A.obj

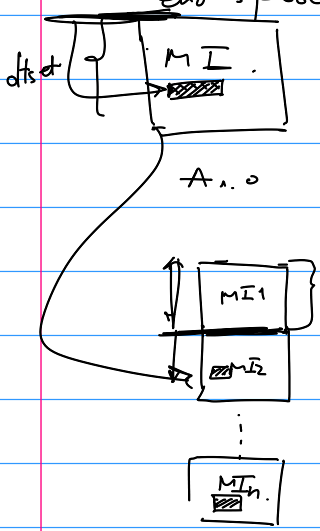
symbols

↑ n

↓ x

end symbol.

n : dw 0



A.c

bss ← int n;

extern int x;

I,

B.c

int n.

X.o

X.c

extern int n;

f() {

:

(n += 1;

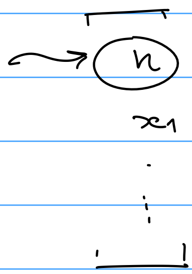
}

A1.o A2.o ... An.o → EXE

symbol Table

0xFAD109

0x12F3C



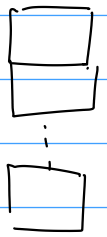
1. Вътрешна деф. символи. (I)

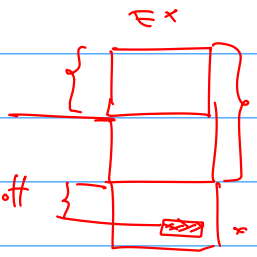
2. не деф. символ. (II)

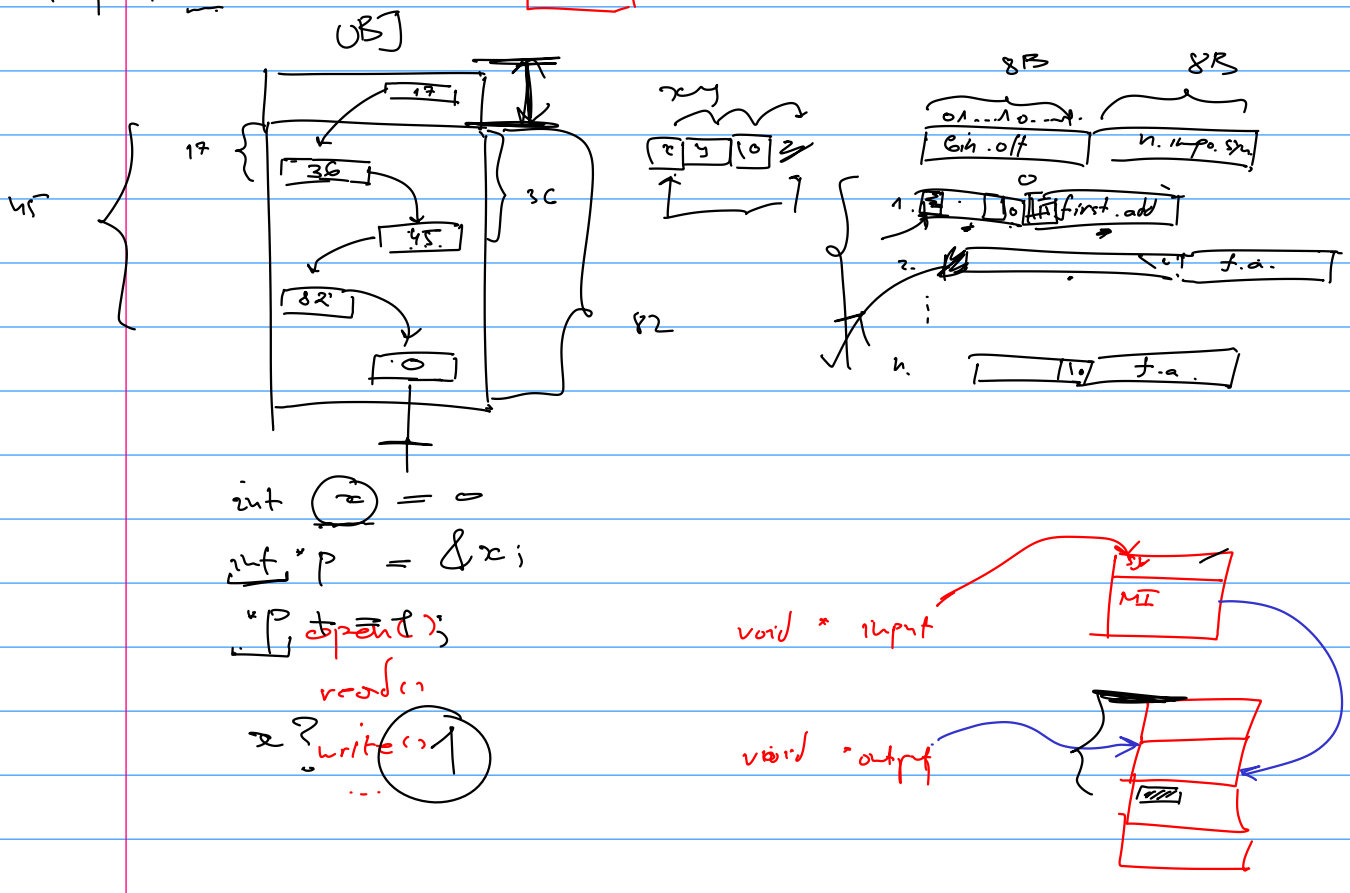
2. (10 poena)

Kod za prvi prolaz nekog linkera dat je u nastavku. (U prvom prolazu linker samo prikuplja izvezene simbole, a ne proverava i da li su svi uvezeni simboli i definisani; tu proveru radi prilikom obrade uvezenih simbola u drugom prolazu.) Popuniti izostavljene delove koda označene komentarima `/***1***/` do `/***5***/`.

```
void Linker::firstPass () {
  this.status = OK;
  this.binarySize = /***1***/; //BinarySize of processed output
  for (FileReader::reset(); !FileReader::isDone(); FileReader::next()) {
    ObjectFile* objFile = FileReader::currentObjectFile();
    if (objFile==0) {
      Output::error("Fatal internal error: null pointer exception.");
      exit(-1);
    }
    for (objFile.reset(); !objFile.isDone(); objFile.nextSymbol()) {
      Symbol* sym = objFile->getCurrentSymbol();
      if (sym==0) {
        /***2***/
      }
      if (sym->getKind() == Symbol::export) {
        int offset = sym->getOffset(); // offset in objFile
        offset += /***3***/;
        int status =
          SymbolTable::addSymDef(sym->getName(),offset,objFile->getName());
        if (status==-1) {
          /***4***/
          this.status = ERROR;
        }
      }
    }
    int size = objFile->getBinarySize();
    /***5***/
  }
}
```







2. (10 poena)

Potrebno je implementirati proceduru `resolveSymbols` koja se koristi u drugom prolazu jednog linkera. Ova procedura obrađuje jedan ulazni .obj fajl i treba da razreši adresna polja mašinskih instrukcija koja koriste simbole koje taj fajl uvozi. Ulazni i izlazni fajl su memorijski preslikani, tehnikom virtuelne memorije, tako da se njihov sadržaj može jednostavno posmatrati kao sadržaj memorije procesa. Na početak memorijski preslikanog sadržaja ulaznog .obj fajla ukazuje prvi, a na početak tog prepisanog sadržaja u izlaznom (.exe) fajlu ukazuje drugi argument ove procedure; pre poziva ove procedure, linker je već prepisao sadržaj binarnog prevedenog koda (bez zaglavlja) svih ulaznih .obj fajlova u sadržaj izlaznog fajla.

Na samom početku ulaznog fajla nalazi se zaglavlje. Svi pomeraji (*offset*, odnosno relativne adrese) u njemu izraženi su u jedinicama `sizeof(char)==1`, a veličine su `unsigned long` (skraćeno `ulong`). Sadržaj početka zaglavlja je, redom, sledeći:

- jedan `ulong` koji sadrži pomeraj početka binarnog prevedenog koda u .obj fajlu u odnosu na početak celog sadržaja tog fajla (zapravo sadrži veličinu celog zaglavlja iza koga sledi binarni prevedeni kod);
- jedan `ulong` koji sadrži ukupan broj simbola koji se uvoze (*n*);
- *n* redom poređanih parova: ime simbola koji se uvozi (niz znakova proizvoljne dužine, završen znakom `'\0'`), iza koga sledi jedan `ulong` koji predstavlja pomeraj prvog nerazrešenog adresnog polja u mašinskoj instrukciji koje treba da sadrži vrednost razrešene adrese tog simbola; takva polja su dalje ulančana u jednostruku listu, tako da svako polje sadrži pomeraj narednog takvog polja za isti simbol, s tim da vrednost pomeraja 0 označava kraj liste (poslednje takvo polje za taj simbol); ovi pomeraji su relativni u odnosu na početak prevedenog binarnog koda unutar sadržaja .obj fajla.

Linker poseduje tabelu simbola čija operacija:

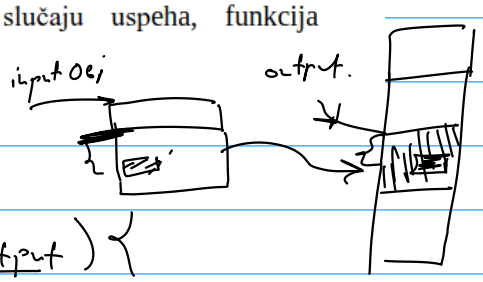
```
ulong SymbolTable::resolveSymbol(char* symbol);
```

vraća pomeraj (relativnu adresu) u odnosu na početak izlaznog fajla (.exe) u koji se dati simbol prevodi, ukoliko on postoji u tabeli, a 0 ako ga nema. Grešku nedefinisanog simbola treba obraditi pozivom funkcije:

```
int errorSymbolUndefined(char* symbol);
```

Ova funkcija ispisuje korisniku poruku o nedefinisanom datom simbolu i vraća -1, što u tom slučaju treba da vrati i funkcija `resolveSymbols`. U slučaju uspeha, funkcija `resolveSymbols` treba da vrati 0.

```
int resolveSymbols (char* inputObj, char* output);
```




```
int resolveSymbols (char* inputObj, char* output) {
    ulong ent_off = *((ulong*) inputObj);

    ulong n_sym = *((ulong*) (inputObj + sizeof(ulong)));
    char* sym = inputObj + 2*sizeof(ulong);
    for (int i=0; i<n_sym; i++) {
        ulong sym_off = *((ulong*)(sym + strlen(sym) + 1));
        ulong sym_def = SymbolTable::resolveSymbol(sym);
        if (sym_def == 0) return errorSymbolUndefined(sym);
        while (sym_off != 0) {
            ulong* current_field = (ulong*)(obj + sym_off);
            sym_off = *current_field;
            *current_field = sym_def;
        }
```

```
        sym += strlen(sym) + 1 + sizeof(long);
    }

    return 0;
}
```