```
#include <iostream>
#include <cstdlib>
#include <stdlib.h>
#include <stdio.h>
#include <cstring>
#include <string>
#include <sstream>
#include <fstream>
using namespace std;
/**********************************************************************************************************
*                                       LINKER                                                            *
**********************************************************************************************************/

const size_t MAX_MEM=600;
const size_t MAX_SYMBOLS=200;
const size_t MAX_ND=100;
const size_t MAX_NU=100;
struct symbols {
                  char sym_text[8];     //Symbol up to 8 characters
                  int address;          //Address (abs) of symbol
                  char comment[60];     //Warnings or errors on symbols
                  int def_in;           //module the symbol is defined in
                  int last_used;        //module where the symbol was last used
               };
struct symbols symbol_tbl[MAX_SYMBOLS]; //Up to MAX_SYMBOLS  total in all modules
int s1=0;                               //symbol table index

struct memory  {
                  int comm_num;
                  char adtype;
                  char origad[4];
                  char spc[5];
                  char resolve_ext[8];
                  char command[4];
                  char *error;
               };

struct memory  memory_map[MAX_MEM];
```

```
//Warning definitions
struct node  {
               char        w_data[80];
               struct node *next;
             };

struct node *warning_start=NULL;
bool   warn_f, err_f;

//function prototypes

void init_arrays(void);
bool warn(char *);                     //create warnings list
void print_warn(struct node *);        //print warning list
int lookup(char s[],int);              //Look up a symbol
bool ind(int, int, char *);            //Array index limit check

int main (int argc, char * const argv[]) {

fstream  f(argv[1], ios::in);          //Input file

//int loc=0;                             // location in the input
char tk[8]; char current_sym[8];
//char token[MAX_MEM][8];
int base=0;                            //module base address
int module=1;                          //module number in input
struct use     {
                 char reference[8];
                   bool used;
               } use_list[MAX_NU];
struct inst    {
                  char ad_type;
                    char add[4];
               } instructions[MAX_MEM];

int            module_abs_ad[100];      // up to 100 modules per input file
char           nd_list[MAX_ND][8];      // up to 100 symbol def per module
```

```
int            nd=0, nu=0, u2=0;
int            n1=0,  displacement=0;
int            temp_add=0, curr_add=0;
char           module_c[4];
char           warning_str[80]="";
bool           v;

//Initialize arrays
init_arrays();

//PASS1 - Has two basic functions. (1) Build the Symbol table
//        and (2) calculate the Base addresses of all modules

f >> tk;                                //first token from input
while(f){
   //process a module
     nd=atoi(tk);                         //definition list size
     v=ind(nd,MAX_ND, "ND list");
    //++loc;

   //build symbol table from definition list
   for( int i1=0; i1<nd; ++i1){
      f >> tk;
       strcpy(current_sym, tk);
       f >> tk;
       curr_add=atoi(tk);

       //Check if the current symbol was already defined ==> Multiply defined (1)
       bool sym_used = false;
       for (int chk=0; chk<s1; ++chk){
         if (!strcmp(symbol_tbl[chk].sym_text,current_sym)){
         strcpy(symbol_tbl[chk].comment,
         " Error: This value was multiply defined. First value used");
         sym_used = true;

      }
       }
       if(!sym_used){                     //Add to the symbol table if not defined before
```

```
             if (s1>199) { cout << "\n Too many symbols, overriden last entry"; s1=199;}
            strcpy(symbol_tbl[s1].sym_text,current_sym);
            symbol_tbl[s1].address=base + curr_add;
            symbol_tbl[s1].def_in=module;
            ++s1;
       }
   }

   f >> tk;                               //Use list counter
   nu=atoi(tk);
   v=ind(nu,MAX_NU, "NU list");

   //read use list
   for(int i2=0;i2<nu;++i2){
      f >> tk;
   }
   // get module size
   f >> tk;
   int pgm_len=atoi(tk);
   v=ind(pgm_len,MAX_MEM, "Memory Map");

   //read module instructions, skip through them here, to get to the next module.
   for(int i3=0;i3<pgm_len*2;++i3){
      f >> tk;
   }

   module_abs_ad[module] = base;
   base+=pgm_len;
   ++module;
   f >> tk;                               //next module's ND
}

  //PASS 2 - Resolves external references & relative addresses

  f.close();
  fstream  g(argv[1], ios::in);
```

```
int next_command=0; module=1;
int pgm_len;
g >> tk;
while(g){
    nd=atoi(tk);                         //definition list size
   //Build def list
   for( int i1=0; i1<2*nd; ++i1){
       g >>tk;
    strcpy(nd_list[i1],tk);
     }

   g >> tk;
   nu=atoi(tk);
    // Build the USE list
   for(int ul=0; ul<nu; ++ul){
      g >> tk;
      strcpy(use_list[ul].reference,tk);
    use_list[ul].used=false;
   }

   g >> tk;
   pgm_len=atoi(tk);

   //Parse the program's text
   int num_add; div_t d; char strnum[4]; char error_str[60];
   for(int ic=0;ic<pgm_len;++ic){
      g >> instructions[ic].ad_type;   // A, E, R or I
       g >> instructions[ic].add;       // Op code + address format: Oddd

    memory_map[next_command].comm_num=next_command;
    memory_map[next_command].adtype=instructions[ic].ad_type;
    strcpy(memory_map[next_command].origad,instructions[ic].add);
       switch(instructions[ic].ad_type){
          case 'A': num_add=atoi(instructions[ic].add);
                 d=div(num_add,1000);
                       if(d.rem > MAX_MEM){
                          memory_map[next_command].error=
                            "Error: Absolute address exceed machine size; zero used.";
```

```
                  sprintf(strnum, "%d", d.quot*1000);
            strcpy(memory_map[next_command].command,strnum);
             }
             else{

                strcpy(memory_map[next_command].command,instructions[ic].add);
                strcpy(memory_map[next_command].resolve_ext," ");
           }
           break;
case 'I': strcpy(memory_map[next_command].command,instructions[ic].add);
             strcpy(memory_map[next_command].resolve_ext," ");
           break;

case 'R': num_add=atoi(instructions[ic].add);
          d=div(num_add,1000);
             if(d.rem > pgm_len){
                memory_map[next_command].error   =
                  "Error: Relative address exceeds module size; zero used.";
                  sprintf(strnum, "%d", d.quot*1000);
             }
             else{
                sprintf(strnum, "%d", d.quot*1000 + module_abs_ad[module]+temp_add);
             }
             strcpy(memory_map[next_command].command,strnum);
             strcpy(memory_map[next_command].resolve_ext," ");
             break;

 case 'E': num_add=atoi(instructions[ic].add);
           d=div(num_add,1000);
             if(d.rem >=nu){
                  memory_map[next_command].error=
                  "Error: External address exceeds length of use list; treated as immediate.";
                strcpy(memory_map[next_command].command,instructions[ic].add);  //as immediate
             }
             else {
                        displacement=lookup(use_list[d.rem].reference,module); //check symbol table
                        if (displacement < 0) {
                              strcpy(error_str, "Error: "); strcat(error_str,use_list[d.rem].reference );
```

```
                                          strcat(error_str, " is not defined; zero used. ");
                                          memory_map[next_command].error = error_str;
                                          displacement=0;
                                    }
                                    sprintf(strnum, "%d", d.quot*1000 + displacement);
                                    strcpy(memory_map[next_command].command,strnum);
                                    //strcpy(memory_map[next_command].point,"--> ");
                                    strcpy(memory_map[next_command].resolve_ext ,use_list[d.rem].reference);
                                    use_list[d.rem].used=true;
                          }
                          break;

             default : strcpy(memory_map[next_command].command,"unkn");  //Should not happen because
         } //end switch                                                      the input is sound.

      ++next_command;                                                   // of the entire input batch
      }   //instructions

       //Check if a symbol was on the use list but not used (6)
        for(u2=0; u2<nu;++u2){
          if(!use_list[u2].used){
            sprintf(module_c, "%d", module);
            strcpy(warning_str,"Warning: in Module  "); strcat(warning_str,module_c);
            strcat(warning_str,",  symbol: "); strcat(warning_str,use_list[u2].reference);
            strcat(warning_str, " is on USE list, but never used");
          warn_f=warn(warning_str);
       }
        }

        ++module;
        g >> tk;
}


     //Printing results
     //-----------------------------------------------------------------------------------------------------
         cout << "\n" << "Symbol Table:" <<
                     "\n=============\n";
         int sym=0;
         while(sym<s1){
            cout <<         "             " << symbol_tbl[sym].sym_text << "=" << symbol_tbl[sym].address <<
            symbol_tbl[sym].comment <<"\n";
            //" Last used in module: " << symbol_tbl[sym].last_used << "\n";
             ++sym;
         }


         cout << "\n" << "Memory MAP:" << "\n===========\n";
       for(int zu=0; zu< next_command; ++zu){
          printf("\n Command #: %3d %c %5s %5s %8s %4s %3s",memory_map[zu].comm_num,
                   memory_map[zu].adtype, memory_map[zu].origad,
                   memory_map[zu].spc, memory_map[zu].resolve_ext,
                     memory_map[zu].command, memory_map[zu].error);
          }

         //Check if warning is warranted. symbols were defined and not used
         //----------------------------------------------------------------

       for(n1=0;n1<s1;++n1){
          if(symbol_tbl[n1].last_used==0){
          sprintf(module_c, "%d", symbol_tbl[n1].def_in);
            strcpy(warning_str,"Warning: symbol   "); strcat(warning_str,symbol_tbl[n1].sym_text);
            strcat(warning_str,", is defined in module:"); strcat(warning_str,module_c);
            strcat(warning_str, ", but never used.");

            warn_f=warn(warning_str);
       }
        }

       if (warning_start!=NULL){
           print_warn(warning_start);  //Printing warnings start to end
        }

    return 0;
}//End of main
```

```
//FUNCTIONS
//---------------------------------------------------------------------------------------

//***************************************************************************************
int lookup(char s[],int mod){
   //s1 is a global var that keeps the next running index of the symbol table

   int j=0; bool not_found= true;
   while(j<s1 && not_found){
      if (!strcmp(symbol_tbl[j].sym_text,s)){
          not_found = false;
       }
      ++j;
   }
   if(!not_found){
       symbol_tbl[j-1].last_used=mod; //Module last used it
       return symbol_tbl[j-1].address;
   }
   else return -1;
}
//***************************************************************************************
void init_arrays(void){
   for(int l1=0;l1<MAX_SYMBOLS;++l1){
       strcpy(symbol_tbl[l1].sym_text," ");
              symbol_tbl[l1].address=0;
              symbol_tbl[l1].def_in=0;
              symbol_tbl[l1].last_used=0;
   }
   for(int l2=0;l2<MAX_MEM;++l2){
      memory_map[l2].comm_num=0;
       memory_map[l2].adtype=' ';
       strcpy(memory_map[l2].origad,"   ");
      strcpy(memory_map[l2].spc," ");
       strcpy(memory_map[l2].resolve_ext," ");
       strcpy(memory_map[l2].command," ");
       memory_map[l2].error = " ";
   }
```

```
}
//***************************************************************************************

bool warn(char *t1){
   //This warn function creates a linked list of warnings
   //warning_start always points to the first warning found.
   struct node* newnode =(struct node *) malloc(sizeof(struct node));
   if (newnode==NULL){ //memory allocation failed, print the warning istead of saving it.
        cout << "\n" << t1;
   }
   else {
      strcpy(newnode->w_data,t1);
      struct node* curr=warning_start;
      if (curr==NULL){
         warning_start=newnode;
          newnode->next=NULL;
      }
      else {
         while(curr->next !=NULL){
             curr=curr->next;
          }
          curr->next=newnode;
          newnode->next=NULL;
      }
   }
   return true;
}

//***************************************************************************************
void print_warn(struct node *e){
   struct node *curr=e;
   struct node *advance;
   cout <<"\n\n Warnings" <<
            "\n ========\n";
   while(curr !=NULL){
      cout << "\n" << curr->w_data;
       advance=curr->next;
       free(curr);
       curr=advance;
   }
}
//***************************************************************************************
//Check index validity
bool ind(int index, int limit,char *name){
   if (index <= limit){ return true;}
   else{
        cout << "\n Fatal error. Array index exceeds the limit" <<
            ". The limit is:" << limit << " The value is:" << index <<
              " array is: " << name << "\n";
      exit(1);
   }
}
//***************************************************************************************
// END OF SOURCE
```