|  |
| --- |
|  |
| TWO PASS MACROPROCESSOR |
|  |
|  |
| **INCHARGE: Dheeraj D.**  **Asst.Prof**  **Dept.ISE,PESIT.**  **Dept.ISE,PESIT.**    **USN: 1PI10IS134**  **1PI10IS109**  **1PI10IS124** |
|  |

**ABSTRACT**

A macro represents a commonly used group of statements in the source programming language. The macroprocessor replaces each macro instruction with this corresponding group of source languagestatements. This is called expanding the macros.

Macroinstructions allow the programmer to write a shorthand version of a program and leave the mechanical details to be handled by the macroprocessor. In a 2-pass macroprocessor, the macro definition is in the 1st pass and the macro calls and expansion take place in the 2nd pass.

**PASS 1-MACRO DEFINITION:** The algorithm for pass 1 tests each input line. If it is a MACRO pseudo-op, the entire macrodefinition that follows is saved in the next available locations in the Macro Definition Table(MDT).The first line of the definition is the macro name line. The name is entered into the Macro Name Table (MNT), along with a pointer to the first location of the MDT entry of the definition. When the END pseudo-op is encountered , all of the macrodefinitions have been processed, so control transfers to pass2 in order to process calls.

**PASS 2-MACRO CALLS and EXPANSIONS:** The algorithm for pass 2 tests the operation mneumonic of each input line to see if it is a name in the MNT. When a call is found, the call processor seta a pointer, the Macro Definition Pointer(MDTP),to the corresponding macro definition stored in the MDT. The initial value of the MDTP is obtained, from the “MDT index” field of the MNT entry. The macro expander prepares the Argument List Array(ALA) consisting of a table of dummy argument indices and corresponding arguments to the call. This list is simply a succession of symbols ordered to match the dummy arguments on the name card(the first in the label argument, which is considered to have an index of zero).Arguments not represented in a call are considered blank, and superfluous arguments are ignored. In the case of argument reference by position, this scheme is completely straightforward. For reference by name, the macroprocessor locates the dummy argument on the macro name line (which is available at the beginning of the definition in the MDT) in order to determine the proper index.

Reading proceeds from the MDT; as each successive line is read, the values from the argument list are substituted for dummy argument indices in the macro definition. Reading of the MEND line in the MDT terminates expansion of the macro and scanning continues from the input file. When the END pseudo-op is encountered, the expanded pseudo deck is transferred to the assembler for further processing.

**INTRODUCTION TO PROJECT**

The project implements a simple 2-pass macroprocessor in C language in two phases, pass 1 and pass 2.Pass 1 code implements macro definition phase, while pass 2 code implements the macro calls and expansion phase of the macroprocessor. The program handles single macro definition with no nested or recursive macro calls.

Figures 1 and 2 are flowcharts of the macro definition and expansion algorithms. Each of the algorithms makes a line-by-line scan over input. The ‘READ’ boxes refer to the fetching of successive input lines from secondary storage into a workspace.

**REQUIREMENT SPECIFICATION**

* TOOLS: Codeblocks, Notepad.
* OPERATING SYSTEM : Windows.
* LANGUAGE: C

**IMPLEMENTATION**

The program is implemented using the standard C library files and the usual C programming language constructs.

The input is a source programming language program in a text file which is processed by the program. The output is the processed files obtained after scanning the input file.

The program consists of two .c files,pass1.c and pass2.c. Pass 1 code implements macro definition phase, while pass 2 code implements the macro calls and expansion phase of the macroprocessor.The output of first pass is the macro name table(MNT or nametab) and the macro definition table(MDT or deftab),which are written into the nametab and deftab files respectively.The output of the second pass is the Argument List Array(ALA or argtab) and the source code with the expanded macro in the output file.

The input code is usually an assembly language code consisting of instructions. The instructions consist of the index,mneumonic and operands fields. The ‘MACRO’ and ‘MEND’ mneumonics indicate the start and end of a macro definition.The entities in its operands field are the macro formal arguments which are replaced with the actual arguments in the macro call for expansion.

File pointers are used to open,read/write and close the files( FILE \*fp; ).fopen(): FILE \*fopen(const char \*filename, const char \*mode) ,fscanf(),fprintf() and fclose():int fclose(FILE \*a\_file) inbuilt functions are used to open,scan,print in the files and close the files respectively.A file can be opened to read(r),write(w) or read and write(overwrite:w+) options. fseek():int fseek(FILE \*stream, long int offset, int whence), is used to reposition the file pointer to the required position in the file stream.getch():int getch(void), is used to obtain the next character from the keyboard.strlen() function: size\_t strlen(const char \*str), is used to obtain the length of a string(character array).strcmp():int strcmp(const char \*str1, const char \*str2), is used to compare two strings.If they are the same, they return 0.

**CONCLUSION**

The program works in two passes as expected with the desired results. Implementation of a macroprocessor in two passes can be understood. The same program can be developed to handle more than one macro definition in the input code and nested macro calls, which are currently not dealt with. We understand that a 2-pass macroprocessor is simpler and more efficient than a single pass macroprocessor in that the two operations: macro definition and macro calls and expansion are performed explicitly in two separate passes allowing for error recovery after each of the phase.Although ,memory usage might be a drawback since separate MNT,MDT and ALA data structures have to be maintained. Thus in this project, we examine the preprocessor function of the compiler in expanding the macros performed by the macroprocessor.