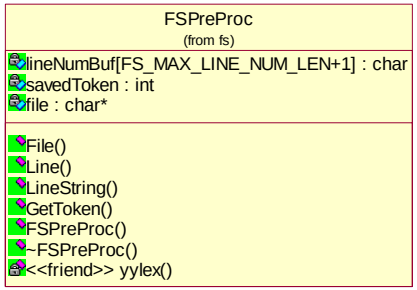
1. Interfejs pretprocesora prema ostatku sistema

Interfejs pretprocesora prema ostatku sistema nalazi se u modulu FCPPROC i čine ga klasa **FSPreProc**, funkcija **yylex()** i globalne promenljive **yytext** i **yyleng**.



Klasa FSPreProc odgovorna je za inicijalizaciju pretprocesora na početku analize modula, clean up na kraju analize modula i preko nje se zadaju konfiguracioni parametri za pretprocesor (npr. da li se prepoznaju CPP komentari i sl.).

1.1. Interfejs pretprocesora prema parseru

Funkcija yylex (koja vraća ceo broj koji predstavlja klasni deo tekućeg tokena) i promenljive yytext (sadrži leksemu tj. niz znakova ulaznog teksta koji odgovaraju tekućem tokenu) i yyleng (dužina niza znakova u yytext) sačinjavaju standardan interfejs pretprocesora prema parseru. Parser dakle vidi pretprocesor kao leksički analizator.

Klasa FSPreProc sadrži servise za dobijanje informacija koje su potrebne u sintaksnoj analizi van standardnog interfejsa, kao što su ime tekućeg fajla i broj linije u fajlu u kojoj se nalazi tekući token.

1.2. Princip korišćenja pretprocesora

Pretprocesor se koristi na sledeći način:

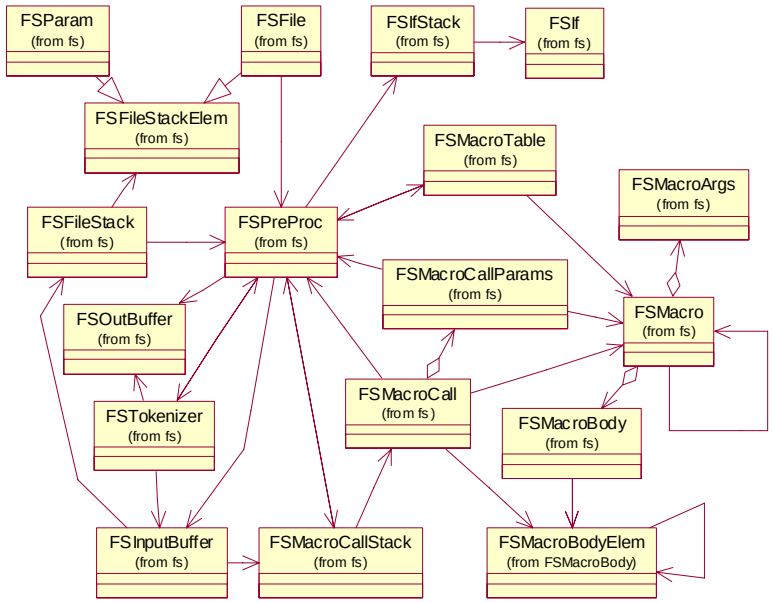
```
int main( int, char *argv[] )
{
   int ret_code;
   FSPreProc *pPreProc = new FSPreProc( argv[1] );
   pPreProc->cppEnabled = TRUE;

      while ( ( ret_code = yylex() ) != EOF )
   {
      printf( "code:%d, yytext=%s, yyleng=%d, file:%s, line:%u\n",
           ret_code, yytext, yyleng, pPreProc->File(), pPreProc->Line() );
      //ili samo printf("%s ", yytext);
   }
      return 0;
}
```


1.3. Semantičke akcije u okviru leksičke analize

Leksička analiza koji obavlja pretprocesor se može proširiti semantičkim akcijama (tipično se to svodi na dodelu vrednosti promenljivoj **yylval**) tako što se na kraju funkcije yylex(), dodaje switch iskaz na osnovu koda tokena i u okviru njega dodaju semantičke akcije.

2. Interna organizacija pretprocesora



Pretprocesor se sastoji iz sledećih modula:

- FSPPROC sadrži interfejsnu klasu klasu FSPreProc, funkciju yylex() i promenljive yytext i yyleng
- FSTOKEN sadrži baferske klase FSInputBuffer i FSOutBuffer, kao i pretprocesorski tokenizator (klasa FSTokenizer).
- FSINBUF sadrži klase koje realizuju stekove (FSFileStack, FSMacroCallStack i FSIfStack i pomoćne klase) i tabelu makro definicija (FSMacroTable i pomoćne klase).
- FSIFEXPR sadrži parser za izraze u okviru #if i #if(n)def direktiva.
- FSERROR sadrži funkcije za prijavu upozorenja, običnih i fatalnih grešaka (tipa prepunjavanja nekog bafera i sl.)

2.1. Funkcija yylex()

Rad pretprocesora, posle inicijalizacije struktura u konstruktoru instance klase FSPreProc, odvija se višestrukim pozivanjem, od strane parsera, funkcije yylex().

U okviru jednog poziva yylex(), odvijaju se sledeće aktivnosti:

- ažurira se ime tekućeg fajla i broj tekuće linije.
- u "glavnoj" petlji se odvija poziv pretprocesorskog tokenizatora (klasa FSTokenizer). Funkcija FSTokenizer::Get vraća kod takozvanog pretprocesorskog tokena. (Pretprocesorski tokeni se razlikuju od takozvanih izlaznih tokena koje vraća yylex()).
- na osnovu koda pretprocesorskog tokena, bira se jedna od sledećih akcija
    1. funkcija yylex() završava rad i pretprocesorski token se vraća kao izlazni token (ovo je najčešći slučaj i dešava se kada token ne zahteva posebnu pretprocesorsku akciju).
    2. token zahteva neku pretprocesorsku akciju u koje spadaju: parsiranje i obrada pretprocesorske direktive i zamena makroa. Ovaj token u najvećem broju slučajeva ne šalje se parseru već inicira neku akciju po čijem završetku se ide u novu iteraciju "glavne" petlje.

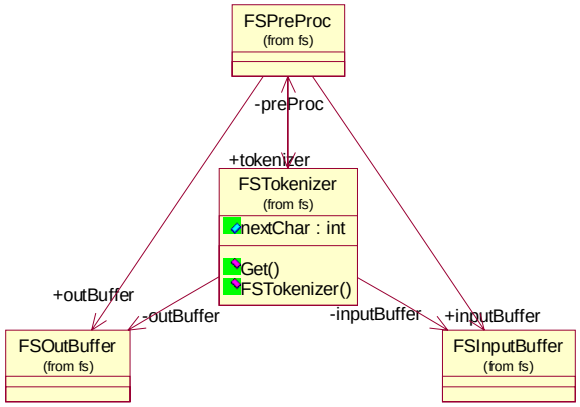
U nekim situacijama događa se da se radi odluke o akciji mora pročitati jedan pretprocesorski token više: na primer, ako je makro definisan sa:

#define MACRO() *zamena*

a u ulaznom tekstu se naiđe na identifikator MACRO, mora se pročitati sledeći token da bi se odlučilo da li se radi o pozivu makroa (ako je sledeći token otvorena zagrada) ili se

radi o običnom identifikatoru koji se odmah prosleđuje parseru. U ovom drugom slučaju, pročitani token viška pamti se u promenljivoj savedToken klase FSPreProc da bi se upotrebio u sledećem pozivu funkcije yylex().

2.2. Pretprocesorski tokenizator



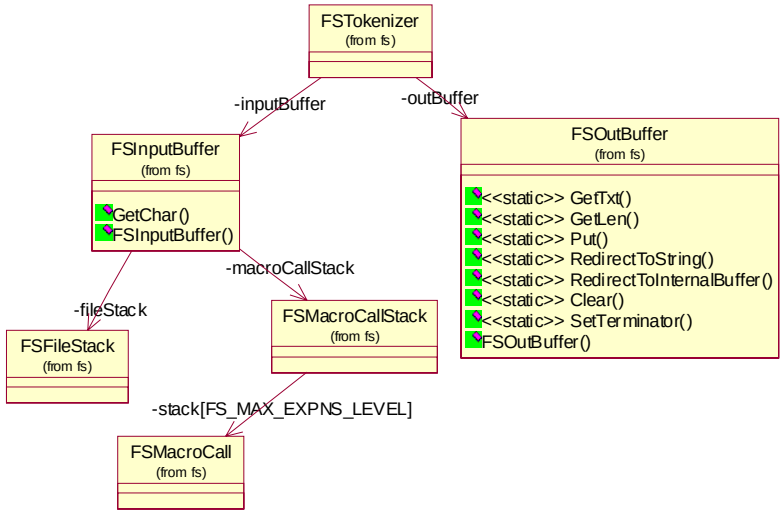
Izdvajanje pretprocesorskih tokena iz ulaznog teksta zadatak je klase FSTokenizer. Najvažniji servis ove klase je funkcija Get() kojom se čita tekst iz ulaznog bafera (instanca klase FSInputBuffer), prebacuje se u izlazni bafer (instanca klase FSOutBuffer) i kada se oformi kompletan token, vraća se njegov kod. Funkcija Get realizovana je konačnim automatom. Pretprocesorski tokeni obuhvataju sve normalne parserske tokene i još nekoliko

dodatnih: SHARP, SHARPSHARP i NL. Dodatni tokeni se nikada ne prosleđuju parseru, već služe za aktiviranje pojedinih pretprocesorskih aktivnosti.

U okviru tokenizatora vrši se baferisanje jednog znaka iz ulaznog bafera (promenljiva nextChar klase FSTokenizer). O tome se vodi računa pri makroekspanziji i obradi #include direktive, da se taj znak zapamti, a posle obrade makroa ili include datoteke vrati stara vrednost.

Promenljiva processBackslashInString klase FSTokenizer koristi se za zadavanje režima rada tokenizatora: vrednost TRUE označava da se \ tretira kao početak iskejp sekvence, u suprotnom se tretira kao svaki drugi znak. Ovaj drugi režim je potreban kod procesiranja imena fajla u #include direktivi.

2.3. Ulazni i izlazni baferi



Klasa FSInputBuffer realizuje ulazni bafer pretprocesora, iz koga pretprocesorski tokenizator (instanca klase FSTokenizer) čita tekst. Tekst dolazi ili iz datoteke, ili iz tela makroa ukoliko se pretprocesor nalazi u režimu makroekspanzije. Zbog mogućnosti ugneždavanja #include datoteka, informacije o otvorenim fajlovima nalaze se na steku (klasa FSFileStack). Zbog mogućnosti ugneždavanja makropoziva, aktivni makropozivi nalaze se takođe na steku. Ukoliko stek makropoziva nije prazan, pretprocesor je u režimu makroekspanzije i tekst se čita iz tela makroa, u suprotnom se čita iz datoteke.

Pretprocesorski tokenizator ne koristi direktno promenljive yytext i yyleng da bi upisao izdvojenu leksemu, već postoji klasa FSOutBuffer koja predstavlja apstrakciju izlaznog bafera. Servisi ove klase ažuriraju i promenljive yytext(koja najčešće pokazuje na početak niza outBuffer, člana klase FSOutBuffer) i yyleng. Iz izlaznog bafera čita se i u okviru samog pretprocesora na više mesta, kada god nije dovoljan samo kod tokena da bi se odlučilo koju akciju preduzeti (na primer, kada treba prepoznati da li je identifikator makro poziv ili ne).

U situaciji kada se između dva poziva yylex() mora zapamtiti unapred učitani token (videti diskusiju u odeljku 2.1.), promenljiva yytext preusmerava se da ukazuje na prethodni token, dok u nizu outBuffer ostaje tekst unapred učitanog tokena, da bi se vrednost yytext vratila na outBuffer u sledećem pozivu yylex().

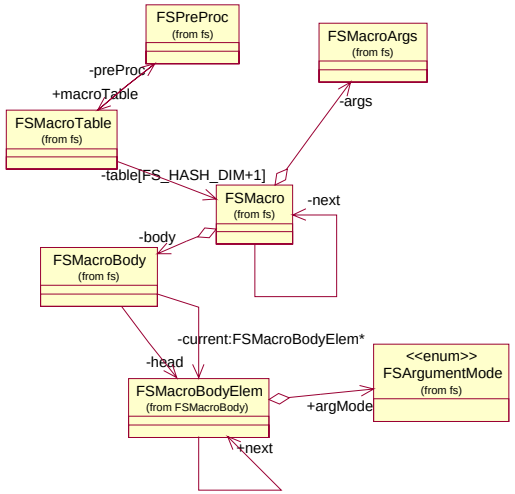
2.4. Obrada makrodefinicija (direktive #define i #undef)

2.4.1. Tabela makrodefinicija

Makrodefinicije (instance klase FSMacro) se čuvaju u tabeli makrodefinicija, instanci klase FSMacroTable. Makrodefinicije je moguće dodati u tabelu (servis Insert koji ima dve varijante u zavisnosti od toga da li je reč o preddefinisanoj ili "običnoj" makrodefiniciji), ukloniti iz tabele (ako je to dozvoljeno) - servis Remove i tražiti u tabeli na osnovu imena - servis LookUp. Pri dodavanju makrodefinicije proverava se da li već postoji makrodefinicija sa istim imenom i ako postoji, da li su identične.

Odabrano je rešenje po kome se za uklonjene makrodefinicije memorija ne oslobađa odmah, nego se one sakupljaju u posebnoj listi, tako da se brišu u destruktoru FSMacroTable. Ideja je da se smanji fragmentacija memorijskih segmenata za dinamičku alokaciju memorije.

2.4.2. Makrodefinicije



Svaka makrodefinicija (instanca klase FSMacro) ima ime, listu formalnih argumenata (instanca klase FSMacroArgs) i telo (instancu klase FSMacroBody). U okviru konstruktora klase FSMacro čita se sa ulaza (iz inputBuffer-a) kompletna makrodefinicija. Pri tome je konstruktor klase FSMacroArgs dužan da parsira deo makrodefinicije koji se odnosi na argumente, a FSMacroBody deo koji se odnosi na telo makrodefinicije.

Klasa FSMacroArgs čuva listu imena svih formalnih argumenata, njihov broj i da li makro ima zagrade ili ne (nije svejedno za makro sa 0 argumenata da li je definisan sa zagradama ili nije). Imena argumenata potrebna su da bi se u telu identifikovala mesta upotrebe argumenata.

Klasa FSMacroBody procesira telo makroa i u njemu identifikuje formalne argumente i makrooperatore # i ## (pogledati diskusiju u odeljku 2.5.1). Telo se pamti kao lista instanci klase FSMacroBodyElem. Svaki elemenat tela ime dve glavne komponente:

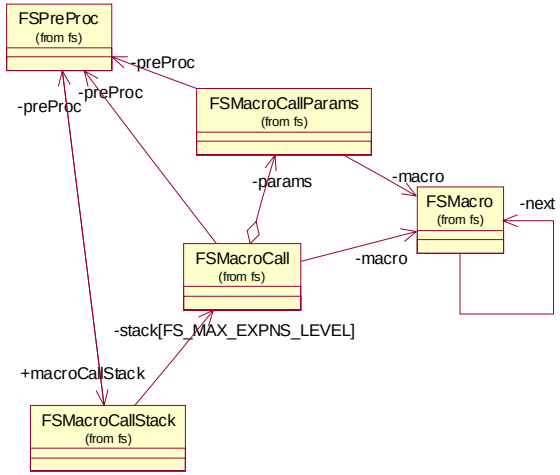
- string txt koji pamti deo tela koji sadrži običan tekst bez parametara ili operatora
- celobrojni indeks formalnog argumenta (u listi argumenata) koji treba pri ekspanziji ubaciti iza txt-a.

Prema tome, telo se deli na elemente od kojih svaki obuhvata deo običnog teksta zaključno sa pojavom formalnog argumenta. Naravno, ako telo počinje sa argumentom ili se dva argumenta nalaze neposredno jedan do drugog, string txt će biti prazan. Takođe, ako se telo završava običnim tekstom indeks argumenta u poslednjem elementu tela će biti 0, što označava da ne treba ubacivati stvarni parametar iza txt-a poslednjeg elementa. Indeks 0 je moguć i u situaciji kada je običan tekst u telu dovoljno dugačak da ne može ceo stati u string txt (koji je fiksirane dužine) tako da se za taj deo tela mora alocirati više od jednog elementa.

Uz svaki formalni argument u FSBodyElem pamti se u promenljivoj argMode i režim zamene argumenta stvarnim parametrom pri ekspanziji makroa koji može biti:

- FS_AR_NORMAL što znači da stvarni argument treba da se ubaci neekspandovan (znači onakav kakav je pokupljen iz liste stvarnih argumenata). Ovo se primenljuje u slučaju kada je formalni argument jedan od operanada operatora konkatenacije ##.
- FS_AR_STRING što znači da stvarni parametar treba uokviriti navodnicima pri čemu treba dodati znak \ ispred svake pojave znakova \ ili " u parametru. Ovo je posledica primene operatora stringizacije # na formalni argument.
- FS_AR_EXPAND što znači da u tekst treba ubaciti rezultat makroekspanzije stvarnog parametra (normalna situacija kada formalni argument nije pod dejstvom operatora # ili ##).
- FS_AR_FILENAME se ne koristi za parametre, već označava da u tekst pri ekspanziji treba ubaciti ime fajla (podrška za preddefinisani makro __FILE__).
- FS_AR_LINENUM se ne koristi za parametre, već označava da u tekst pri ekspanziji treba ubaciti tekući broj linije (podrška za preddefinisani makro __LINE__).

2.5. Makroekspanzija



Kada se u ulaznom tekstu identifikuje makropoziv (u funkciji yylex()), kreira se instanca klase FSMacroCall i potisne na stek makropoziva (klasa FSMacroCallStack). Od tog trenutka ulaz pretprocesora ide iz tela tekućeg makroa. Kada se celo telo obradi, sa steka se skida tekući poziv.

Instanca klase FSMacroCall sadrži instancu klase FSMacroCallParams koja je odgovorna za kupljenje i predskaniranje stvarnih parametara u svom konstruktoru. Tek kada su svi parametri sakupljeni i predskanirani, makropoziv ide na stek. Pre poziva se čuva vrednost lookahead znaka tokenizatora, da bi se ista posle završenog makropoziva vratila.

2.5.1.Predskaniranje parametara makropoziva u C pretprocesoru

Kod C pretprocesora zahteva se sledeća semantika izračunavanja argumenata makropoziva:

- Argumenti se uvek skupljaju bez makroekspanzije. Pri tome, ako se makropoziv Y nalazi u telu nekog drugog makroa X, pri skupljanju argumenata poziva Y vrši se zamena argumenata X-a svojim vrednostima.

  Primer:

  ```
  #define X 1
  #define A( x ) #x
  #define B( y ) A( y ) A( X )
  ```

  Poziv B(X) daje rezultat "1" "X".

- Sakupljeni argumenti se izolovano makroekspanduju.

  Primer:

  ```
  #define X 1,2
  ```



  ```
  #define A(x,y) x,y
  ```

  Poziv A(X) daje poruku o greški jer se ne vrši ekspanzija X pri brojanju argumenata poziva A.

- U telu makropoziva vrši se zamena formalnih parametara stvarnim argumentima tako da se svuda zamenjuju njihove makroekspandovane vrednosti, OSIM ako su pod dejstvom operatora # ili ##, kada se zamenjuju njihove pokupljene vrednosti.

  Primer:

  ```
  #define M2 2
  #define UNUTRASNJI(x , y ) x ## y
  ```

  Poziv UNUTRASNJI(TEKST, M2) daje rezultat TEKSTM2 jer nije izvršena makroekspanzija argumenta y pod dejstvom operatora ##.

  Ako želimo da izlaz bude TEKST2, moramo dodati novu makrodefiniciju

  ```
  #define SPOLJNI( x, y ) UNUTRASNJI( x, y )
  ```

  pri čemu se koristi pravilo 1 o zameni argumenata ugnezđujuće makrodefinicije. Poziv SPOLJNI(TEKST, M2) daje TEKST2.

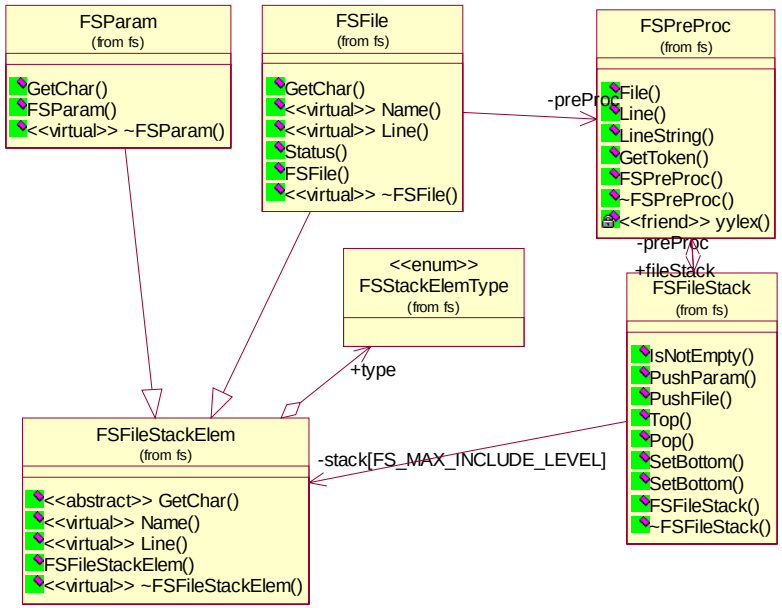
2.5.2. Realizacija predskaniranja

Posle svakog skupljanja (koje se odvija bez detektovanja makropoziva) argumenti se izolovano makroproširuju pa da se uz svaki makropoziv pamti i sakupljena i proširena vrednost. Pri svakoj pojavi se zamenjuje jedna od ovih vrednosti prema pravilima datim u odeljku 2.4.2. Ekspandovanje argumenata je rekurzivni postupak, jer pri ekspanziji opet dolazi do sakupljanja i ekspanzije argumenata ugneždenog makropoziva.

Vrednosti dna steka datoteka (instanca klase FSFileStack) i stek makropoziva (instanca klase FSMacroCallStack) zapamte se, a zatim se dna postave da ukazuju na vrhove stekova. Time se efektivno postiže pražnjenje stekova uz čuvanje prethodnog sadržaja.

Parametar (instanca klase FSParam) ekspanduje se tako što se postavi na stek datoteka. Vrednost parametra sada predstavlja ulaz za pretprocesor, normalno se propušta kroz tokenizator i po potrebi vrši ekspanzija makropoziva unutar parametra. Kada se iščita ceo parametar, on se skida sa steka datoteka koji u tom trenutku postaje prazan. Izlaz pretprocesora za vreme čitanja parametra pamti se kao ekspandovana vrednost parametra. Postupak se ponavlja za sve parametre redom u konstruktoru FSMacroCallParams. Kada su svi parametri obrađeni, restauriraju se stare konfiguracije stekova datoteka i makropoziva.

2.6. Uključivanje datoteka zaglavlja (direktiva #include)

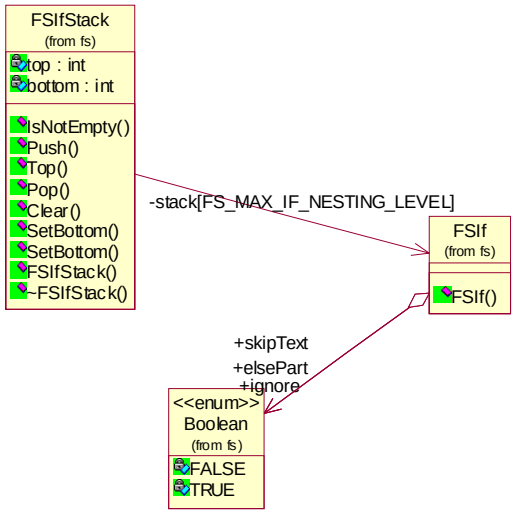


Klasa FSFile izvedena iz apstraktne klase FSFileStackElem odgovorna je za ulaz iz datoteke (funkcija GetChar()). Pri čitanju iz datoteke obezbeđuje se eliminacija kombinacije znakova \ i \n. Obezbeđuje se takođe postojajanje znaka \n pre kraja datoteke i vodi se evidencija o tekućoj liniji.

Klasa FSParam takođe je izvedena iz apstraktne klase FSFileStackElem i ima ulogu u predskaniranju parametara makropoziva.

S obzirom na mogućnost ugnežđavanja #include direktiva, instanca klase FSFile potiskuje se na stek datoteka (klasa FSFileStack) kada se naiđe na #include, a skida sa steka kada se dođe do kraja datoteke.

2.7. Obrada direktiva uslovnog prevođenja (#if, #ifdef, #ifndef, #elif, #endif)



Svaki put kada se naiđe na #if ili #if(n)def direktivu, parser pretprocesorskih izraza (modul FSIFEXPR) procesira direktivu i na stek if-ova (klasa FSIfStack) potiskuje se instanca klase FSIf. Kada se naiđe na #endif, sa steka if-ova se skida vršna instanca klase FSIf. Klasa FSIf je jednostavna, sadrži samo indikatore da li je preduslov zadovoljen i da li je naišao #elif.

Pri svakom #include mora se zapamtiti konfiguracija steka if-ova a stek isprazniti nameštanjem dna da ukazuje na vrh. Kada se završi čitanje iz #include, treba restaurirati stek if-ova. *Napomena: ovo tek treba realizovati u FSIfStack klasi*. Ovo je potrebno da bi unutar svakog fajla pojedinačno bili uravnoteženi #if sa #endif.

Na primer, ako je sadržaj fajla a.h:

```
#if 1
#include "b.h"
#endif
```

a sadržaj fajla "b.h"

```
#endif
```

pretprocesor treba da prijavi grešku da se u fajlu b.h pojavila direktiva #endif viška, a da upari #if sa #endif-om iz fajla a.h.