

Linguaggi e Traduttori

a.a. 2005/2006

Tesina n° 5

Titolo: Easy C

Descrizione

Si intende realizzare un traduttore che permetta di semplificare la stesura di programmi C e permetta di fare il debugging di funzioni C.

Dato un file pseudo-C (un C in cui sono aggiunti comandi nuovi non presenti nella sintassi C) in ingresso, si deve produrre un file aderente alla sintassi ansi C.

In primo luogo il traduttore deve permettere alcune semplificazioni nella stesura di programmi C che verrà chiamato “C semplificato” le cui specifiche saranno di seguito riportate.

C semplificato:

- Nel C standard le variabili devono essere dichiarate all’inizio delle funzioni. Nella versione semplificata le variabili possono essere dichiarate in qualsiasi punto del codice. Sarà poi compito del traduttore riscriverle all’inizio delle funzioni nel file di output
- La funzione printf del C è spesso lunga da scrivere e bisogna conoscere il tipo di variabile per stamparla. Si vuole avere una versione semplificata della funzione printf di stampa che verrà chiamata PRINT. La funzione PRINT avrà la stessa sintassi della funzione System.out.print di Java, unica differenza che al posto del carattere “+” per separare stringhe da variabili si vuole utilizzare il carattere “,”.

Es. PRINT(“Valore variabile a : “,a,”\n”);

Ipotizzando che la variabile a sia di tipo intero, l’istruzione PRINT nel file di output dovrà essere scritta nel modo seguente:

```
printf(“Valore variabile a : %d \n”,a);
```

Non si richiede la verifica che nel file di ingresso sia presente la direttiva `#include <stdio.h>`

La realizzazione della funzione PRINT è opzionale.

- Per indicare i caratteri di inizio e fine blocco, rispettivamente “{” e “}” si possono utilizzare i caratteri “[” e “]”

Esiste poi la keyword debug, studiata per facilitare la stesura di nuove funzioni e il loro debugging che consiste, dati diversi ingressi ad una funzione, nel controllarne il suo comportamento. Il codice della direttiva debug è delimitata da un blocco tramite i caratteri “{” e “}” e in coerenza con quanto definito prima tramite i caratteri “[” e “]”. All’interno della keyword debug valgono esattamente le stesse regole del “C semplificato”.

`debug {/* codice */}` o `debug [[/* codice */]]` sono equivalenti.

All’interno della keyword debug in ordine ci sono:

- In esclusione l’istruzione `main;` o l’istruzione `fz <nome_fz>;`. Il loro significato verrà spiegato più avanti nel testo
- Una funzione in normale sintassi “C semplificata”

Es:

```
float fz(float a, float b) {/* codice */}
```

- La keyword `data {/* codice */}`

Keyword data:

Permette di fornire una serie di ingressi alla funzione, al fine di vedere il comportamento di quest’ultima con input diversi. La sintassi delle keyword è la seguente:

<etichetta> { /* codice assegnamento */ } o <etichetta> [[/* codice assegnamento */]]
 <etichetta> è un identificatore. /* codice assegnamento */ è il codice che governa gli ingressi alla funzione da debuggare.

Se ad esempio è stato scritto il codice della seguente funzione:

```
float fz(float a, float b) { /* codice */ }
```

potrà essere scritta la seguente direttiva *data*:

```
data {
  0 {a=0.10; b=0.10; }
  1 {a=-0.10; b=0.10; }
  2 {a=0.10; b=-0.10; }
  3 {a=-0.10; b=-0.10; }
}
```

Essa lancerà per 4 volte la funzione con gli ingressi specificati.

Esiste poi la possibilità di definire dei costrutti FOR annidati che permettono di generare automaticamente i pattern di test per la funzione:

```
FOR x = 0:1:0.5
```

```
FOR y = 0:1:0.5
```

```
0 {a=x; b=y; }
```

Il codice scritto sopra è riscritto come:

```
0:0:0 {a=0; b=0; }
```

```
0:0:0.5 {a=0; b=0.5; }
```

```
0:0:1 {a=0; b=1; }
```

```
0:0.5:0 {a=0; b=0.5; }
```

```
0:0.5:0.5 {a=0.5; b=0.5; }
```

...

Il costrutto FOR avrà perciò la seguente sintassi:

```
FOR <var> = <numero_iniziale>:<numero_finale>:step
```

Rimane ancora aperto lo studio del comportamento della delle due keyword *main*; o *fz* <nome_fz>;

Quando viene definita una keyword *data* dovrà essere inserito il seguente codice in qualche parte del programma:

```
float a;
float b;
printf("0:\n");
a=0.10; b=0.10;
fz(a, b);
printf("1:\n");
a=-0.10; b=0.10;
fz(a, b);
printf("2:\n");
a=0.10; b=-0.10;
fz(a, b);
printf("3:\n");
a=-0.10; b=-0.10;
fz(a, b);
```

Nel caso che all'interno della keyword *debug* sia presente il comando *main*; verrà generata un'istruzione `void main(void) { /* Codice scritto sopra */ }`, nel caso invece si presente il comando *fz* <nome_fz>; verrà generata la seguente funzione `void <nome_fz>(void) { /* Codice scritto sopra */ }` con il codice dovuto alla direttiva *data* al suo interno.

Esempio:

Dato il seguente file in ingresso:

```
#include <stdio.h>

debug {
    fz debugStampaNumero;
    void stampaNumero(int n){
        PRINT("Numero :",n,"\\n");
    }
    data[[
        FOR a=1:3:1
        num { n=a; }
    ]]
}

void stampa(int a) [[
    debugStampaNumero();
    int c;
    for(c=0;c<a;c++)
        PRINT(c," ");
    PRINT("\\n");
]]

void main(void)[[
    stampa(3);
]
```

dovrà essere prodotto il seguente file:

```
#include <stdio.h>

void stampaNumero(int n){
    printf("Numero : %d\\n",n);
}

void debugStampaNumero(void) {
    int n;
    printf("num:1:\\n");
    n=1;
    stampaNumero(n);
    printf("num:2:\\n");
    n=2;
    stampaNumero(n);
    printf("num:3:\\n");
    n=3;
    stampaNumero(n);
}

void stampa(int a) {
    int c;
    debugStampaNumero();
    for(c=0;c<a;c++)
        printf("%d ",c);
    printf("\\n");
}

void main(void) {
    stampa(3);
}
```

Assieme al programma dovrà essere allegata una breve relazione contenente una veloce descrizione delle strutture dati utilizzate, di come il programma funziona e delle specifiche/limiti di funzionamento. Dovranno poi essere allegati dei **file di esempio** per verificare il corretto funzionamento del programma.