innovate achieve lead

BITS Pilani

Lecture 6

Expression and statement constructs

List of constructs in a programming language

- Expressions
- Function definitions and call statements (returning values/ not returning any value)
- Statements
  - Declaration statements
  - Assignment (a=b*c+d-f/s; uni-processor view, instruction cycle, semantic)
  - Iterative statements (For, while, do while, repeat until)
  - Conditional statement (If, switch-case values)
  - Goto statements
  - Return, break statements
- Variables
- Datatypes (pointers, records, arrays, objects, lists etc.)
- Separator

Program elements from language designer's perspective

- Variables-characters and words, length and patterns
- Numbers-integer, real, precision, form (23.56, 2.56e+2)
- Operators- patterns and meaning (=, ==, <=, >=, &&, ||), associativity and precedence of ops
- Keywords-patterns, names. why?
- Grammar-syntactic structure
- Constructs for computation – expressions,
- Constructs for execution flow
- Semantic privileges-precedence of function def before its call, recursive, overloading etc.
- Type system

Grammar

- A grammar is defined as a four valued tuple (N, T, S, P) where N is the set of non-terminals, T is the set of terminals, S is the start symbol and P is the set of production rules.
- Example: Grammar G to generate palindrome strings over alphabet {0,1} is defined as below

  G= ({S}, {0,1}, S, P)

Where p is

S→0 S 0

S→1 S 1

S→0

S→1

Understanding expression construct

- Examples:

a *(one term)*

b*c *(one term with a binary operator)*

a+b*c *(one expression with two terms and +)*

a-d**+b*c** *(one expression with one expression and two terms and +)*

a+b*c-d

Recursive construction of an expression

- An expression
  - can be a term
  - Can be a combination of two terms with a binary operator
  - A term can also be an expression

- Grammar rules (P) to derive an expression are given below
  1. <expression>→ <term>
  2. <expression>→ <term> <operator> <term>
  3. <term>→<expression>
  4. <term>→ID //ID is a token name for a,b,c and d
  5. <operator>→PLUS
  6. <operator>→MINUS
  7. <operator>→MUL
- Grammar G=({<expression>, <term>, <operator>}, {ID, PLUS, MINUS, MUL}, <expression>, P)

What do we mean by deriving an expression using the underlying grammar?

a+b*c-d

<expression>→ <term>

<term>→<expression>

<expression>→ $<term_1>$ <operator> $<term_2>$

$<term_1>$→a //token ID is taken in actual

<operator>→PLUS

$<term_2>$→<expression>

<expression>→ $<term_1>$ <operator> $<term_2>$

$<term_1>$→<expression> // for b*c

<expression>→ $<term_1>$ <operator> $<term_2>$

$<term_1>$→b

<operator>→MUL

$<term_2>$→c

$<term_2>$→d

<operator>→MINUS

Parse tree for a+b*c-d

- Home work

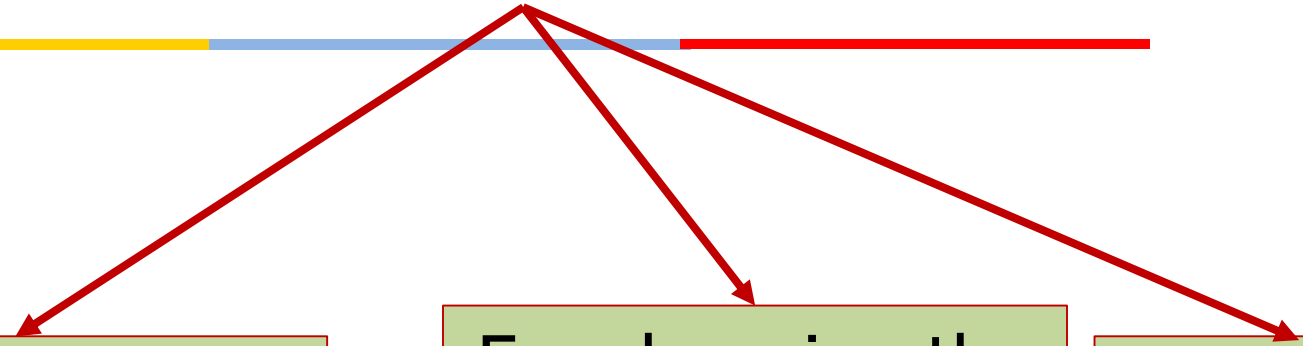
Redefining grammar for expression

- Grammar rules (P) to derive an expression are given below
  1. <expression>→ <expression> <operator1> <term>
  2. <expression>→ <term>
  3. <term>→<term> <operator2> <factor>
  4. <term>→<factor>
  5. <factor>→ ID
  6. <operator1>→PLUS | MINUS
  7. <operator2>→MUL |DIV
- Grammar G=({<expression>, <term>, <operator1, <operator2>, <factor>}, {ID, PLUS, MINUS, MUL}, <expression>, P)

Design issues for expression construct

- Which operators are to be included for computations?
- What are the operator associativity rules?
- What are the operator precedence rules?
- Does the language allow operator overloading?
- What will be type rules for operators? Whether type conversion will be allowed implicitly?

‘Statements’ construct



For Computation

- Assignment statement
- Function call statement
- Input statement
- Increment statement

For changing the execution flow

- conditional statement
- Iterative statement
- goto statement
- break statement
- Return statement

others

- Declaration statement
- output statement

Computation based statements

- The most general form of an assignment statement has at least one variable on the left hand side of the assignment operator.

  example a=b+c*d;

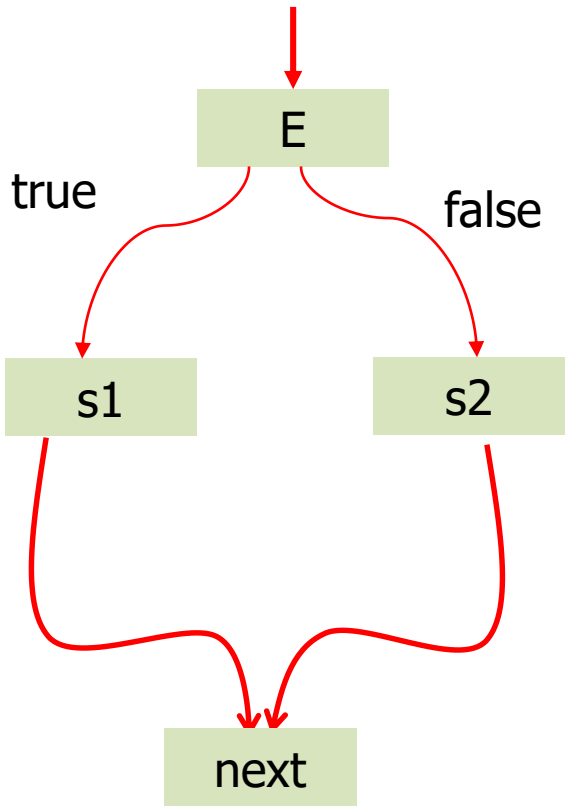
  a=sum(b,c,d);

- The RHS of the statement gets the computed value which is then stored in the LHS variable.
- The RHS can be an expression or a function call.
- Other forms of statements which can change the value of a variable are read statements and increment statements.

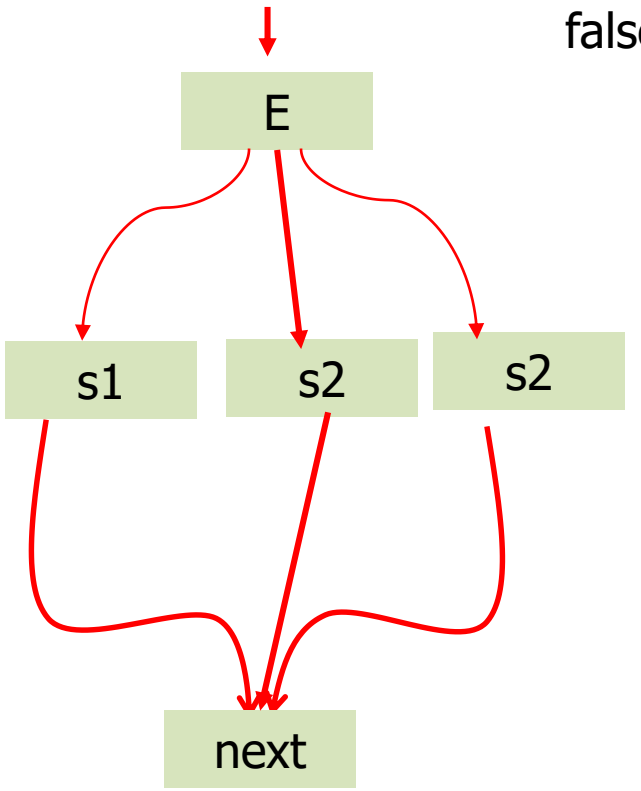
Statements that handle the execution flow

- These statements are called “control statements”
- All such statements exhibit “single entry-single exit” control flow.
- Example statements that change the flow of execution: if-then-else, switch, for, while, break, return etc.

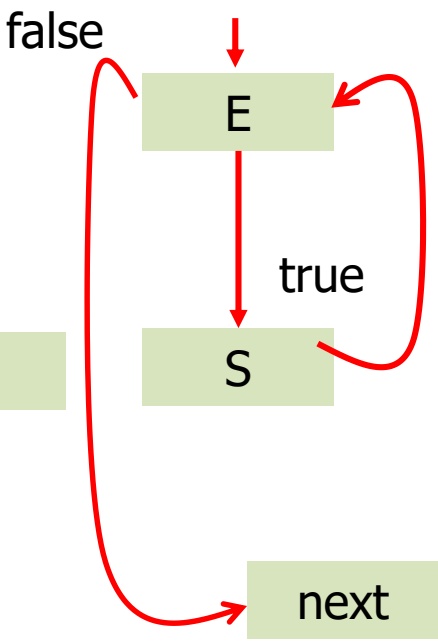
Flow of execution



If statement



switch statement



While statement

Design issues

- The execution flow should be evident from the syntax.
- Selection statements
  - What should be the form and type of expression that controls selection?
  - How should the meaning of nested selector be specified?
- Iterative statements
  - What are the type and scope of loop variables?
  - Can the loop variable be reassigned a value within the loop body?
  - Should the loop parameter be evaluated only once or once for every iteration?
  - Should the loops be counter controlled or logic controlled or both?

Assignment statement

- Syntax

  <assignment_stmt>→ ID ASSIGNOP <expression>

- Semantics

  Compute the value of <expression> and copy the value in the memory location bound to the variable identifier (tokenized as ID)

- Example

  abc_1 = xyz+qr*pr (ID=ID+ID*ID)

- Nonterminal on the LHS of the grammar rule represents the construct: <assignment_stmt>

Function call statement

- Syntax

  <functioncall_stmt>→ ID ASSIGNOP FUNCTION_ID <actual para_list>

- Semantics

  Copy the value returned by the function in the memory location bound to the variable identifier (tokenized as ID)

- Example

  abc_1 = sum(a,b)

- Nonterminal on the LHS of the grammar rule represents the construct: <functioncall_stmt>

Iterative statements (FOR)

- Syntax

  <iterative_for_stmt>→ FOR OP <c1> semicolon <c2> semicolon <c3> CP <statements>

  <c1>→ ID ASSIGNOP <expression>

  <c2>→ <expression> <relationalOp> <expression>

  <c3>→ <assignment_stmt>

  <relationalOp>→ LT|GT|EQ|LE|GE|NE

- Semantics

  Initialize <c1> and execute only once

  Repeat the execution of <statements> based on <c2>

  Execute <c3> everytime before leaving the loop.

- Example

  for (x=0; x<a+10; x=x+2) y=x+a*2;

- Nonterminal on the LHS of the grammar rule represents the construct: **<iterative_for_stmt>**