

```

bool Is_Valid_Label(char* label)
{
    if(strlen(label) > 6)
    {
        return false;
    }else
    {
        //1st char of label must be alpha
        if((label[0] >= 65 && label[0] <= 90) || (label[0] >= 97 && label[0] <= 122))
        {
            //check rest of chars in symbol, must be alpha-numeric
            int i = 1;
            while(i < strlen(label))
            {
                if((label[i]>=48 && label[i]<=57) || (label[i]>=65 && label[i]<=90) || (label[i]>=97 && label[i]<=122))
                {
                    //[i] char of the label is legal, check next one
                    i++;
                }else //label has illegal char
                {
                    return false;
                }
            }
        }else //label doesn't start w/ alpha
        {
            return false;
        }
        //label is valid
        return true;
    }
}

bool Is_In_Range(int i) //make sure a value is in representable range
{
    if(i < (-1*pow(2,19)) || i > (pow(2,19)-1))
    {
        return false;
    }else
    {
        return true;
    }
}

void Pass_One(ifstream& source, ofstream& intermediate, ofstream& listing, int& location_counter, Table& symbol_table, Table& literal_table, ENT_Table& ent_table, Table& ext_table)
{
    Table op_table;
    char buffer[80];
    char* token;
    int start_addr = 0;
    bool first_exe_op = true;
    bool End_Of_File = false;
    int Num_Source_Records = 1;

    do
    {
        source.getline(buffer, 80);
    }while (buffer[0] == ';');

    if(strlen(buffer)==0)
    {
        listing << "\nBlank line(s) in the file";
        cerr << "Program exited abnormally\n";
        exit(1);
    }

    token = strtok(buffer, " ");
    if(strcmp(token, "ORI")== 0) // some error checkin'
    {
        listing << "\nORI operation has no label.";
    }
}

```

```

cerr << "Program exited abnormally\n";
exit(1);
}
else
{
if(Is_Valid_Label(token)) //check for valid label
{
    intermediate << token << '\n';
}
else
{
    listing << "\nInvalid Label \"" << token << "\"";
    cerr << "Program exited abnormally\n";
    exit(1);
}
}
token = strtok(NULL, " ");
if (token == NULL || strcmp(token, "ORI") != 0)
{
    listing << "\nFirst non-comment record \"" << token << "\" (not ORI)";
    cerr << "Program exited abnormally\n";
    exit(1);
}

char* Initial_Load_Address;
Initial_Load_Address = strtok(NULL, " ");
if (Initial_Load_Address != NULL)
{
    intermediate << Initial_Load_Address << '\n';
}
else
{
    intermediate << "M\n"; //Tell Pass_Two() the program is relocatable
}

while (!source.eof())
{
do
{
    source.getline(buffer, 80);
}while (buffer[0] == ';');
if(strlen(buffer)==0 && !source.eof() && !End_Of_File)
{
    listing << "\nBlank line(s) in the file";
    cerr << "Program exited abnormally\n";
    exit(1);
}
char label[6];
char op_code[3];
bool Add_Symbol = false;
bool Look_For_Literal = true;

if (strlen(buffer) > 0 && buffer[0] != '\n')
{
    if(End_Of_File)
    {
        listing << "\nInstruction/Comment/Label following an END operation";
        cerr << "Program exited abnormally\n";
        exit(1);
    }
    Num_Source_Records++;
    if(Num_Source_Records > 200)
    {
        listing << "\nNumber of source records exceeded 200\n";
        cerr << "Program exited abnormally\n";
        exit(1);
    }
    token = strtok(buffer, " ");

```

```

if (! op_table.Is_In_Table(token)) //if the first token is a label/symbol...
{
if(Is_Valid_Label(token)) //check for valid label/
{
strcpy(label, token);
Add_Symbol = true;
token = strtok(NULL, " ");
}
else
{
listing << "\nInvalid Label/Symbol \"" << token << "\"";
cerr << "Program exited abnormally\n";
exit(1);
}
}
if (token == NULL)
{
listing << "\nMissing op code";
cerr << "Program exited abnormally\n";
exit(1);
}
strcpy(op_code, token);
token = strtok(NULL, " ");
//strtok(token, " ");
if (strcmp(op_code, "ORI") != 0 && strcmp(op_code, "END") != 0 && token == NULL)
{
listing << "\nNo operand for \"" << op_code << "\" operation";
cerr << "Program exited abnormally\n";
exit(1);
}
if(first_exe_op)
{
if(op_table.Get_Value(op_code) <= 15)
{
start_addr = location_counter;
first_exe_op = false;
}
}

int value = 0;
int num = 0; // a new variable used below
if (strcmp(op_code, "EQU") == 0)
{
if(Add_Symbol==false)// some error checkin'
{
listing << "\nEQU operation has no label.";
cerr << "Program exited abnormally\n";
exit(1);
}
value = atoi(token);
if (value < 0 || value > 255)
{
listing << "\nEQU operand \"" << token << "\" not in range.";
cerr << "Program exited abnormally\n";
exit(1);
}
}
Look_For_Literal = false;
}
else if (strcmp(op_code, "RES") == 0)
{
num=atoi(token);
if(num < 0 || num > 255) // some error checkin'
{
listing << "\nOperand in RES pseudo-op \"" << token
<< "\" not an integer in range";
cerr << "Program exited abnormally\n";
exit(1);
}
}
}

```

```

value = location_counter;
location_counter += atoi(token);
Look_For_Literal = false;
}
else if (strcmp(op_code, "ORI") == 0)
{
listing << "\nUnexpected ORI\n";
cerr << "Program exited abnormally\n";
exit(1);
}
else if (strcmp(op_code, "END") == 0)
{
if(Add_Symbol==true) // some error checkin'
{
listing << "\nEND operation has label \"" << label
<< '\n';
cerr << "Program exited abnormally\n";
exit(1);
}
value = location_counter;
if (token == NULL)
{
intermediate << start_addr << '\n'; //address of 1st executable op
}
else if (symbol_table.Is_In_Table(token))
{
//make sure value of symbol is in range
int tok_val = symbol_table.Get_Value(token);
if(tok_val < 0 || tok_val > 255)
{
listing << "\nOperand in END psuedo-op \"" << tok_val
<< "\" not an integer in range";
cerr << "Program exited abnormally\n";
exit(1);
}
else
{
intermediate << tok_val << '\n';
}
}
}
else
{
//make sure value of int is in range
int tok_val = atoi(token);
if(tok_val < 0 || tok_val > 255)
{
listing << "\nOperand in END psuedo-op \"" << tok_val
<< "\" not an integer in range";
cerr << "Program exited abnormally\n";
exit(1);
}
else
{
intermediate << tok_val << '\n';
}
}
}

Look_For_Literal = false;
End_Of_File = true;
}
else if (strcmp(op_code, "ENT") == 0)
{
if(token != NULL && token[0] != ';')
{
char* temp;
temp = strtok(token, ",");
while(temp != NULL && temp[0] != ';')
{
if(ent_table.Is_In_Table(temp))

```

```

    {
        listing << "\nENT symbol multiply defined\n";
        cerr << "Program exited abnormally\n";
        exit(1);
    }
    else if(! Is_Valid_Label(temp))
    {
        listing << "\nInvalid Label \"" << token << "\"";
        cerr << "Program exited abnormally\n";
        exit(1);
    }else
    {
        ent_table.Put_In_Table(temp, 0);
        temp = strtok(NULL, ",");
    }
}
}
else if(strcmp(op_code, "EXT") == 0)
{
    if(token != NULL && token[0] != ';')
    {
        char* temp;
        temp = strtok(token, ",");
        while(temp != NULL && temp[0] != ';')
        {
            if(ext_table.Is_In_Table(temp))
            {
                listing << "\nEXT symbol multiply defined\n";
                cerr << "Program exited abnormally\n";
                exit(1);
            }
            else if(! Is_Valid_Label(temp))
            {
                listing << "\nInvalid Label \"" << token << "\"";
                cerr << "Program exited abnormally\n";
                exit(1);
            }else
            {
                ext_table.Put_In_Table(temp, 0);
                temp = strtok(NULL, ",");
            }
        }
    }
}
else //Not a Pseudo-Op
{
    token[strlen(token)] = '.';
    value = location_counter;
    char* temp_tok;
    temp_tok = strtok(token, ",");
    token = strtok(NULL, "(.)");
    location_counter++;
}
if (Add_Symbol)
{
    if (symbol_table.Is_In_Table(label))
    {
        listing << "\n\"" << label << "\" symbol defined twice";
        cerr << "Program exited abnormally\n";
        exit(1);
    }else if(ext_table.Is_In_Table(label))
    {
        listing << "\nExternal symbol \"" << label << "\" illegally defined";
        cerr << "Program exited abnormally\n";
        exit(1);
    }
}
else

```

```

    {
        symbol_table.Put_In_Table(label, value);
        if (ent_table.Is_In_Table(label))
        {
            ent_table.Define_This(label, value);
        }
    }
}

if (Look_For_Literal)
{
    if (token != NULL && (! literal_table.Is_In_Table(token)))
    {
        if (token[0] == '=')
        {
            if (strcmp(op_code, "BR") == 0 || strcmp(op_code, "BRZ") == 0 || strcmp(op_code, "BRN") == 0 || strcmp(op_code, "BRS") == 0 || strcmp(op_code, "ST") == 0 || strcmp(op_code, "SHR") == 0 || strcmp(op_code, "SHL") == 0 || strcmp(op_code, "IO") == 0)
            {
                listing << "\nAttempt to use a literal with \""
                    << op_code << "\" operation";
                cerr << "Program exited abnormally\n";
                exit(1);
            }
            char* temp_tok;
            temp_tok = strtok(token, "=");
            int literal = atoi(temp_tok);

            if(! Is_In_Range(literal))
            {
                listing << "\nLiteral not an integer in range.";
                cerr << "Program exited abnormally\n";
                exit(1);
            }
            literal_table.Put_In_Table(token, 0);
        }
    }
}

if(! ent_table.All_Defined()) //make sure all ENT symbols have been defined
{
    listing << "\nNot all entry symbols defined";
    cerr << "Program exited abnormally\n";
    exit(1);
}
location_counter = literal_table.Update_Values(location_counter);
intermediate << location_counter;
}

```