STATEMENTS

Kashika grover
Computer science engineering,
Dronacharya college of engineering

ABSTRACT:- **: this paper basically dals with the topic : THE C PROGRAMMING. C language programming is most one of important basic curriculums for computer curriculum teaching in science and engineering college.**

INTRODUCTION

We present formal operational semantics for the C programming language. This paper will basically covers thestatements as well as the representation of the c language.

Statement Classiv cation in C

there are six categories of statements in C:

1. Expression statements, which evaluate the associated expression.
2. Selection statements (if and switch). 3. Iteration statements (for, while, and do-while).
3. Iteration statements (for, while, and do-while).
4. Jump statements (goto, continue, break, and return).
5. Labeled statements (case and default statements used within the scope of a switch statement, and targets of goto statements).
6. Compound statements, consisting of a (possibly empty) list of local variable declarations and a (possibly empty) list of statements.

Expression Statements

An expression statement has one of the following forms:

expression-statement ! ; expression-statement ! expression ;

To execute an expression statement, evaluate the attached expression (if any), even though the resulting value will not be used. While this may seem unnecessary, note that the evaluation of an expression in C may generate side-e ects (such as assigning a value to a variable). Note also that the evaluation of an expression may not halt. In this algebra, the evaluation of expressions is handled by an external function TestValue: tasks ! results . Since expression statements perform no additional work, the algebra simply proceeds to the next task.

Selection Statements

There are two types of selection statements in C:

1.if statements

2.switch statements.

IF STATEMENT

An if statement has one of the following forms: if-statement ! if (expression) statement1

if-statement ! if (expression) statement1 else. statement2 where statement1 and statement2 are statements.

switch Statements

The branching decision made in the if statement is represented by an element of the tasks universe for which the TaskType function returns branch.,

A switch statement has the following form:

switch-statement ! switch (expression) body

where body is a statement, usually compound. Within the body of a switch statement there are labeled case and default statements. Each case or default is associated with the smallest enclosing switch statement. To execute a switch statement, evaluate the guard expression, and within the body of the switch, transfer control to the case statement for the switch whose labeled value matches the value of the expression, or to the default statement for the switch, whichever comes. If no such statement is found, transfer control to the statement following the switch statement.

While Statements

A while statement has the following form:

while-statement ! while (expression) body

where body is a statement. To execute a while statement, keep evaluating the guard expression until the value of the expression becomes zero. Each time that the value of the guard expression is not zero, execute body. Since the only types of tasks used to represent while statements are the expression and branch tasks. In such a situation, our abstract machine would continue as if the loop had been entered normally (i.e., after completion of the statement body, control returns to the guard expression to be evaluated). We nbelieve this is a reasonable interpretation of such an event.

do-while Statements

A do-while statement has the following form:

do-while-statement ! do body while (expression) ;

where body is a statement.

do-while statements are identical to while statements except that the guard expression and

statement body are visited in the opposite order. As with while loops, no new transition rules are required to model the behavior of do-while statements.

for Statements

The most complete form of the for statement is:

for-statement ! for (initializer ; test ; update) body

where test , and update are expressions, any of which may be omitted, and body is a statement, usually compound. We begin by describing the behavior and representation of a for statement when all expressions are present. In executing a for statement, begin by evaluating the initializer. Evaluate the test next; if the result is non-zero, execute the body and evaluate the update (in that order) and re-evaluate the test. If the value of the test is zero, transfer control to the statement following the for loop.

Jump Statements

A jump statement has one of the following forms: jump-statement ! goto identier ; jump-statement ! continue ; jump-statement ! break ; jump-statement ! return ; jump-statement ! return expression ;

Each of these jump statements is a command indicating that control should be unconditionally transferred to another task in the task graph:

goto statements :indicate directly the task to which control passes**.**

continue statements may only occur within the body of an iteration statement. For a given continue statement C, let S be the smallest iteration statement which includes C. Executing C transfers control to the task within S following the statement body of S: e.g., for for statements, control passes to the update expression, while for while statements, control passes to the guard expression.

break statements may occur within the body of an iteration or switch statement. For a given break statement B, let S be the smallest iteration or switch statement which includes B. **return statements** occur within the body of function abstractions, indicating that the current function execution should be terminated. A more complete discussion of return statements will be presented in Algebra Four, where function abstractions are presented. For now, we assert that executing a return statement should set CurTask to undef , which will bring a halt to the algebra, since we only have one function (main) being executed.

Labeled Statements

A labeled statement has one of the following forms: labeled-statement ! identi er : statement ;

labeled-statement ! case constant-expression : statement

labeled-statement ! default : statement

Statement labels identify the targets for control transfer in goto and switch statement

Compound Statements

A compound statement has the following form:

compound-statemen- !{i f declaration-list statement-list }

where the declaration and/or statement lists may be empty.)

Since NextTask indicates the order in which tasks are processed, we have no need for rules concerning compound statements. Each statement or declaration in a compound statement is linked to its successor via NextTask . (Declarations are not treated until Algebra Three; nonetheless, the same principle holds for declaration tasks.)