I file in C (III)

Laboratorio di Linguaggi di Programmazione
a.a. 2001/2002

dott.ssa Francesca A. Lisi
lisi@di.uniba.it

Sommario

- I file di testo (ancora per un'ultima volta)
  - realizzazione di analizzatori sintattici

Riferimenti

- capp. 11-12 di Deitel & Deitel
- par. 4.3 e 5.12 di Kernighan & Ritchie
- Tremblay, J. P., and Sorenson, P. G.: The Theory and Practice of Compiler Writing, McGraw-Hill, New York, 1985.
- Dispense del corso II parte

Realizzazione di analizzatori sintattici:
parsing di espressioni aritmetiche

Si consideri la seguente grammatica G per il linguaggio **miniP** mostrato la volta scorsa:

<expression> ::= <term>**+**<expression> |
<term>**-**<expression> |<term>
<term>::= <factor>***<term> | <factor>**/**<term> | <factor>
<factor>::= <identificatore> | **(**<expression>**)**

e si costruisca un programma C che realizzi un parser ricorsivo-discendente per G

Parsing di espressioni aritmetiche (I)

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

#define MAXTOKEN 256

struct token {
   char name[MAXTOKEN+1];
   int cat;};

int gettoken(FILE *src, struct token *tkn,
  int max);
```

Parsing di espressioni aritmetiche (II)

```
int main() {
  FILE *source;
  struct token next;

  source = fopen("sorgente","r");
  if (expression(source))
     printf(”Espressione corretta!");
  else printf(”Espressione non corretta!");
  fclose(source);
  system("PAUSE");
  return 0;
}
```

Parsing di espressioni aritmetiche (III)

```
int expression(FILE *src){
  struct token next;
  int ok=1;

  if (term(src)) {
     gettoken(src, &next, MAXTOKEN));
     if ((next.cat==PLUS)||
                 (next.cat==MINUS))
          if !(expression(src)) ok=0;
     else ok=0;}
  else ok=0;
  return ok;
}
```

Parsing di espressioni aritmetiche (IV)

```
int term(FILE *src){
  struct token next;
  int ok=1;

  if (factor(src)) {
     gettoken(src, &next, MAXTOKEN));
     if ((next.cat==MULT)|| next.cat==DIV))
          if !(term(src)) ok=0;
     else ok=0;}
  else ok=0;
  return ok;
}
```

Parsing di espressioni aritmetiche (V)

```
int factor(FILE *src){
  struct token next;
  int ok=1;

  gettoken(src, &next, MAXTOKEN));
  if (next.cat==PAR_AP)
     if (expression(src)){
          gettoken(src, &next, MAXTOKEN));
          if (next.cat!=PAR_CH) ok=0;}
     else ok=0;
  else if (next.cat!=ID) ok=0;
  return ok;
}
```

Realizzazione di analizzatori sintattici:
parsing di espressioni relazionali

Si costruisca un programma in C che realizzi un parser ricorsivo-discendente per la seguente grammatica:

<expr_relaz> ::= <expression> <op_relaz><expression>

<op_relaz>::= **>** | **>=** | **<** | **<=** | **<>** | **=**

dove <expression> è definita come in G.

Realizzazione di analizzatori sintattici:
parsing di un blocco di istruzioni

Si costruisca un programma in C che realizzi un parser ricorsivo-discendente per la seguente grammatica:

<instr>::= <assign_instr> | <while-do-instr> | <instr_block>

<assign_instr> ::= <identifier> **:=** <expression>**;**

<while-do-instr> ::= **while** <expr_relaz> **do** <instr_block>

<instr_block> ::= <instr> | **begin** <instr_seq> **end**

<instr_seq> ::= <instr> | <instr> <instr_seq>

dove <expression> è definita come in G.