DATE :

EXPERIMENT NO – 2.1
IMPLEMENT PASS 1 OF A TWO PASS ASSEMBLER

AIM

To implement pass 1 of a two pass assembler .

ALGORITHM

1. Start
2. Main Function (main()):
   - Declares variables for label, opcode, and operand.
   - Calls the passOne() function.
3. Pass One (passOne() Function):
   - Declares variables for locctr (location counter), start, and length.
   - Opens files for input, output, symbol table, intermediate code, and length.
   - Reads the first instruction from "input.txt".
   - If the opcode is "START", sets the start address and initializes locctr. Writes the initial line to the intermediate file.
   - Enters a loop until the opcode is "END":
     - Writes intermediate code and updates the symbol table if a label is present.
     - Reads opcode and mnemonic from "optab.txt" and checks for opcode match.
     - Increments locctr based on different opcode types: WORD, RESW, BYTE, RESB.
     - Reads the next instruction from "input.txt".
   - Writes the final instruction to the intermediate file.
   - Closes files and calls the display() function.
   - Calculates the program length and writes it to the length file.
4. Display (display() Function):
   - Opens files for input, intermediate, and symbol table.
   - Displays the contents of these files character by character.
   - Closes the files after display.
5. The passOne() function performs the first pass of the assembler, generating intermediate code and symbol table information . The display() function simply displays the contents of input, intermediate, and symbol table files.
6. Stop

INPUT

input.txt

```
**        START     2000
**        LDA       FIVE
**        STA       ALPHA
**        LDCH      CHARZ
**        STCH      C1
ALPHA     RESW      2
FIVE      WORD      5
CHARZ     BYTE      C'Z'
C1        RESB      1
**        END       **
```

optab.txt

```
LDA    03
STA    0f
LDCH   53
STCH   57
END    *
```

OUTPUT

symtab.txt

```
ALPHA        2012
FIVE         2018
CHARZ        2021
C1           2022
```

intermediate.txt

```
          **          START   2000
2000      **          LDA     FIVE
2003      **           STA    ALPHA
2006     **          LDCH      CHARZ
2009     **           STCH     C1
```

PROGRAM

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void passOne(char label[10], char opcode[10], char operand[10], char code[10], char
mnemonic[3]);
void display();
int main()
{
    char label[10], opcode[10], operand[10];
    char code[10], mnemonic[3];
    passOne(label, opcode, operand, code, mnemonic);
    return 0;
}
void passOne(char label[10], char opcode[10], char operand[10], char code[10], char
mnemonic[3])
{
    int locctr, start, length;
    FILE *fp1, *fp2, *fp3, *fp4, *fp5;
    fp1 = fopen("input.txt", "r");
    fp2 = fopen("optab.txt", "r");
    fp3 = fopen("symtab.txt", "w");
    fp4 = fopen("intermediate.txt", "w");
    fp5 = fopen("length.txt", "w");
    fscanf(fp1, "%s\t%s\t%s", label, opcode, operand);
    if (strcmp(opcode, "START") == 0) {
        start = atoi(operand);
        locctr = start;
        fprintf(fp4, "\t%s\t%s\t%s\n", label, opcode, operand);
        fscanf(fp1, "%s\t%s\t%s", label, opcode, operand);
    }
    else {
        locctr = 0;
    }
    while (strcmp(opcode, "END") != 0) {
        fprintf(fp4, "%d\t%s\t%s\t%s\n", locctr, label, opcode, operand);
        if (strcmp(label, "**") != 0) {
            fprintf(fp3, "%s\t%d\n", label, locctr);
```

```
2012    ALPHA         RESW    2
2018    FIVE          WORD    5
2021    CHARZ        BYTE     C'Z'
2022    C1            RESB     1
2023     **          END       **
```

length.txt

```
23
```

TERMINAL

```
The contents of Input Table :

**          START       2000
**          LDA         FIVE
**          STA         ALPHA
**          LDCH        CHARZ
**          STCH        C1
ALPHA       RESW        2
FIVE        WORD        5
CHARZ       BYTE        C'Z'
C1          RESB        1
**          END         **


The contents of Output Table :

        **          START       2000
2000    **          LDA         FIVE
2003    **          STA         ALPHA
2006    **          LDCH        CHARZ
2009    **          STCH        C1
2012    ALPHA       RESW        2
2018    FIVE        WORD        5
2021    CHARZ       BYTE        C'Z'
2022    C1          RESB        1
2023    **          END         **


The contents of Symbol Table :
        }
        fscanf(fp2, "%s\t%s", code, mnemonic);
        while (strcmp(code, "END") != 0) {
            if (strcmp(opcode, code) == 0) {
                locctr += 3;
                break;
            }
            fscanf(fp2, "%s\t%s", code, mnemonic);
        }
        if (strcmp(opcode, "WORD") == 0) {
            locctr += 3;
        }
        else if (strcmp(opcode, "RESW") == 0) {
            locctr += (3 * (atoi(operand)));
        }
        else if (strcmp(opcode, "BYTE") == 0) {
            ++locctr;
        }
        else if (strcmp(opcode, "RESB") == 0) {
            locctr += atoi(operand);
        }
        fscanf(fp1, "%s\t%s\t%s", label, opcode, operand);
    }
    fprintf(fp4, "%d\t%s\t%s\t%s\n", locctr, label, opcode, operand);
    fclose(fp4);
    fclose(fp3);
    fclose(fp2);
    fclose(fp1);
    display();
    length = locctr - start;
    fprintf(fp5, "%d", length);
    fclose(fp5);
    printf("\nThe length of the code : %d\n", length);
}
void display() {
    char str;
    FILE *fp1, *fp2, *fp3;
    printf("\nThe contents of Input Table :\n\n");
```

```
ALPHA        2012
FIVE         2018
CHARZ        2021
C1           2022

The length of the code : 23
```

```
    fp1 = fopen("input.txt", "r");
    str = fgetc(fp1);
    while (str != EOF) {
        printf("%c", str);
        str = fgetc(fp1);
    }
    fclose(fp1);
    printf("\n\nThe contents of Output Table :\n\n");
    fp2 = fopen("intermediate.txt", "r");
    str = fgetc(fp2);
    while (str != EOF) {
        printf("%c", str);
        str = fgetc(fp2);
    }
    fclose(fp2);
    printf("\n\nThe contents of Symbol Table :\n\n");
    fp3 = fopen("symtab.txt", "r");
    str = fgetc(fp3);
    while (str != EOF) {
        printf("%c", str);
        str = fgetc(fp3);
    }
    fclose(fp3);
}
```

RESULT

C program for the implementation of pass 1 of a two pass assembler is executed successfully .

DATE :

EXPERIMENT NO – 2.2
IMPLEMENT PASS 2 OF A TWO PASS ASSEMBLER

AIM

To implement pass 2 of a two pass assembler .

ALGORITHM

1. Function Declarations:
   - display() function is declared to display the contents of various files.
2. Swap Function:
   - swap() function swaps the values of two characters using pointers.
3. Reverse Function:
   - reverse() function reverses a character buffer from index i to index j.
4. Integer to ASCII Conversion (itoa() Function):
   - itoa() converts an integer value to a string in a specified base.
   - Checks if the base is within the valid range (2 to 32).
   - Converts the absolute value of the number to a string representation in the specified base.
   - Handles negative numbers for base 10 by adding a negative sign.
   - Returns the reversed string representation of the number.
5. Main Function:
   - Declares variables for file pointers, addresses, lengths, symbols, opcodes, and mnemonics.
   - Opens input and output files.
   - Reads the "intermediate.txt" file to determine the final address and sets it as finaddr.
   - Processes the "intermediate.txt" file line by line until the opcode is "END":
     - Handles BYTE, WORD, RESB, RESW, and other operations.
     - Generates object code based on different opcodes and operands.
   - Writes formatted output to "output.txt" and object code to "objcode.txt".
   - Closes files and calls the display() function.
6. Display Function:
   - Opens various files to display their contents:
     - "intermediate.txt", "symtab.txt", "output.txt", and "objcode.txt".
   - Prints the contents of each file character by character.
   - Closes the files after display.

INPUT

intermediate.txt

```
          **          START   2000
2000      **          LDA      FIVE
2003      **           STA     ALPHA
2006     **          LDCH       CHARZ
2009     **           STCH      C1
2012    ALPHA         RESW      2
2018    FIVE          WORD      5
2021    CHARZ        BYTE       C'Z'
2022    C1            RESB      1
2023     **          END         **
```

symtab.txt

```
ALPHA          2012
FIVE           2018
CHARZ          2021
C1             2022
```

OUTPUT

objcode.txt

```
H^**^002000^0023
T^002000^22^332018^442012^532021^572022^000005^5A
E^002000
```

output.txt

```
                **     START          2000
2000            **     LDA            FIVE           332018
2003            **     STA            ALPHA          442012
2006            **     LDCH           CHARZ          532021
2009            **     STCH           C1             572022
2012         ALPHA     RESW           2
2018         FIVE      WORD           5              000005
```

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void display();
void swap(char *x, char *y) {
   char t = *x; *x = *y; *y = t;
}
char* reverse(char *buffer, int i, int j)
{
   while (i < j) {
      swap(&buffer[i++], &buffer[j--]);
   }
   return buffer;
}
char* itoa(int value, char* buffer, int base)
{
   if (base < 2 || base > 32) {
      return buffer;
   }
   int n = abs(value);
   int i = 0;
   while (n)
   {
      int r = n % base;
      if (r >= 10) {
         buffer[i++] = 65 + (r - 10);
      }
      else {
         buffer[i++] = 48 + r;
      }
      n = n / base;
   }
   if (i == 0) {
      buffer[i++] = '0';
   }
   if (value < 0 && base == 10) {
      buffer[i++] = '-';
```

```
2021      CHARZ   BYTE      C'Z'      5A
2022       C1     RESB      1
2023       **     END       **
```

TERMINAL

Intermediate file is converted into object code

The contents of Intermediate file:

```
        **        START     2000
2000    **        LDA       FIVE
2003    **        STA       ALPHA
2006    **        LDCH      CHARZ
2009    **        STCH      C1
2012    ALPHA     RESW      2
2018    FIVE      WORD      5
2021    CHARZ     BYTE      C'Z'
2022    C1        RESB      1
2023    **        END       **
```

The contents of Symbol Table :

```
ALPHA       2012
FIVE        2018
CHARZ       2021
C1          2022
```

The contents of Output file :

```
        **        START     2000
2000    **        LDA       FIVE      332018
2003    **        STA       ALPHA     442012
2006    **        LDCH      CHARZ     532021
2009    **        STCH      C1        572022
2012    ALPHA     RESW      2
2018    FIVE      WORD      5         000005
```

```
    }
    buffer[i] = '\0';
    return reverse(buffer, 0, i - 1);
}
int main()
{
    char a[10], ad[10], label[10], opcode[10], operand[10], symbol[10];
    int start, diff, i, address, add, len, actual_len, finaddr, prevaddr, j = 0;
    char mnemonic[15][15] = {"LDA", "STA", "LDCH", "STCH"};
    char code[15][15] = {"33", "44", "53", "57"};
    FILE *fp1, *fp2, *fp3, *fp4;
    fp1 = fopen("output.txt", "w");
    fp2 = fopen("symtab.txt", "r");
    fp3 = fopen("intermediate.txt", "r");
    fp4 = fopen("objcode.txt", "w");
    fscanf(fp3, "%s\t%s\t%s", label, opcode, operand);
    while (strcmp(opcode, "END") != 0)
    {
        prevaddr = address;
        fscanf(fp3, "%d%s%s%s", &address, label, opcode, operand);
    }
    finaddr = address;
    fclose(fp3);
    fp3 = fopen("intermediate.txt", "r");
    fscanf(fp3, "\t%s\t%s\t%s", label, opcode, operand);
    if (strcmp(opcode, "START") == 0)
    {
        fprintf(fp1, "\t%s\t%s\t%s\n", label, opcode, operand);
        fprintf(fp4, "H^%s^00%s^00%d\n", label, operand, finaddr-atoi(operand));
        fscanf(fp3, "%d%s%s%s", &address, label, opcode, operand);
        start = address;
        diff = prevaddr - start;
        fprintf(fp4, "T^00%d^%d", address, diff);
    }

    while (strcmp(opcode, "END") != 0)
    {
        if (strcmp(opcode, "BYTE") == 0)
```

```
2021    CHARZ       BYTE        C'Z'        5A
2022    C1          RESB        1
2023    **          END         **
```

The contents of Object code file :

```
H^**^002000^0023
T^002000^22^332018^442012^532021^572022^000005^5A
E^002000
```

```
    {
        fprintf(fp1, "%d\t%s\t%s\t%s\t", address, label, opcode, operand);
        len = strlen(operand);
        actual_len = len - 3;
        fprintf(fp4, "^");
        for (i = 2; i < (actual_len + 2); i++)
        {
            itoa(operand[i], ad, 16);
            fprintf(fp1, "%s", ad);
            fprintf(fp4, "%s", ad);
        }
        fprintf(fp1, "\n");
    }
    else if (strcmp(opcode, "WORD") == 0)
    {
        len = strlen(operand);
        itoa(atoi(operand), a, 10);
        fprintf(fp1, "%d\t%s\t%s\t%s\t00000%s\n", address, label, opcode, operand, a);
        fprintf(fp4, "^00000%s", a);
    }
    else if ((strcmp(opcode, "RESB") == 0) || (strcmp(opcode, "RESW") == 0)) {
        fprintf(fp1, "%d\t%s\t%s\t%s\n", address, label, opcode, operand);
    }
    else
    {
        while (strcmp(opcode, mnemonic[j]) != 0)
            j++;
        if (strcmp(operand, "COPY") == 0)
            fprintf(fp1, "%d\t%s\t%s\t%s\t%s0000\n", address, label, opcode, operand, code[j]);
        else
        {
            rewind(fp2);
            fscanf(fp2, "%s%d", symbol, &add);
            while (strcmp(operand, symbol) != 0)
                fscanf(fp2, "%s%d", symbol, &add);
            fprintf(fp1, "%d\t%s\t%s\t%s\t%s%d\n", address, label, opcode, operand, code[j],
add);
            fprintf(fp4, "^%s%d", code[j], add);
```

```
            }
        }
        fscanf(fp3, "%d%s%s%s", &address, label, opcode, operand);
    }
    fprintf(fp1, "%d\t%s\t%s\t%s\n", address, label, opcode, operand);
    fprintf(fp4, "\nE^00%d", start);
    fclose(fp4);
    fclose(fp3);
    fclose(fp2);
    fclose(fp1);
    display();
    return 0;
}
void display() {
    char ch;
    FILE *fp1, *fp2, *fp3, *fp4;
    printf("\nIntermediate file is converted into object code");
    printf("\n\nThe contents of Intermediate file:\n\n");
    fp3 = fopen("intermediate.txt", "r");
    ch = fgetc(fp3);
    while (ch != EOF)
    {
        printf("%c", ch);
        ch = fgetc(fp3);
    }
    fclose(fp3);
    printf("\n\nThe contents of Symbol Table :\n\n");
    fp2 = fopen("symtab.txt", "r");
    ch = fgetc(fp2);
    while (ch != EOF)
    {
        printf("%c", ch);
        ch = fgetc(fp2);
    }
    fclose(fp2);
    printf("\n\nThe contents of Output file :\n\n");
    fp1 = fopen("output.txt", "r");
    ch = fgetc(fp1);
```

```
  while (ch != EOF)
  {
     printf("%c", ch);
     ch = fgetc(fp1);
  }
  fclose(fp1);
  printf("\n\nThe contents of Object code file :\n\n");
  fp4 = fopen("objcode.txt", "r");
  ch = fgetc(fp4);
  while (ch != EOF)
  {
     printf("%c", ch);
     ch = fgetc(fp4);
  }
  fclose(fp4);
}
```

RESULT

C program for the implementation of pass 2 of a two pass assembler is executed successfully .

DATE :

EXPERIMENT NO – 2.3
IMPLEMENT A SINGLE PASS MACROPROCESSOR

AIM

To implement a single pass macroprocessor .

ALGORITHM

1. Start the program
2. Open files: **input.txt**, **namtab.txt**, **deftab.txt**, **argtab.txt**, and **op.txt** for reading and writing.
3. Read the input file **input.txt** and process lines until encountering **END**.
4. If the line contains **MACRO**, extract the macro name **la** and its operands **opnd**.
   - Write the macro name to **namtab.txt**.
   - Write the macro name and operands to **deftab.txt**.
   - Process lines until **MEND** is encountered:
     - If the operand starts with **&**, replace it with **?** followed by a positional number (**pos1**).
     - Write the macro instructions and modified operands to **deftab.txt**.

5. If not a macro definition:
   - Compare the line's mnemonic **mne** with names in **namtab.txt**.
   - If a match is found, extract and format operands from the line to **argtab.txt**.
   - Seek to the beginning of **deftab.txt** and **argtab.txt**.
   - Read the macro instructions and operands:
     - Write the macro name and operands to **op.txt**.
     - Process lines until encountering **MEND**:
       - If the operand starts with **?**, substitute it with arguments from **argtab.txt**.
       - Write substituted instructions and operands to **op.txt**.
6. Close all opened files and print a success message.
7. End the program .

INPUT

input.txt

```
PGM    START      0
ABC    MACRO      &A,&B
-      STA        &A
-      STB        &B
-      MEND        -
-      ABC        P,Q
-      ABC        R,S
P      RESW       1
Q      RESW       1
R      RESW       1
S      RESW       1
-      END         -
```

OUTPUT

namtab.txt

```
ABC
```

argtab.txt

```
R
S
```

deftab.txt

```
ABC    &A,&B
STA    ?
STB    ?
MEND
```

op.txt

```
PGM    START      0
.      ABC        P,Q
-      STA        ?
-      STB        ?
.      ABC        R,S
```

PROGRAM

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void main()
{
FILE *f1,*f2,*f3,*f4,*f5;
int len,i,pos=1;
char
arg[20],mne[20],opnd[20],la[20],name[20],
mne1[20],opnd1[20],pos1[10],pos2[10];
f1=fopen("input.txt","r");
f2=fopen("namtab.txt","w+");
f3=fopen("deftab.txt","w+");
f4=fopen("argtab.txt","w+");
f5=fopen("op.txt","w+");
fscanf(f1,"%s%s%s",la,mne,opnd);
while(strcmp(mne,"END")!=0)
{
if(strcmp(mne,"MACRO")==0)
{
fprintf(f2,"%s\n",la);
fseek(f2,SEEK_SET,0);
fprintf(f3,"%s\t%s\n",la,opnd);
fscanf(f1,"%s%s%s",la,mne,opnd);
while(strcmp(mne,"MEND")!=0)
{
if(opnd[0]=='&')
{
(pos,pos1,5);
strcpy(pos2,"?");
strcpy(opnd,strcat(pos2,pos1));
pos=pos+1;
}
fprintf(f3,"%s\t%s\n",mne,opnd);
fscanf(f1,"%s%s%s",la,mne,opnd);
}
fprintf(f3,"%s",mne);
```

```
-       STA       ?
-       STB       ?
P       RESW      1
Q       RESW      1
R       RESW      1
S       RESW      1
-       END       -
```

TERMINAL

```
Successfull !!!
```

```
}
else
{
fscanf(f2,"%s",name);
if(strcmp(mne,name)==0)
{
len=strlen(opnd);
for(i=0;i<len;i++)
{
if(opnd[i]!=',')
fprintf(f4,"%c",opnd[i]);
else
fprintf(f4,"\n");
}
fseek(f3,SEEK_SET,0);
fseek(f4,SEEK_SET,0);
fscanf(f3,"%s%s",mne1,opnd1);
fprintf(f5,".\t%s\t%s\n",mne1,opnd);
fscanf(f3,"%s%s",mne1,opnd1);
while(strcmp(mne1,"MEND")!=0)
{
if((opnd[0]=='?'))
{
fscanf(f4,"%s",arg);
fprintf(f5,"-\t%s\t%s\n",mne1,arg);
}
else
fprintf(f5,"-\t%s\t%s\n",mne1,opnd1);
fscanf(f3,"%s%s",mne1,opnd1);
}
}
else
fprintf(f5,"%s\t%s\t%s\n",la,mne,opnd);
}
fscanf(f1,"%s%s%s",la,mne,opnd);
}
fprintf(f5,"%s\t%s\t%s",la,mne,opnd);
fclose(f1);
```

```
fclose(f2);
fclose(f3);
fclose(f4);
fclose(f5);
printf("Successfull !!! \n");
}
```

RESULT

C program for the implementation of one pass macroprocessor has been implemented successfully .

DATE :

EXPERIMENT NO – 2.4
IMPLEMENT AN ABSOLUTE LOADER

AIM

To implement an absolute loader .

ALGORITHM

1. **Include Necessary Libraries and Declare Variables**
   - ➢ Include the standard libraries **<stdio.h>** and **<string.h>**.
   - ➢ Declare variables:
     - **input[10]**, **label[10]**: Arrays to store input and label.
     - **addr**, **start**, **ptaddr**, **l**, **length**, **end**, **count**, **k**, **taddr**, **address**, **i**: Integer variables used for memory address manipulation and counting.
     - **ch1**, **ch2**: Characters used for file processing.
     - **fp1**, **fp2**: File pointers for input and output files.
2. **Function Declaration**
   - ➢ **check()**: A function to perform checks and manipulate file pointers and counters.
3. **Main Function (main()):**
   - ➢ Open the input file ("input.txt") in read mode (**"r"**) and the output file ("output.txt") in write mode (**"w"**).
   - ➢ Read the first string from the input file and display a header for the loader.
   - ➢ Write the header for the output file ("MEMORY ADDRESS", "CONTENTS").
   - ➢ Enter a loop that continues until the input is not "E":
     - Check if the input is "H":
       - o Read label, start address, end address, and the next input string from the file.
       - o Set the memory address (**address**) to the start address.
     - Otherwise, if the input is "T":
       - o Store the current length and program text address (**l** and **ptaddr**).
       - o Read target address, length, and input from the file.
       - o Update the memory address (**address**) and check for gaps in memory.
       - o If **w** is 0, set **ptaddr** to **address**.
       - o Write data to the output file in a specified format, update counters, and perform checks.
     - If the input is neither "H" nor "T":

INPUT

input.txt

```
H COPY 001000 00107A
T 001000 1E 141033 482039 001036 281030 301015 482061 3C1003 00102A 0C1039 00102D
T 00101E 15 0C1036 482061 081033 4C0000 454F46 000003 000000
T 001047 1E 041030 001030 E0205D 30203F D8205D 281030 302057 549039 2C205E 38203F
T 001077 1C 101036 4C0000 000000 001000 041030 E02079 302064 509039 DC2079 2C1036
E 001000
```

OUTPUT

```
ABSOLUTE LOADER
 The contents of output file:
MEMORY ADDRESS                    CONTENTS

1000                   14103348  20390010  36281030  30101548

1010                   20613C10  0300102A  0C103900  102D0C10

1020                   36482061  0810334C  0000454F  46000003

1030                   000000xx  xxxxxxxx  xxxxxxxx  xxxxxxxx

1040                   xxxxxxxx  xxxxxx04  10300010  30E0205D

1050                   30203FD8  205D2810  30302057  5490392C

1060                   205E3820  3Fxxxxxx  xxxxxxxx  xxxxxxxx

1070                   xxxxxxxx  xxxxxx10  10364C00  00000000

1080                   00100004  1030E020  79302064  509039DC

1090                   20792C10  36
```

        - Write data to the output file in a specified format, update counters, and perform checks.
    - Read the next input from the file.
  - Close both input and output files.
  - Display the contents of the output file on the console by opening "output.txt" in read mode and printing its contents.

4. **check() Function:**
   - Increment various counters and memory addresses (**count**, **address**, **taddr**) based on certain conditions.
   - Write appropriate formatting to the output file (**fp2**).
5. End of program

PROGRAM

```
#include <stdio.h>
#include <string.h>
char input[10], label[10], ch1, ch2;
int addr, w = 0, start, ptaddr, l, length = 0, end, count = 0, k, taddr, address, i = 0;
FILE *fp1, *fp2;
void check();
void main()
{
        fp1 = fopen("input.txt", "r");
        fp2 = fopen("output.txt", "w");
        fscanf(fp1, "%s", input);
        printf("\n\nABSOLUTE LOADER\n");
        fprintf(fp2, "MEMORY ADDRESS\t\t\tCONTENTS");
        while (strcmp(input, "E") != 0)
        {
                if (strcmp(input, "H") == 0)
                {
                        fscanf(fp1, "%s %x %x %s", label, &start, &end, input);
                        address = start;
                }
                else if (strcmp(input, "T") == 0)
                {
                        l = length;
                        ptaddr = addr;
                        fscanf(fp1, "%x %x %s", &taddr, &length, input);
```

```
        addr = taddr;
        if (w == 0)
        {
            ptaddr = address;
            w = 1;
        }
        for (k = 0; k < (taddr - (ptaddr + l)); k++)
        {
            address = address + 1;
            fprintf(fp2, "xx");
            count++;
            if (count == 4)
            {
                fprintf(fp2, "  ");
                i++;
                if (i == 4)
                {
                    fprintf(fp2, "\n\n%x\t\t", address);
                    i = 0;
                }
                count = 0;
            }
        }
        if (taddr == start)
            fprintf(fp2, "\n\n%x\t\t", taddr);
        fprintf(fp2, "%c%c", input[0], input[1]);
        check();
        fprintf(fp2, "%c%c", input[2], input[3]);
        check();
        fprintf(fp2, "%c%c", input[4], input[5]);
        check();
        fscanf(fp1, "%s", input);
    }
    else
    {
        fprintf(fp2, "%c%c", input[0], input[1]);
        check();
        fprintf(fp2, "%c%c", input[2], input[3]);
```

```
                        check();
                        fprintf(fp2, "%c%c", input[4], input[5]);
                        check();
                        fscanf(fp1, "%s", input);
                }
        }
        fclose(fp1);
        fclose(fp2);
        printf("\n\n The contents of output file:\n");
        fp2 = fopen("output.txt", "r");
        ch2 = fgetc(fp2);
        while (ch2 != EOF)
        {
                printf("%c", ch2);
                ch2 = fgetc(fp2);
        }
        fclose(fp2);
}
void check()
{
        count++;
        address++;
        taddr = taddr + 1;
        if (count == 4)
        {
                fprintf(fp2, "  ");
                i++;
                if (i == 4)
                {
                        fprintf(fp2, "\n\n%x\t\t", taddr);
                        i = 0;
                }
                count = 0;
        }
}
```

RESULT

C program for the implementation of an absolute loader has been implemented successfully .

DATE :

EXPERIMENT NO – 2.5
IMPLEMENT A RELOCATING LOADER

AIM

To implement a relocating loader .

ALGORITHM

1. Start the program
2. **Include Necessary Libraries and Declare Variables**
   - Include standard libraries: **<stdio.h>**, **<string.h>**, **<stdlib.h>**.
   - Declare variables:
     - **bit[30]**: Array to store bits converted from the bitmask.
     - **bitmask[20]**: Array to store the bitmask.
     - **objptr**: File pointer for reading input from "relinput.txt".
     - **start**, **addr**: Integer variables to store the starting address and current address.
     - **rec[20]**: Array to store a record from the input file.
     - **name[20]**: Array to store the program name.
     - **modif_obj_code**: Integer variable to hold modified object code.
     - **first[3]**, **second[5]**: Arrays to parse the object code into parts.
     - **bitmask_index**, **i**, **add**, **len**: Integer variables used for indexing and looping.
3. **Function bitmask_convert(char mask[]):**
   - Converts the given bitmask into bits and stores the result in the **bit[]** array.
4. **Main Function (main()):**
   - Prompt the user to enter the starting address of the program.
   - Read the starting address from the user input.
   - Open the input file ("relinput.txt") for reading.
   - Read the first record from the file.
   - Check if the record is a header record ("H"):
     - Read the program name, starting address, and length.
     - Display a header for the output ("ADDRESS OBJECT CODE").
   - If the record is not a header record, display an error message and exit the program.
   - Read the next record from the file.
   - Enter a loop until the record is not "E":
     - If the record is "T":

INPUT

relinput.txt

```
H COPY 000000 00107A
T 000000 1E FFC 140033 481039 100036 280030 300015 481061 3C0003 20002A 1C0039
30002D
T 002500 15 E00 1D0036 481061 180033 4C1000 801000 601003
E 000000
```

OUTPUT

```
ENTER THE STARTING ADDRESS OF THE PROGRAM
2000
 ADDRESS   OBJECT CODE
______________________________
2000    142033
2003    483039
2006    102036
2009    282030
200C    302015
200F    483061
2012    3C2003
2015    20202A
2018    1C2039
201B    30202D
4500    1D2036
4503    483061
4506    182033
4509    4C1000
450C    801000
450F    601003
```

        - Read the address, length, and bitmask.
        - Convert the bitmask to bits using the **bitmask_convert()** function.
        - Read the next record.
    - If the bit at **bitmask_index** is '1':
        - Extract parts of the object code and convert to modified object code.
        - Print the modified address and object code.
    - If the bit is '0':
        - Print the address and object code without modification.
    - Increment the address and bitmask_index.
    - Read the next record.

5. **Close File and Terminate**
    - Close the input file.
    - Terminate the program.

PROGRAM

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
char bit[30];
char bitmask[20];
void bitmask_convert(char mask[])
{
  int len;
  len = strlen(mask);
  strcpy(bit, "");
  int i;
  for (i = 0; i < len; ++i){
    switch (mask[i])
    {
    case '0':
      strcat(bit, "0000");
      break;
    case '1':
      strcat(bit, "0001");
      break;
    case '2':
      strcat(bit, "0010");
```

```
   break;
case '3':
   strcat(bit, "0011");
   break;
case '4':
   strcat(bit, "0100");
   break;
case '5':
   strcat(bit, "0101");
   break;
case '6':
   strcat(bit, "0110");
   break;
case '7':
   strcat(bit, "0111");
   break;
case '8':
   strcat(bit, "1000");
   break;
case '9':
   strcat(bit, "1001");
   break;
case 'A':
   strcat(bit, "1010");
   break;
case 'B':
   strcat(bit, "1011");
   break;
case 'C':
   strcat(bit, "1100");
   break;
case 'D':
   strcat(bit, "1101");
   break;
case 'E':
   strcat(bit, "1110");
   break;
case 'F':
```

```
            strcat(bit, "1111");
            break;
         default:
            break;
         }
      }
   }
   void main(){
      FILE *objptr;
      int start, addr;
      char rec[20];
      char name[20];
      int modif_obj_code;
      char first[3];
      char second[5];
      int bitmask_index = 0;
      int i;
      int add, len;
      printf("ENTER THE STARTING ADDRESS OF THE PROGRAM\n");
      scanf("%X", &start);
      addr = start;
      objptr = fopen("relinput.txt", "r");
      fscanf(objptr, "%s", rec);
      if (strcmp(rec, "H") == 0)
      {
         fscanf(objptr, "%s", name);
         fscanf(objptr, "%X", &add);
         fscanf(objptr, "%X", &len);
         printf(" ADDRESS   OBJECT CODE \n");
         printf("______________________________\n");
      }
      else
      {
         printf("INAVLID OBJECT CODE FORMAT\n");
         fclose(objptr);
         exit(1);
      }
```

```
strcpy(rec, "");
fscanf(objptr, "%s", rec);
while (strcmp(rec, "E") != 0)
{
   if (strcmp(rec, "T") == 0)
   {
      fscanf(objptr, "%X", &add);
      fscanf(objptr, "%X", &len);
      fscanf(objptr, "%s", bitmask);
      add += start;
      bitmask_index = 0;
      bitmask_convert(bitmask);
      fscanf(objptr, "%s", rec);
   }
   if (bit[bitmask_index] == '1')
   {
      for (i = 0; i < 6; ++i)
      {
         if (i < 2)
         {
            first[i] = rec[i];
         }
         else
         {
            second[i - 2] = rec[i];
         }
      }
      first[2] = '\0';
      second[4] = '\0';
      modif_obj_code = strtol(second, NULL, 16);
      modif_obj_code += start;
      printf("%X\t%s%X\n", add, first, modif_obj_code);
   }
   else
   {
      printf("%X\t%s\n", add, rec);
   }
   add += 3;
```

```
        bitmask_index++;
        fscanf(objptr, "%s", rec);
    }
    fclose(objptr);
}
```

RESULT

C program for the implementation of a relocating loader has been implemented successfully .