Compiler Construction

BPDC

(Lab - 06)

1 Mini-compiler (Version II)

As before, start with the solution for Lab-05.

1. Update the compiler to let the user to include other header files too. Either from standard library, say $\#include < something.h >$, or from a user defined directory(the filename in double quotes, say, $\#include"/home/user/something.h''$). The compiler doesn't have to verify the existence of the so included header file, just have to check the template structure.
2. Extend your compiler to let the programmer to use arrays in programs.
   (a) The compiler should successfully parse declaration statements of the form $int\ x, a[10]$;
   (b) Correspondingly, we have to extend the symbol table implementation to encode this additional information.
   (c) Further, the compiler has to ensure that those variables declared as arrays will be accessed only using indices (say, given a declaration statement of the form $int\ a[20], c$; an expression $a[i] = a[j] + c$; should compile successfully, while $a[i] = a + c$; should throw an error).
3. Incorporate read($scanf$) and write($printf$) functions to your compiler.
   (a) The template of $scanf$ function is: $scanf("formatstring'', var1, var2, ...)$; where the format string should contain the right sequence of format specifiers ,$\%d(int)$, $\%c(char)$, $\%f(float)$, $\%lf(double)$ or $\%s(character\ array)$ that exactly matches the data types of those associated variables (this has to be verified by referring to the symbol table). Here, there is a minor conflict with the previous part of the question. If the user declares a variable as a character array, and refers to it using the format specifier $\%s$, then this should be treated as an exceptional case and the user should be permitted to refer to the character array using the name alone (say, $scanf("\%s'', str)$;), and otherwise should be forced to refer using indexing (say, as $a[i]$).
   (b) You could assume the user to use % operator inside the format string only as in a format specifier, but not as a regular character.
   (c) Further, for all those format specifiers other than $\%s$, $scanf$ requires to be provided with the address of the variable ($scanf("\%d'', \&x)$; and $scanf("\%s'', str)$; are valid expressions while $scanf("\%d'', x)$; is an invalid expression assuming x to be of type int and str to be $char\ array$).