```
struct Uzel { int Typ;
            union { struct { Uzel *prvni,*druhy,*treti,*ctvrty; } z;
                    int Cislo;
                    const char *Retez;
                    int *Adresa; } z; };
Uzel *Koren=0;    // Kořen stromu interní formy
```

```
void Abort() { system("pause"); abort(); }

int Interpr(const Uzel *u)    // Interpretace interní formy
{
  if (u==NULL) return 0;
  #define  prvni    u->z.z.prvni
  #define  druhy    u->z.z.druhy
  #define  treti    u->z.z.treti
  #define  ctvrty   u->z.z.ctvrty
  switch (u->Typ) {
    case 0: Interpr(prvni); Interpr(druhy);
            return 0;
    case '=': return *prvni->z.Adresa = Interpr(druhy);
    case P_DELEN: { int d=Interpr(druhy);
                    if (d==0) { printf("\nDeleni nulou\n"); Abort(); }
                      return *prvni->z.Adresa/=d; }
    case PRINT: if (prvni->Typ!=RETEZ)
                { printf("%i",Interpr(prvni)); return 0; }
                if (druhy)
                { printf(prvni->z.Retez,Interpr(druhy)); return 0; }
                printf(prvni->z.Retez);
                return 0;
    case FOR: Interpr(prvni);
              while (Interpr(druhy)) { Interpr(ctvrty);
                                       Interpr(treti); }
              return 0;
    case IF: if (Interpr(prvni)) Interpr(druhy);
             else Interpr(treti);
             return 0;
    case INKREM: if (prvni) return ++*prvni->z.Adresa;
                 return (*druhy->z.Adresa)++;
    case '!': return !Interpr(prvni);
    case '<': return Interpr(prvni) < Interpr(druhy);
    case OR:  return Interpr(prvni) || Interpr(druhy);
    case '-': if (druhy) return Interpr(prvni) - Interpr(druhy);
```

```
                return -Interpr(prvni);
    case '/': { int d=Interpr(druhy);
                if (d==0) { printf("\nDeleni nulou\n"); Abort()(); }
                return Interpr(prvni)/d; }
    case CISLO: return u->z.Cislo;
    case PROMENNA: return *u->z.Adresa;
    default: printf(u->Typ<256?"\nNeznamy symbol: '%c'"
                        :"\nNeznamy symbol: %i",u->Typ); Abort()();
  }
}
```

```
  extern FILE *yyin;
  int yyparse();
  // cesta k souboru se zdrojovým programem mC
  // je zadána na příkazovém řádku při volání interpretu
  if ((yyin=fopen("zdrojový program","rt"))==NULL)
                      { /* neotevřel se zdrojový soubor */ }
  Flush();
  LexInic();
  int parse=yyparse();
  fclose(yyin);
  if (!(Chyby || parse)) Interpr(Koren);
```

```
Uzel *GenUzel(int typ,Uzel *prvni=0,Uzel *druhy=0,Uzel *treti=0,Uzel *ctvrty=0);
```

Alokuje paměť pro `Uzel`, uloží do něho hodnoty z parametrů:

- `Typ: typ`
- `prvni: prvni`
- `druhy: druhy`
- `treti: treti`
- `ctvrty: ctvrty`

Ukazatel na `Uzel` vrátí jako funkční hodnotu.

```
Uzel *GenCislo(int cislo);
```

Alokuje paměť pro `Uzel`, uloží do něho hodnoty:

- `Typ: CISLO`
- `Cislo: cislo`

Ukazatel na `Uzel` vrátí jako funkční hodnotu.

```
Uzel *GenRetez(const char *retez);
```

Alokuje paměť pro `Uzel`, uloží do něho hodnoty:

```
Typ: RETEZ
Retez: retez
```

Ukazatel na `Uzel` vrátí jako funkční hodnotu.

Ve funkci `GenPromen` je použita datová struktura typu mapa:

klíč: jméno proměnné

data: adresa proměnné v paměti

```
Uzel *GenPromen(const char *jmeno);
```

Vyhledá v mapě záznam se jménem proměnné uvedeném v parametru funkce:

- Záznam nebyl nalezen (první výskyt proměnné s daným jménem v programu):

  Alokuje paměť pro uložení hodnoty proměnné.

  Do mapy uloží nový záznam `jmeno` + `adresa`.

  Alokuje paměť pro `Uzel`, uloží do něho hodnoty:

  ```
  Typ: PROMENNA
  Adresa: adresa
  ```

  Ukazatel na `Uzel` vrátí jako funkční hodnotu.

- Záznam byl nalezen:

  Alokuje paměť pro `Uzel`, uloží do něho hodnoty:

  ```
  Typ: PROMENNA
  Adresa: adresa ze záznamu
  ```

  Ukazatel na `Uzel` vrátí jako funkční hodnotu.

```
extern unsigned Radek,Sloupec;
bool Chyby=false;
void Chyba(const char *S, Pozice P)
```

nastaví `Chyby=true`

dle parametru `P` vypíše zprávu o chybě:

```
SLOUPEC: řádek.sloupec text chyby S
  RADEK: řádek text chyby S
BEZPOZICE: text chyby S
```

```
void yyerror(const char *S)
```

zavolá funkci `Chyba(S)`
