operatore occorre lasciare il contatore dopo l'operatore in modo da continuare la ricerca di eventuali errori.
- Costanti numeriche contenenti caratteri che non sono cifre, o con più di un punto, o che finiscono con il punto: procedendo (analogamente a prima) nel vettore da sinistra occorre cercare una cifra decimale e verificare che entro il prossimo operatore o parentesi chiusa (che indica la fine della costante) ci siano solo cifre decimali o punti. Occorre anche verificare che non ci siano più di due punti e che il punto non sia l'ultimo carattere. Sono quindi considerate errori espressioni del tipo 123a*x, 12.34.55*x, 12.*x
- Operatori moltiplicativi (*, / e %) preceduti da caratteri che non sono alfanumerici, sottotratti o parentesi chiuse: questi casi non sono controllati dal controllo degli identificatori e costanti. Potrebbe infatti esserci un errore di due operatori moltiplicativi accostati, oppure un operatore moltiplicativo preceduto da un + o –.
  Sono quindi rilevati gli errori del tipo (*34), a**b, a+*b, a&*, 12.*, ab.*
  (notare che gli ultimi tre casi sono già segnalati dal controllo costanti e identificatori).
- Parentesi aperte che contengano prima una parentesi chiusa
  Si rilevano errori del tipo *ciao)(espr)* non rilevati dai precedenti controlli, infatti il controllo dell'id. ciao da esito positivo (fino alla parentesi chiusa esistono solo caratteri alfanumerici) e si ferma al carattere '(' che non viene controllato, dal quale viene effettuato questo controllo. Se la espressione contiene una sequenza)(viene segnalata la mancanza dell'operatore fra le sottoespressioni.
- Controllo che i caratteri siano effettivamente di una espressione aritmetica semplice, cioè devono essere alfanumerici, o sottotratti, o parentesi o operatori: segnala errori del tipo pippo*&ciao, infatti

il controllo di pippo lascia il contatore al carattere successivo all'operatore (cioè al carattere &) e non verrebbe controllato neanche dal controllo di ciao. Nel caso invece pippo&*ciao l'errore veniva già rilevato (fra la p e il * esiste infatti un carattere non di un Id.). Anche il punto è un errore perché non può trovarsi all'inizio.

- Controllo che il numero di parentesi aperte corrisponda al numero di parentesi chiuse: Il controllo viene eseguito per semplicità in nuovo ciclo in modo che vengano conteggiate tutte le parentesi. Nel caso la condizione sia verificata ma una parentesi non sia nella posizione giusta [es: *pippo(ciao*2)* l'errore viene rilevato dal controllo identificatori].
- Controllo che non ci siano operatori di modulo su costanti o identificatori di variabili double: In questo caso le parentesi devono essere comunque messe ma il tipo non può essere calcolato e deve essere notificato l'errore. Nella implementazione in linguaggio C questo ultimo controllo viene effettuato a parte nelle sottoespressioni e se viene trovato questo tipo di errore si agisce sulla variabile globale tipo, che in fase di stampa verrà tenuta in considerazione.

Con questi controlli si rilevano tutti gli errori di input, qualsiasi cosa sia presente nel file (notare che se l'ultimo carattere del file è un return, viene catturato ed è considerato errore).
Tutti gli errori vengono notificati e se ne esiste almeno uno il programma termina. Nell'implementazione il controllo dell'input è lasciato interamente alla funzione *int contr_espress(char *a,int n)* il cui valore è condizione di continuazione nella main.

Inserimento delle parentesi nella espressione catturata

Come accennato nella fase 2, occorre risolvere (per risolvere si intende in questo caso inserire le parentesi riflettenti precedenza e associatività degli operatori) **prima le sottoespressioni** (cioè le parti di espressione rinchiuse fra parentesi tonde), poi segnalare quelle che sono state risolte (sostituendo le parentesi tonde con quelle quadre e gli operatori con altri caratteri) e risolvere tutte le altre fino a che non è risolta tutta la espressione (cioè fino a che il primo carattere del vettore espressione è una parentesi tonda aperta, quando verrà risolta l'espressione principale il primo carattere diventerà una parentesi quadra aperta e la risoluzione termina).
Nell'implementazione il compito di risoluzione è effettuato dalla funzione *risolvi_tutto(…)* che cerca le sottoespressioni e le passa alla *sottoespre(…)* che le risolve. La ricerca delle sottoespressioni valide viene effettuata con la collaborazione delle funzione pros_par_val che restituisce la prossima parentesi tonda e il suo indice.

La risoluzione di una sottoespressione è eseguita, nella implementazione, dalla funzione *void risolvi_sottoesp(..*) che richiama a sua volta la *metti_par(…)* (inserisce gli indici parentesi) e la *is_op(…)* (se il carattere passato è operatore).
Il funzionamento è il seguente:

Data una **sottoespressione** da risolvere, ovvero il vettore con l'intervallo di posizioni da considerare e che non contiene parentesi, occorre:

- partire da **destra** e valutare ogni volta se l'operatore contenuto nella posizione corrente è unario. L'operatore è **unario** (in una sottoespressione) se è preceduto da un altro operatore (comprese le parentesi secondo la funzione *int is_op(…)* nella implementazione). Se la sottoespressione inizia con un operatore + o -, questo è unario in quanto il carattere precedente è l'operatore parentesi aperta (rispetterebbe la condizione di unario anche un operatore + o – dopo una parentesi chiusa, ma siccome ci si trova una sottoespressione priva di parentesi e si parte da destra, la condizione di riconoscimento è corretta). Se l'operatore è unario bisogna associarlo con la procedura descritta in seguito.
- Partire da **sinistra** e ricercare gli operatori binari *moltiplicativi* (caratteri *, / o %) e associarli con la procedura descritta in seguito.
- Partire da **sinistra** e ricerca gli operatori binari **additivi** (un operatore è additivo in una sottoespressione se è il carattere + o – e prima non ci sono altri operatori) che devono essere associati con la procedura descritta in seguito
- Inserire delle parentesi **quadre** al posto delle tonde rinchiudenti l'espressione appena risolta a indicare che non deve essere più risolta.
- **Sostituzione temporanea degli operatori** della sottoespressione con caratteri temporanei che alla fine verranno ricostituiti, questo per evitare che vengano nuovamente tenuti in considerazioni dalla risoluzione della espressione madre.

Procedura di associazione (Svolta nella implementazione dalla funzione *metti_par*):
Per maggiore chiarezza si intende operando di un operatore:
una costante, un identificatore o un gruppo già associato da un operatore con precedenza maggiore.
Es: in *a*b+10* l'operando sinistro del + è *a*b* e l'operando destro è *10*
Es: in [a+b+4]*[12] l'operando sinistro del * è [a+b+4] e il destro è [12]

La procedura di associazione di un operatore consiste nel mettere la parentesi tonda aperta prima dell'operando sinistro e una parentesi chiusa dopo l'operando destro.
Mettere una parentesi aperta nel vettore espressione prima della posizione i equivale ad aumentare di 1 l'elemento i-esimo del vettore parentesi.
Mettere una parentesi chiusa nel vettore espressione dopo la posizione i equivale a diminuire di 1 l'elemento i-esimo del vettore parentesi.

Es:

Vett espress	(	+	1	2	*	a	b	*	c	/	d	+	4	)
Vett par	0	5	0	-1	0	0	-1	0	-1	0	-1	0	-1	0

Corrisponde alla espressione (((((+12)*ab)*c)/d)+4)

In fase di stampa saranno tenute in considerazione le convenzioni adottate e combinando il vettore espressione con il vettore parentesi si ottiene l'espressione risolta.

L'occupazione di memoria è circa equivalente all'occupazione effettuata nel case di una soluzione a lista di char o scorrimento di array.
In espressioni infatti del tipo: id1 * id2 * id3 * …. * idn (dove idn è il un identificatore di variabile o costante e * è un operatore moltiplicativo binario o additivo) occorre inserire 2n parentesi.
Con la soluzione di due vettori paralleli adottata, consideriamo che un int occupa in genere il doppio di un char, l 'occupazione di memoria è circa la stessa.

Il riconoscimento dell'inizio dell'operando sinistro (non per gli unari) per l'inserimento della parentesi aperta avviene con una ricerca all'indietro sui vettori e termina inserendo la parentesi quando:

- la somma dei termini del vettore parentesi fino a quel punto è nulla
- la posizione precedente del vettore espressione contiene un operatore o una parentesi quadra

Il riconoscimento dell'operando destro avviene in modo analogo con una ricerca in avanti e con la seconda condizione relativa alla posizione successiva del vettore espressione.

Dunque quando viene trovato un unario nella sottoespressione, viene incrementato il valore corrispondente delle parentesi e diminuito quello al termine dell'operando destro.
Con gli operatori binari l'associazione avviene invece da entrambi i lati.

Es: Supponendo di avere acquisito l'espressione *uno*due+tre*
Avremo i vettori:

Vett espress	(	u	n	o	*	d	u	e	+	t	r	e	)
Vett par	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9		10	11

P0rocedendo con l'associazione degli unari da sx riconosciamo un moltiplicativo nella posizione 4 e associamo ottenendo :

Vett espress	(	u	n	o	*	d	u	e	+	t	r	e	)
Vett par	0	1	0	0	0	0	0	-1	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9		10	11

Infatti l'aumento di vett_par[1] rispetta la condizioni di somma parziale nulla e presenza operatore in vett_espress[0].

Procedendo poi con gli additivi se ne riconosce uno nella posizione 8 e la ricerca all 'indietro va ad aumentare ancora di 1 vett_par[1] perché rispetta ancora una volta le condizioni di somma parziale nulla (+1 -1=0) e presenza operatore in vett_espress[0].

Vett espress	(	u	n	o	*	d	u	e	+	t	r	e	)
Vett par	0	2	0	0	0	0	0	-1	0	0	0	-1	0
	0	1	2	3	4	5	6	7	8	9		10	11

Notare che se non ci fosse stata la condizione di somma parziale sarebbe stato aumentato l 'elemento vett_par[5] poiché vett_espress[4] contiene un operatore.
L'espressione risultante in stampa sarà: ((uno*due)+tre) che è corretta.

Stampa a video i due vettori utilizzati per controllo funzionamento
Non è necessaria ma aiuta a comprendere il funzionamento del programma stampando a video in modo allineato i due vettori.
Nell'implementazione può essere rimossa senza compromettere il funzionamento del programma. Può essere comunque utile al programmatore in caso di revisioni e manutenzione del codice.

Calcolo del tipo dell'espressione e assegnazione del risultato ad una variabile
Il tipo, come già spiegato nella fase 2 si restringe alla ricerca di una combinazione di caratteri :

[	lettera fra m e z
+	lettera fra M e Z
-	sottotratto _
*	
%	

Ovvero se trovi una coppia il cui primo elemento è uno di quelli indicati nella prima colonna e il secondo è uno di quelli indicati nella seconda colonna, l'espressione è di tipo double.
(notare che quando viene effettuato il calcolo del tipo non ci sono più parentesi tonde).
E' di tipo double anche se viene trovata una coppia del tipo *qualsiasi carattere*|*punto .*

Dalle seguenti tabelle si dimostra che se almeno un operando è double, il risultato dell'operazione è double ed essendo un'espressione costituita da operazioni, la presenza di un double implica l'intera espressione double.

+	INT	DOUBLE
INT	INT	DOUBLE
DOUBLE	DOUBLE	DOUBLE

-	INT	DOUBLE
INT	INT	DOUBLE
DOUBLE	DOUBLE	DOUBLE

%	INT	DOUBLE
INT	INT	ERRORE
DOUBLE	ERRORE	ERRORE

*****	INT	DOUBLE
INT	INT	DOUBLE
DOUBLE	DOUBLE	DOUBLE

/	INT	DOUBLE
INT	INT	DOUBLE
DOUBLE	DOUBLE	DOUBLE

(nelle tabelle non sono compresi gli unari che non modificano il tipo e sono quindi irrilevanti al fine del calcolo del tipo)

Nell'implementazione, la funzione calcolo_tipo(…) rispetta esattamente queste condizioni ed effettua un controllo a coppie dall'inizio alla fine dell'array tenendo conto del fatto che l'espressione è compresa fra parentesi e che l'opereatore di AND ha precedenza sulll'OR.

Tutto questo calcolo del tipo è eseguito se non ci sono errori di operatori di modulo in cui un operando sia double. In questo caso il tipo non viene calcolato e viene restituita notifica dell'errore.
Nell'implementazione questo controllo viene effettuato per semplicità in ogni sottoespressione e appena viene trovata un operatore % agente su operando double la variabile globale *tipo* assume il valore -1 e il calcolo del tipo non viene eseguito.

Il riconoscimento di quest'errore è eseguito nell'implementazione alla fine delle funzione che risolve le sottoespressioni e funziona nel seguente modo :
Se viene trovato un operatore di modulo nella sottoespressione:

- cerca all'indietro fino a che non trovi un `operatore +o – oppure una parentesi aperta , e se trovi intanto un punto (cioè esiste una costante double) oppure alla penultima posizione trovi un inizio di id. di var. double, assegna -1 a lla variabile globale *tipo*.
  La ricerca non si ferma agli altri due operatori moltiplicativi perché potrebbe esserci un costante double prima di loro e in tal caso sarebbe di tipo int l'operazione. La ricerca si ferma invece agli operatori + e - perché vengono associati dopo i moltiplicativi e se esiste una costante double prima non è un errore.
  es.
  (zeta%ciao) è un errore
  *(zeta*ciao%ciao)* è comunque un errore, infatti l'operatore * è prima di % e siccome opera su zeta ciao dà valore double complessivamente all'espressione zeta*ciao. Quando poi zeta*ciao va associata al % è un errore perché è ti tipo double. Se ci fosse fermati all'operatore * invece l'errore non sarebbe stato trovato poiché ciao è int.
  (zeta+ciao%ciao) non è un errore, infatti rispettando la precedenza dell'operatore di modulo, viene prima eseguito e ciao%ciao è corretto e di tipo int, poi non importa qual è il valore della variabile o costante prima del + , infatti viene associata dopo.

- Cerca in avanti saltando gli eventuali operatori unari e se trovi subito dopo un carattere che indica l'inizio di id. di var. double oppure trovi un punto entro il prossimo operatore, assegna -1 alla variabile globale *tipo*
  Diversamente da prima, basta fermarsi al successivo operatore, poiché l'associazione dei moltiplicativi è associativa da sinistra.
  es.
  (ciao%ciao*zeta) non è un errore, viene eseguito in fatti prima ciao%ciao (int) e solo successivamente [ciao%ciao]*zeta (int*double=double)

Se l'operatore di modulo non esiste o non agisce su operandi double, il valore del tipo viene lasciato a 0 e i
questo caso il calcolo del tipo viene effettuato.

La scelta dell'uso di una variabile globale nella implementazione deriva da fatto che altrimenti si sarebbe dovuta dichiarare nella main e poi passarla a catena alle funzioni risolvi_tutto, risolvi_sottoepr, calcolo_tip
e stampa_su_file.

Stampa combinata su file dell'espressione con le parentesi combinando i due vettori e stampa del tipo (precedentemente calcolato).
Come già descritto, la stampa si occupa semplicemente di stampare il vettore parentesi (escluse le quadre)
e le parentesi indicate nel vettore parentesi con un ciclo in n che parte dal primo carattere all'ultimo e dove

- se n-esimo elemento del vettore parentesi è nullo stampa il carattere corrispondente del vettore espressione (a meno che non sia una quadra)
- se n-esimo elemento del vettore parentesi è positivo stampa tante parentesi aperte quanto indica l'elemento e poi stampa il carattere corrispondente del vettore espressione
- se n-esimo elemento del vettore parentesi è negativo poi stampa il carattere corrispondente del vettore espressione e poi stampa tante parentesi chiuse quanto indica il modulo dell'elemento del vettore parentesi.

Dopo la stampa dell'espressione viene eseguita anche la stampa del tipo.
Nell'implementazione avviene in funzione del valore della variabile globale tipo:
Se tipo=**-1** stampa che l'espressione contiene almeno un errore di operatore di modulo su operandi double (rilevati dalla funzione risolvi_sottoespres),
Se tipo= **0** stampa che l'espressione è di tipo INT (la var tipo non viene mai modificata),
Se tipo= **1** stampa che l'espressione è di tipo DOUBLE (la funzione calcolo_tipo ha trovato un operando double),

Es.
Inserendo l'espressione ++12*(4.5+pippo*pluto)/3

Nei due vettori diventa :

Vett espress	[	+	+	1	2	*	[	4	,	5	+	p	i	p	p	o	*	p	l	u	t	o	]	/	3	]
Vett par	0	3	1	0	#	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	#	0	0	#	0

In stampa produce il risultato (((+(+12))*(4.5+(pippo*pluto)))/3) e scrive che è di tipo double.
L'espressione così stampata è quella inserita con le parentesi riflettenti precedenza e associatività degli operatori. Il tipo è double poiché ci sono 3 costanti double

Fase 4 : Implementazione dell'algoritmo in linguaggio ANSI C

Contenuto del file sorgente prog.c:

```
/* Programma che, acquisita una espressione aritmetica semplice da file, inserisce le
parentesi riflettenti
    la precedenza e la associatività degli operatori (i due unari, i tre moltiplicativi e
i tre additivi ),
    tenendo conto anche delle eventuali parentesi inserite.
    Il programma calcola inoltre il tipo dell'espressione acquisita, assumendo che le
costanti che variabili
    con una lettera tra a ed l sono di tipo int e le altre di tipo double).

    Studente: Elvis Ciotti (Matr. 200244)
    Prof: Marco Bernardo
    C.d.l in informatica applicata - Programmazione degli elaboratori
    7/02/2002    */


#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#define FILE_INPUT "input.dat"    /* Nome del file di input*/
#define FILE_OUTPUT "output.dat"  /* Nome del file di output */


int tipo = 0;  /* tiene in memoria il tipo dell'espressione  */


int n_carat(FILE *file_f);
void scansione_file(char *a, FILE *file_f);
int contr_espress(char *a,int n);
void risolvi_tutto(char *a, int *b, int n);
char pros_par_val(char *a, int part, int *fine);
void risolvi_sottoesp(char *a, int *b, int part, int fine);
void metti_par(char *a, int *b, int indice_op, int prima, int dopo);
int is_op(char c);
int is_molt(char c);
int is_ci_double(char c);
void stampa_vett(char *a, int *b, int n); /* facoltativa */
void calcola_tipo(char *a, int n);
void stampa_su_file(char *a, int *b, int n, FILE *output_f);


int main(void)
{
  FILE *input_f = NULL;
  FILE *output_f = NULL;
  int n = n_carat(input_f);    /* contiene n.di caratteri nel file */
  char *vet_espr;    /* vettore con espressione */
  int *vet_par;      /* vettore con i numeri di parentesi */

  if (n) /* se il file esiste e non è vuoto */
  {
      /* allocamento memoria ai vettori di lavoro */
     vet_espr = (char *) calloc((size_t)(n + 2), sizeof(char)); /* allocamento memoria */
     vet_par = (int *) calloc((size_t)(n + 2), sizeof(int));
     if(vet_espr != NULL && vet_par != NULL)  /* controllo disponibilita memoria */
     {
          /* acquisizione espressione sul vettore espressione */
       scansione_file(vet_espr, input_f);
          /* controllo espressione, in caso di errori si occupa la funzione di stamparne
il tipo */
       if (contr_espress(vet_espr, n))
         {
             /* inserimento indici di parentesi in vettore parentesi */
          risolvi_tutto(vet_espr, vet_par, n);
             /* Stampa a video facoltativa dei due vettori per controllo */
            stampa_vett(vet_espr, vet_par, n);
          calcola_tipo(vet_espr, n);
             /* stampa su file dell'espressione con le parentesi */
          stampa_su_file(vet_espr, vet_par, n, output_f);
         }
```

```
        free(vet_espr); /* disalloca memoria utilizzata */
        free(vet_par);
     }
     else
       printf("Memoria non disponibile.");
  }
  else
    printf("ERRORE: file di input %s inesistente o vuoto.\n", FILE_INPUT);

  return (0);
}


 /* restituisce 0 se il file non esiste o non contiene nessun carattere,
    altrimenti restituisce il numero di caratteri presenti tralasciando gli spazi */
int n_carat(FILE *file_f)
{
   int n = 0;
   char tmp;

   file_f = fopen(FILE_INPUT, "r");
   if (file_f != NULL) /* se il file esiste */
   {
      while(fscanf(file_f, "%c", &tmp) != EOF) /* conta i caratteri tralasciando spazi e
\n*/
           if (tmp != ' ' && tmp != '\n')
       n++;
      fclose(file_f); /* chiude il file per permettere alla funzione successiva di
leggerlo dall'inizio */
   }
   return n;   /* ritorna numero di caratteri */
}


 /* riversa contenuto del file nel vettore passato tralasciando gli spazi */
void scansione_file(char *a, FILE *file_f)
{
  int i = 1;
  char tmp; /* variabile char temporanea */

  a[0] = '(';     /* inserisce parentesi di apertura */

   /* apertura file, il controllo della sua esistenza e gia stato eseguito dalla funzione
precedente */
  file_f = fopen(FILE_INPUT, "r");

   /* scansione contenuto file sul vettore tralasciando gli spazi e\n */
  while(fscanf(file_f, "%c", &tmp) != EOF)
    if (tmp != ' ' && tmp != '\n')
    {
       a[i] = tmp;
       i++;
    }
  a[i] = ')';    /* inserisce parentesi di chiusura */
  fclose(file_f);
}


 /* valuta se il vettore fino alla n-esima posizione contiene una espressione corretta */
int contr_espress(char *a, int n)
{
  int i = 1,
      ret = 1, /* variabile di ritorno (vale 1 se non ci sono errori) */
      n_ap = 0,
      n_ch = 0,
      n_punti = 0;

  while (i < n + 2)
  {
      /* controllo moltiplicativo e par.chiusa che siano precedute da solo da alnum,
sottotratti o altre par.chiuse */
     if (is_molt(a[i]) || a[i] == ')')
     {
      if (!isalnum(a[i - 1]) && a[i - 1] != '_' && a[i - 1] != ')')
      {
         printf("Errore al carattere %d\n", i);
         ret = 0;
```

```
      }
      i++;   /* rilascia contatore alla posizione seguente per controlli successivi */
     }
      /* controllo par. aperte precedute da altre par.chiuse */
     if (i < n + 2 && a[i] == '(')
     {
       if (a[i - 1] == ')')
       {
        printf("Errore al carattere %d, manca operatore\n", i);
        ret = 0;
        i++; /*  rilascia contatore alla posizione seguente */
       }
       i++;   /* rilascia contatore alla posizione seguente */
     }
      /* controllo identificatori di variabili */
     if (i < n + 2 && (isalpha(a[i]) || a[i] == '_')) /* se trovi inizio di identificator
*/
     {
      i++;
       /* finchè non arrivi alla fine dell'identificatore (cioe ad un operatore che non
sia parentesi aperta ) */
      while (!is_op(a[i]) || a[i] == '('   /* applicazione legge de Morgan utilizzando
funzione is_op */
       {
          if (!isalnum(a[i]) && a[i] != '_') /* se trovi carattere non valido per un
identificatore */
          {
             printf("Errore al carattere %d non valido per un'identificatore di
variabile\n", i);
             ret = 0;
          }
          i++; /* controlla il carattere successivo */
       }
         i++; /* rilascia contatore alla posizione seguente l'operatore*/
     }
      /* controllo costante numerica */
     if (i < n + 2 && isdigit(a[i]))
     {
      i++;      /* parti dal carattere successivo */
       /* controlla i caratteri successivi prima del prossimo operatore */
      while (!is_op(a[i]) || a[i] == '(') /* condizione usata anche per gli id. di var. *
      {
          if (!isdigit(a[i]) && a[i] != '.') /* se trovi carattere non valido per costante
numerica */
          {
             printf("Errore al carattere %d non valido per una costante numerica\n", i);
             ret = 0;
          }
          if (a[i] == '.')
             n_punti++;  /* incrementa n. di punti trovati */
          i++;
       }
       if (n_punti > 1) /* se hai trovato piu di un punto nella stessa costante */
       {
          printf("Errore: una costante numerica non deve contenere %d punti\n", n_punti);
          ret = 0;
          n_punti = 0; /* riazzera per controllo eventuali costanti numeriche nella
espressione */
       }
       if (a[i-1] == '.') /* controllo eventuale punto alla fine della costante */
       {
         printf("Errore al carattere %d: una costante non puo terminare con il punto\n",
i);
         ret = 0;
       }
       i++;   /* rilascia contatore alla posizione seguente l'operatore */
     }
      /* in caso trovi + e - incrementa lo stesso il contatore anche se non occorre
       eseguire alcun controllo (qualsiasi carattere di una espr. e valido prima di
questi) */
     if ((i < n + 2 && a[i] == '+') || a[i] == '-')
       i++;
      /* controllo caratteri singoli non accettabili in una espressione */
     if (i < n + 2 && !is_op(a[i]) && !isalnum(a[i]) && a[i] != '_')
     {
       printf("Errore al carattere %d non valido\n", i);
       ret = 0;
```

```
      i++; /* rilascia contatore alla posizione seguente */
    }
  }   /* fine while */

   /* Conteggio par chiuse e par aperte*/
   for (i = 0;
      ( i < n + 2);
      i++)
   {
      if (a[i] == '(')
       n_ap++;
      if (a[i] == ')')
       n_ch++;
   }
   /* controllo n.par */
  if  (n_ap != n_ch)
  {
    printf("Errore: %d parentesi aperte e %d parentesi chiuse.\n", n_ap, n_ch);
    ret = 0;
  }

  if (ret == 0) /* notifica dopo la stampa degli errori */
    printf(" Il file di output non é stato creato.\n");
  return ret;
}


 /* inserisce gli indici parentesi su tutto il vettore */
void risolvi_tutto(char *a, int *b, int n)
{
  int i,
      k; /* viene passato per indirizzo a pros_par_val e assume come valore la posizione
della parentesi terminante la sottoespressione valida */

  while (a[0] == '(')       /* fino a che la espressione radice non è risolta */
    for (i = 0; (i <= n); i++)       /* ciclo per tutti gli elementi del vettore */
     if (a[i] == '(' && pros_par_val(a, i + 1, &k) == ')')    /* se trovi sottoespressior
*/
      risolvi_sottoesp(a, b, i, k);  /* risolvi sottoespressione dalla i-esima alla k-
esima posizione */

   /* annullamento della sostituzione temporanea effettuata dalla funzione
risolvi_sottoesp */
  for(i = 1; (i < n); i++)
    switch(a[i])
    {
     case 15:
       a[i] = '+';
       break;
     case 16:
       a[i] = '-';
       break;
     case 17:
       a[i] = '*';
       break;
     case 18:
       a[i] = '/';
       break;
     case 19:
       a[i] = '%';
       break;
    }
}


 /* Restituisce la prossima parentesi (chiusa o aperta) trovata nel vettore dalla part-
esima posizione.
    Nel terzo parametro viene collocato per indirizzo la posizione della parentesi trovat
*/
char pros_par_val(char *a, int part, int *fine)
{

  while (a[part] != ')' && a[part] != '(')
    part++;
  *fine = part;    /* restituisce anche per indirizzo la posizione della parentesi trovat
nel vettore */
  return a[part];  /* ritorna il carattere par.chiusa o aperta */
```

```
}


 /* risolve la sottoespressione dalla part-esima alla fine-esima riga
    collocando nel vettore b gli indici di parentesi relativi all'espressione del vettore
a*/
void risolvi_sottoesp(char *a, int *b, int part, int fine)
{
  int i,
      k;
  /* inserimento indici parentesi per op. unari con ciclo da destra a sinistra */
  for (i=fine-1; (i > part); i--)
    if ((a[i] == '+'|| a[i] == '-') && is_op(a[i-1]))  /* se la i-esima posizione contiene
un operatore unario */
    {
      b[i]++;   /* incrementa indice parentesi corrispondende all'unario */
      metti_par(a, b, i, 0, 1); /* inserisce parentesi di chiusura relative all'operatore
nella posizione i */
    }
    /* inserimento indici parentesi per op. moltiplicativi */
  for (i = part + 1; (i < fine); i++)
    if (is_molt(a[i])) /* se la i-esima posizione contiene un operatore moltiplicativo */
      metti_par(a, b, i, 1, 1); /* inserisce parentesi prima e dopo relative all'operator
nella posizione i*/

   /* inserimento indici parentesi per op. additivi */
  for (i = part + 1; (i < fine); i++)
    if ((a[i] == '+' || a[i] == '-') && !is_op(a[i - 1]))   /* se la i-esima posizione
contiene un operatore additivo */
      metti_par(a, b, i, 1, 1);  /* inserisce parentesi prima e dopo relative
all'operatore nella posizione i*/

   /* ricerca errori di costanti o id.var di tipo double agenti su operatore di modulo */
   for (i = part + 1; (i < fine); i++)
   {
      if (tipo != -1 && a[i] == '%') /* se non hai già trovato errore e se trovi operator
di modulo */
      {
         /* ricerca costanti e id.var. double all'indietro prima di %*/
       for (k = i - 1;
            (a[k] != '(' && a[k] != '+' && a[k] != '-'); /* fino a che non arrivi
all'inizio dell'espressione o a op. additivo */
            k--)
            /* se trovi punto oppure un id. double */
          if (a[k] == '.' || (is_ci_double(a[k]) && is_op(a[k-1])))
             tipo = -1;

         /* ricerca costanti e id.var. double in avanti dopo il % */
       for (k = i + 1;     /* salto eventuali unari sulla costante successiva al % */
            (a[k] == '+' || a[k] == '-');
               k++);
       if (is_ci_double(a[k]))   /* se trovi identificatore double subito dopo gli unari
*/
          tipo = - 1;
       for (k = k + 1;   /* ricerca costanti double entro il prossimo operatore */
              (!is_op(a[k]));
               k++)
         if (a[k] == '.')
           tipo = -1;
      }
   }

   /* sostituzione parentesi quadre con tonde a indicare la sottoespressione compresa
risolta */
  a[part] = '[';
  a[fine] = ']';

   /* sostituzione temporanea degli operatori in modo da non essere
      nuovamene tenuti in considerazione quando si risolve l'espressione madre */
  for (i = part + 1;
       ( i < fine);
       i++)
    switch(a[i])
    {
      case '+':
      a[i] = 15;
       break;
```

```
      case '-':
       a[i] = 16;
       break;
      case '*':
       a[i] = 17;
       break;
      case '/':
       a[i] = 18;
       break;
      case '%':
       a[i] = 19;
       break;
    }
}


 /* mette nel vettore b i numeri di parentesi corrispondenti all'espressione contenuta ne
vettore a
    e relativi all'operatore nella indice_op-esima posizione.
    Se prima (succ) e diverso da 0 allora viene messo in vettore b l'indice parentesi
prima (dopo) del precedente (successivo) operatore del
    vettore a (gli intervalli di parentesi risolti vengono saltati sfruttando la somma
parziale degli indici parentesi intermedi nel vettore b).*/

void metti_par(char *a, int *b, int indice_op, int prima, int dopo)
{
  int i,
      somma,   /* contatore somma parziale indici utilizzata in vettore */
      cond;    /* variabile sentinella */

  /* inserimento indice parentesi aperta prima (se l'operatore era unario non viene
eseguita) */
  if (prima)
  {
     /* ricerca all'indietro del precedente operatore in vettore a */
    for (i = indice_op-1, somma = 0, cond = 1 ;
       (cond);
       i--)
       {
           somma += b[i];     /* aggiornamento somma parziale degli indici */
            /* se la somma parziale vale 0 oppure esiste un'operatore nella
             posizione precedente del vettore espressione corrispondente */
           if (somma == 0 && (is_op(a[i - 1]) || a[i - 1] == '['))
           {
              b[i]++;  /* incremento indice parentesi dell'elemento in vettore b */
            cond = 0;  /* esce dal ciclo - parentesi messa */
           }
       }
  }

  /* inserimento indice parentesi chiusa dopo */
  if (dopo)
  {
      /* ricerca in avanti del successivo operatore in vettore a */
     for (i = indice_op + 1, somma = 0, cond = 1;
          (cond);
          i++)
        {
         somma += b[i];       /* aggiornamento somma parziale degli indici */
           /* se la somma parziale vale 0  oppure esiste un'operatore nella
              posizione successiva del vettore espressione a corrispondente */
         if (somma == 0 &&  (is_op(a[i + 1]) || a[i + 1] == ']'))
         {
            b[i]--;   /* diminuisci indice parentesi dell'elemento in vettore b */
            cond = 0;   /* esce dal ciclo - parentesi messa */
           }
        }
  }

}


 /* restuisce vero se il carattere è un operatore o una parentesi */
int is_op(char c)
{
  return (c == '+' || c == '-' || c == '*' || c == '/' || c == '%' || c == '(' || c ==
')');
```

```
}

 /* ritorna vero se il char passato è operatore moltiplicativo */
int is_molt(char c)
{
  return (c == '*' || c == '/' || c == '%');
}


 /* ritorna vero se il char passato puo essere inizio di id. di var. double,
    ovvero se il carattere è compreso fra la m e la z oppure e un sottotratto */
int is_ci_double(char c)
{
  return ((c >= 'm' && c <= 'z') || (c >= 'M' && c <= 'Z') || c == '_');
}


 /* stampa a video i due vettori allineati per migliore comprensione dell'algoritmo di
risoluzione.
    Puo essere rimossa senza compromettere il funzionamento dle programma.*/

void stampa_vett(char *vet_espr, int *vet_par, int n)
{
  int i;

  printf("Vettore espres : ");
  for (i = 0; (i < n + 2); i++)
    printf("%2c", vet_espr[i]);

  printf("\nVettore parent : ");
  for (i = 0; (i < n + 2); i++)
    printf("%2d", vet_par[i]);
  printf("\n");
}


 /* Restituisce 0 o 1 a seconda che l'espressione sia rispettivamente int o double.
    Per essere double basta che esista un identificatore di costante double di lettere
dell'alfabeto (fra m e z) o numerica (contenente il carattere '.'), sempre che non si
siano rilevati errori di operatore di modulo su operandi double */
void calcola_tipo(char *a, int n)
{
  int i;
  if (tipo != -1) /* esegue calcolo tipo solo se non ci sono operatori di modulo su
costanti o id. double */
  {
     /* ricerca costanti double */
    for (i = 1;
       ( i < n + 1);
       i++)
       if (((a[i] == '[' || a[i] == '+' || a[i] == '-' || a[i] == '*' || a[i] == '/') &&
is_ci_double(a[i + 1])) || a[i + 1] == '.')
        tipo = 1;  /* se viene trovata costante double la variabile globale tipo diventa
*/
  }
}


 /* Stampa su file l'espressione con le parentesi e il tipo */
void stampa_su_file(char *a, int *b, int n, FILE *output_f)
{
  int i;

  output_f = fopen(FILE_OUTPUT, "w");
  fprintf(output_f, "L'espressione contenuta nel file %s con le parentesi
riflettenti\nprecedenza e associatività degli operatori è la seguente:\n", FILE_INPUT);

  for (i = 1;
       (i <= n);
       i++) /* stampa combinata dell'espressione con parentesi */
  {
     if (b[i] == 0 && a[i] != '[' && a[i] != ']') /* stampa caratteri escluse le par.
quadre*/
      fprintf(output_f, "%c", a[i]);
     if (b[i] > 0)  /* indice vettore parentesi  con numero positivo di parentesi (aperte
*/
     {
```

```
        /* stampa tante parentesi aperte quanto indica l'i-esimo elemento del vettore
parentesi */
      while(b[i] > 0)
      {
         fprintf(output_f, "(");  /* stampa tonde aperte */
         b[i]--;
      }
      fprintf(output_f, "%c", a[i]);  /* stampa il carattere i-esimo della espress. */
     }
     if (b[i] < 0) /* indice vettore parentesi con numero di negativo parentesi (chiuse)
*/
     {
      fprintf(output_f, "%c", a[i]);   /* stampa carattere i-esimo */
         /* stampa tante parentesi aperte chiuse quanto indica i-esimo elemento del
vettore parentesi */
      while (b[i] < 0)
      {
         fprintf(output_f, ")");
         b[i]++;
      }
     }
  }

  switch (tipo) /* stampa del tipo a seconda del valore contenuto nella var globale */
  {
     case 0:
       fprintf(output_f, "\nEspressione di tipo INT.\n");
       break;
     case 1:
       fprintf(output_f, "\nEspressione di tipo DOUBLE.\n");
       break;
     case -1:
       fprintf(output_f, "\nErrore, l'operatore di modulo agisce su operandi double, tipo
della espressione non determinabile.\n");
       break;
  }

  fclose(output_f);
  printf("Espressione risolta correttamente nel file %s.\n", FILE_OUTPUT);
}
/* FINE PROGRAMMA */
```

Fase 5 : Testing del programma

La fase di testing si articola con l'input e output di espressioni di diverso genere anche errate per la verifica del controllo input e correttezza dell'algoritmo di soluzione al problema.

INPUT	OUTPUT	TIPO*
File non esiste	Errore, file non esiste o è vuoto	
File vuoto	Errore, file non esiste o è vuoto	
Cia&o*pluto	Errore carattere 3 non valido per id.	
10 a*34	Errore carattere 3 non valido per cost.	
2ciao+2	Errore carattere 2,3,4,5 on validi per cost.	
(45)(23)	Errore carattere 5, manca operatore	
12.34.4	Errore, costante con 2 punti	
122 3.*45	Errore, costante terminante con punto	
ciao*$due	Errore carattere 6, non valido	
.	Errore carattere 1, non valido	
a * .2	Errore carattere 3, non valido	
(ciao)*(pippo))	Errore, 2 par. aperte e 3 chiuse	
ciao%z	(ciao%z)	ERRORE %
ciao%_Z ia	(ciao%_Zia)	ERRORE %
ciao%+-+zia	(Ciao%(+(-(+zia))))	ERRORE %
ciao%12.34	(ciao%12.34)	ERRORE %
cia o%+-+12.34	(ciao%(+(-(+12.34))))	ERRORE %
zia%ciao	(zia%ciao)	ERRORE %
12.23 % ciA34o	(12.23%ciaA34o)	ERRORE %
_ciao%ciao	(_ciao%ciao)	ERRORE %
9.2 * 12 % ciao	((9.2*12)%ciao)	ERRORE %
_ciao*24/67 % abc	(((_ciao*24)/67)%abc)	ERRORE %
_ciaO+--+2.4/67 % Abc	(_ciaO+(((-(-(+2.4)))/67)%Abc))	ERRORE %
_ciao+--+24/67 % abc	(_ciao+(((-(-(+24)))/67)%abc))	INT
++(46*((12.34%_ciao)+84*25))	(+(+(46*((12.34%_ciao)+(84*25)))))	ERRORE %
a	a	INT
s _ g+ 2. 00 08	(s_g+2.0008)	DOUBLE
2. 4	12.4	DOUBLE
12.3-12%3/88.8	(12.3-((12%3)/88.8))	DOUBLE
((((((((((((1+a))))))))))))))	(1+a)	INT
a+z_s	a+z_s	DOUBLE
10+34.5+90	((10+34.5)+90)	DOUBLE
E45r+34*56/67+2	((E45r+((34*56)/67))+2)	INT
-+uno++-2++54	(((-(+uno))+(+(-2)))+(+54))	DOUBLE
++ciao*+pippo--44.5 4*3	(((+(+ciao))*(+pippo))-((-44.54)*3))	DOUBLE
a+(ciao4 5+56*83/1*Rt)++23	(((-a)+(ciao45+(((56*83)/1)*Rt)))+(+23))	DOUBLE
+-((((48))))/((856)+2)	((+(-48))/(856+2))	INT
a*+ +1/((tr_e)-+(otto))	((a*(+(+1)))/(tr_e-(+otto)))	DOUBLE
++ ++++4*--------8	((+(+(+(+(+(+4))))))*(-(-(-(-(-(-(-(-8))))))))	INT
---++(78*34)	(-(-(-(+(+(78*34))))))	INT

* = ERRORE % significa che è stato rilevato almeno un operatore di modulo agente su almeno un operando double

I test effettuati ricoprono vari casi di input con i vari operatori in combinazione con le parentesi e con errori in input.
Tutti i tipi di errori di input sono stati rilevati correttamente.
In caso di input corretto le parentesi sono state messe in modo adeguato (notare che le parentes in input non necessarie vengono levate e gli spazi vengono tralasciati).