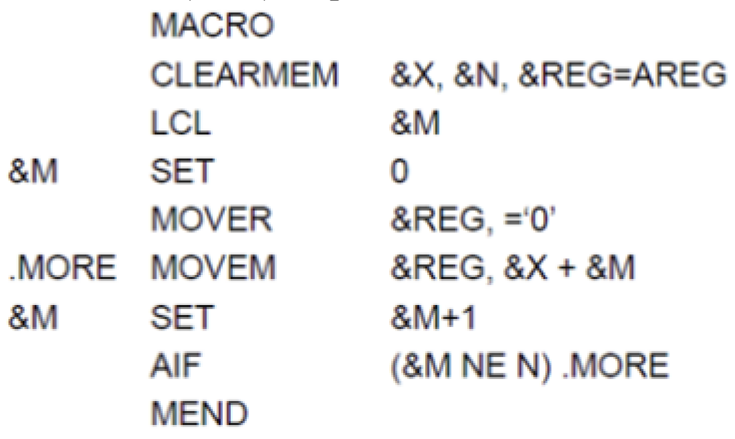
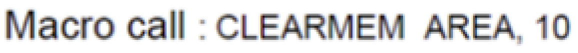
Q:Generate Macro Definition Table(MDT) for given macro definition:



```
        MACRO
        CLEARMEM    &X, &N, &REG=AREG
        LCL         &M
&M      SET         0
        MOVER       &REG, ='0'
.MORE   MOVEM       &REG, &X + &M
&M      SET         &M+1
        AIF         (&M NE N) .MORE
        MEND
```



Macro call : CLEARMEM AREA, 10

CODE:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

typedef struct MNT {
char name[20];
int pp;
int kp;
int ev;
int mdtp;
int kpdtp;
int sstp;
} MNT;

typedef struct MDT {
int index;
char label[20];
char opcode[20];
char operands[100];
} MDT;

typedef struct EVNTAB {
int index;
char name[20];
} EVNTAB;

typedef struct SSNTAB {
int index;
char name[20];
} SSNTAB;
```

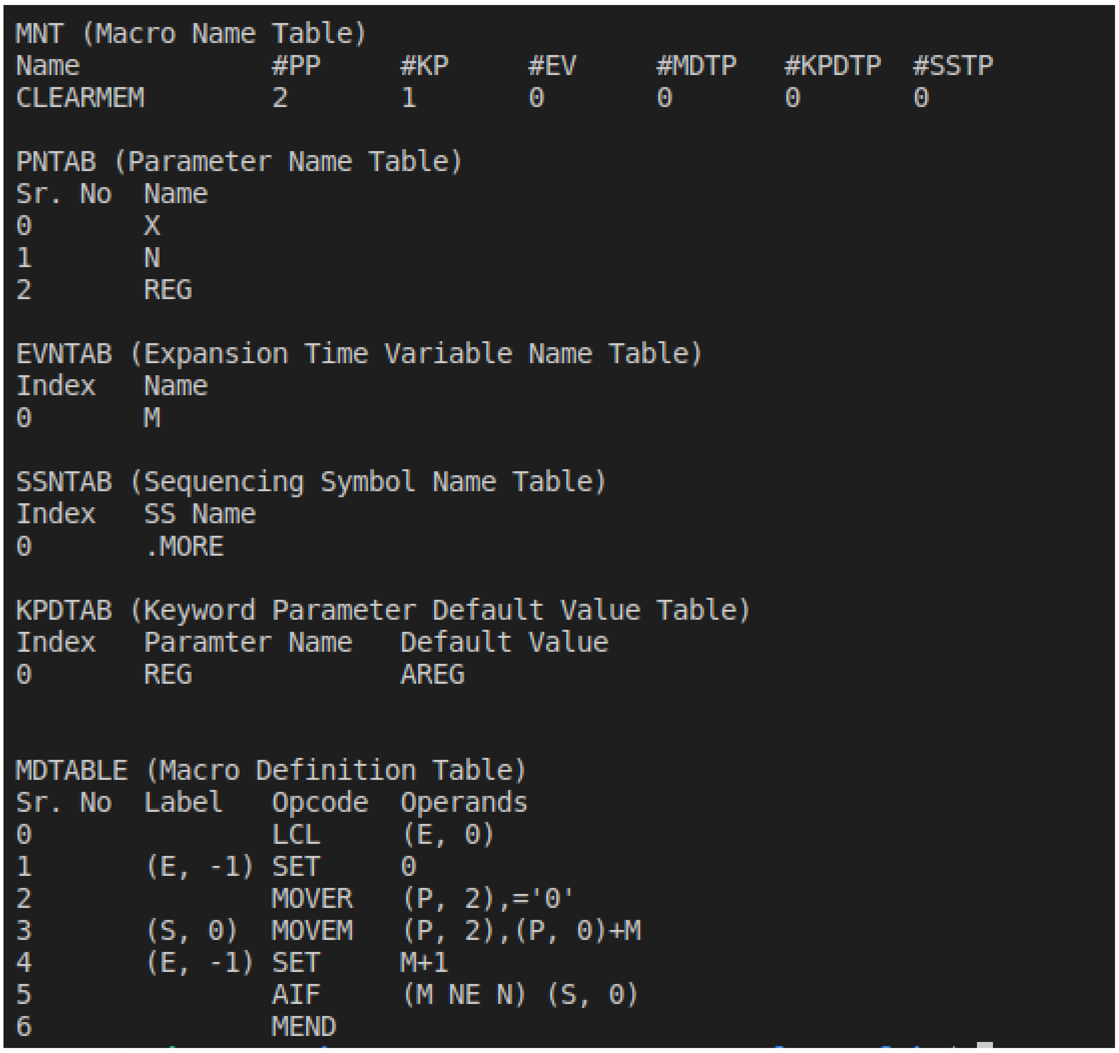
```
typedef struct PNTAB {
int index;
char name[20];
} PNTAB;

typedef struct KPDTAB {
int index;
char name[20];
char default_value[20];
} KPDTAB;

MNT mnt[10];
MDT mdtable[20];
EVNTAB evntab[20];
SSNTAB ssntab[20];
PNTAB pntab[20];
KPDTAB kpdtab[20];

int getSS(char* ss) {
int i;
for (i = 0; i < 20; i++) {
if (strcmp(ssntab[i].name, ss) == 0)
return i;
}
return -1;
}

int getEV(char* ev) {
int i;
for (i = 0; i < 20; i++) {
if (strcmp(evntab[i].name, ev) == 0)
return i;
}
return -1;
}

int getParam(char* p) {
int i;
for (i = 0; i < 20; i++) {
if (strcmp(pntab[i].name, p) == 0)
return i;
}
return -1;
}
int getName(char* name, char* buffer, int i) {
int j = i;
if (buffer[i] == '.')
```

```
j++;
while(isalpha(buffer[j])) {
j++;
}
strncpy(name, buffer+i, j - i);
name[j-i] = '\0';
return j;
}

int mntc = 0, mdtc = 0, evntc = 0, ssntc = 0, pntc = 0, kpdtc = 0;

void getInstruction(char* buffer) {
char label[20], opcode[20], operands[100], temp[20];
strcpy(label, strtok(buffer, " "));
if (label[0] == '.'){
ssntab[ssntc].index = ssntc;
strcpy(ssntab[ssntc].name, label);
sprintf(mdtable[mdtc].label, "(S, %d)", ssntc);
ssntc++;
strcpy(opcode, strtok(NULL, " "));
}
else if (label[0] == '&') {
int ev = getEV(label+1);
sprintf(mdtable[mdtc].label, "(E, %d)", ev);
strcpy(opcode, strtok(NULL, " "));
}
else {
strcpy(opcode, label);
strcpy(mdtable[mdtc].label, "");
}
strcpy(mdtable[mdtc].opcode, opcode);
strcpy(operands, strtok(NULL, ""));
operands[strlen(operands)-1] = '\0';
if (strcmp(opcode, "LCL") == 0 || strcmp(opcode, "GBL") == 0) {
evntab[evntc].index = evntc;
strcpy(evntab[evntc].name, operands+1);
sprintf(mdtable[mdtc].operands, "(E, %d)", evntc);
evntc++;
}
else {
int i = 0;
while (operands[i] != '\0') {
if (operands[i] == '&') {
i = getName(temp, operands, i+1);
int param = getParam(temp);
int ev = getEV(temp);
if (param >= 0) {
sprintf(temp, "(P, %d)", param);
```

```
strcat(mdtable[mdtc].operands, temp);
}
else if (ev >= 0) {
sprintf(temp, "(E, %d)", ev);
strcat(mdtable[mdtc].operands, temp);
}
else {
strcat(mdtable[mdtc].operands, temp);
}
}
else if (operands[i] == '.') {
i = getName(temp, operands, i);
int ss = getSS(temp);
sprintf(temp, "(S, %d)", ss);
strcat(mdtable[mdtc].operands, temp);
}
else {
sprintf(mdtable[mdtc].operands, "%s%c", mdtable[mdtc].operands, operands[i++]);
}
}
}
mdtable[mdtc].index = mdtc;
mdtc++;
}

void main() {
FILE *in, *mdt;

in = fopen("input.txt", "r");

char buffer[200];
while (fgets(buffer, 200, in)) {
if (strstr(buffer, "MACRO")) {
fgets(buffer, 200, in);
strcpy(mnt[mntc].name, strtok(buffer, " "));
mnt[mntc].mdtp = mdtc;
mnt[mntc].kpdtp = kpdtc;
mnt[mntc].sstp = ssntc;
char* temp;
while(temp = strtok(NULL, ", ")) {
char *param;
if (param = strchr(temp, '=')) {
mnt[mntc].kp++;
strcpy(kpdtab[kpdtc].default_value, param+1);
strncpy(kpdtab[kpdtc].name, temp + 1, strlen(temp) - strlen(param) - 1);
kpdtab[kpdtc].name[strlen(temp) - strlen(param) - 1] = '\0';
kpdtab[kpdtc].index = kpdtc;
strcpy(pntab[pntc].name, kpdtab[kpdtc].name);
```

```
pntab[pntc].index = pntc;
kpdtc++;
pntc++;
} else {
mnt[mntc].pp++;
strcpy(pntab[pntc].name, temp + 1);
pntab[pntc].index = pntc;
pntc++;
}
}
mntc++;
while (fgets(buffer, 200, in)) {
if (strstr(buffer, "MEND")) {
strcpy(mdtable[mdtc].opcode, "MEND");
mdtable[mdtc].index = mdtc;
mdtc++;
break;
}
getInstruction(buffer);
}
}
}

fclose(in);

// Macro Name Table
printf("\nMNT (Macro Name Table)\n");
printf("Name\t\t#PP\t#KP\t#EV\t#MDTP\t#KPDTP\t#SSTP\n");
for (int i = 0; i < mntc; i++) {
printf("%s\t%d\t%d\t%d\t%d\t%d\t%d\n", mnt[i].name, mnt[i].pp, mnt[i].kp, mnt[i].ev,
mnt[i].mdtp, mnt[i].kpdtp, mnt[i].sstp);
}

// Parameter Name Table
printf("\nPNTAB (Parameter Name Table)\n");
printf("Sr. No\tName\n");
for (int i = 0; i < pntc; i++) {
printf("%d\t%s\n", pntab[i].index, pntab[i].name);
}

//Expansion Time Variable Name Table
printf("\nEVNTAB (Expansion Time Variable Name Table)\n");
printf("Index\tName\n");
for (int i = 0; i < evntc; i++) {
printf("%d\t%s\n", evntab[i].index, evntab[i].name);
}

// Sequencing Symbol Table
```

```
printf("\nSSNTAB (Sequencing Symbol Name Table)\n");
printf("Index\tSS Name\n");
for (int i = 0; i < ssntc; i++) {
printf("%d\t%s\n", ssntab[i].index, ssntab[i].name);
}

// Keyword Parameter Default Value Table
printf("\nKPDTAB (Keyword Parameter Default Value Table)\n");
printf("Index\tParamter Name\tDefault Value\n");
for (int i = 0; i < kpdtc; i++) {
printf("%d\t%s\t\t%s\n", kpdtab[i].index, kpdtab[i].name, kpdtab[i].default_value);
}

// Macro Definition Table
printf("\nMDTABLE (Macro Definition Table)\n");
printf("Sr. No\tLabel\tOpcode\tOperands\n");
for (int i = 0; i < mdtc; i++) {
printf("%d\t%s\t%s\t%s\n", mdtable[i].index, mdtable[i].label, mdtable[i].opcode,
mdtable[i].operands);
}
}
```



```
MNT (Macro Name Table)
Name            #PP     #KP     #EV     #MDTP   #KPDTP  #SSTP
CLEARMEM        2       1       0       0       0       0

PNTAB (Parameter Name Table)
Sr. No  Name
0       X
1       N
2       REG

EVNTAB (Expansion Time Variable Name Table)
Index   Name
0       M

SSNTAB (Sequencing Symbol Name Table)
Index   SS Name
0       .MORE

KPDTAB (Keyword Parameter Default Value Table)
Index   Paramter Name   Default Value
0       REG             AREG


MDTABLE (Macro Definition Table)
Sr. No  Label   Opcode  Operands
0               LCL     (E, 0)
1       (E, -1) SET     0
2               MOVER   (P, 2),='0'
3       (S, 0)  MOVEM   (P, 2),(P, 0)+M
4       (E, -1) SET     M+1
5               AIF     (M NE N) (S, 0)
6               MEND
```