PRACTICAL SHEET – 1
Compiler Construction
(CS 015)

Faculty: Karamjit Cheema

Implement “Loop-and-switch” or “ad hoc” scanner in C.

Hint :

A "loop & switch" implementation consists of a main loop that reads characters one by one from the input file and uses a switch statement to process the character(s) just read. The output is a list of tokens and lexemes from the source program. The following program fragment shows a skeletal implementation of a simple loop and switch scanner.
The main program calls InitScanner and loops calling ScanOneToken until EOF.
ScanOneToken reads the next character from the file and switches off that char to decide how to handle what is coming up next in the file. The return values from the scanner can be passed on to the parser in the next phase.

```
#define  T_SEMICOLON     ';' // use ASCII values for single char tokens
#define  T_LPAREN  '('
#define  T_RPAREN  ')'
#define  T_ASSIGN  '='
#define  T_DIVIDE   '/'
...

#define T_WHILE 257 // reserved words
#define T_IF 258
#define T_RETURN 259
...

#define T_IDENTIFIER  268 // identifiers, constants, etc.
#define T_INTEGER  269
#define T_DOUBLE  270
#define T_STRING 271
#define T_END         349 // code used when at end of file
#define T_UNKNOWN         350 // token was unrecognized by scanner

struct token_t
{
        int type;         // one of the token codes from above
        union
        {
                char stringValue[256];          // holds lexeme value if string/identifier
                int intValue;               // holds lexeme value if integer
                double doubleValue;  // holds lexeme value if double
        } val;
};
```

```
int main(int argc, char *argv[])
{
        struct token_t token;
        InitScanner();
        while (ScanOneToken(stdin, &token) != T_END)
                ; // here is where you would process each token
        return 0;
}


static void InitScanner()
{
        create_reserved_table();        // table maps reserved words to token type
        insert_reserved("WHILE", T_WHILE)
        insert_reserved("IF", T_IF)
        insert_reserved("RETURN", T_RETURN)
....
}


static int ScanOneToken(FILE *fp, struct token_t *token)
{
        int i, ch, nextch;
        ch = getc(fp);  // read next char from input stream

        while (isspace(ch))     // if necessary, keep reading til non-space char
                ch = getc(fp);  // (discard any white space)

        switch(ch)
         {
                case '/':                       // could either begin comment or T_DIVIDE op
                        nextch = getc(fp);

                if (nextch == '/' || nextch == '*')
                                ; // here you would skip over the comment
                else
                                ungetc(nextch, fp); // fall-through to single-char token case

                case ';':
                case ',':
                case '=':
                // ... and other single char tokens

                        token->type = ch; // ASCII value is used as token type
                        return ch;      // ASCII value used as token type

                case 'A': case 'B': case 'C':     // ... and other upper letters

                        token->val.stringValue[0] = ch;
```

```
                    for (i = 1; isupper(ch = getc(fp)); i++) // gather uppercase
                            token->val.stringValue[i] = ch;

                    ungetc(ch, fp);

                    token->val.stringValue[i] = '\0'; // lookup reserved word
                    token->type = lookup_reserved(token->val.stringValue);

                    return token->type;

                    case 'a': case 'b': case 'c':  // ... and other lower letters

                    token->type = T_IDENTIFIER;
                    token->val.stringValue[0] = ch;

                    for (i = 1; islower(ch = getc(fp)); i++)
                            token->val.stringValue[i] = ch; // gather lowercase

                    ungetc(ch, fp);

                    token->val.stringValue[i] = '\0';

                    if (lookup_symtab(token->val.stringValue) == NULL)
                            add_symtab(token->val.stringValue); // get symbol for ident

                    return T_IDENTIFIER;

                    case '0': case '1': case '2': case '3': //....and other digits

                    token->type = T_INTEGER;
                    token->val.intValue = ch - '0';

                    while (isdigit(ch = getc(fp))) // convert digit char to number
                            token->val.intValue = token->val.intValue * 10 + ch - '0';

                    ungetc(ch, fp);
                    return T_INTEGER;

            case EOF:       return T_END;
            default:    // anything else is not recognized

                    token->val.intValue = ch;
                    token->type = T_UNKNOWN;
                    return T_UNKNOWN;
        }   // Switch ends
}
```