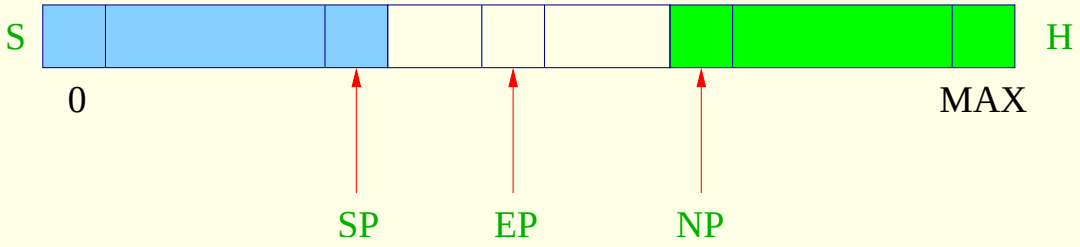
Viidad ja dünaamiline mäluhaldus

Kuhi:



- H on mälupiirkond dünaamilise elueaga andmete hoidmiseks (Heap).
- NP on register, mis sisaldab kuhja kõige alumise täidetud pesa aadressi (New Pointer).
- EP on register, mis sisaldab kõige ülemise pesa aadressi, kuhu SP võib antud funktsiooni täitmise ajal viidata (Extreme Pointer).

Viidad ja dünaamiline mäluhaldus

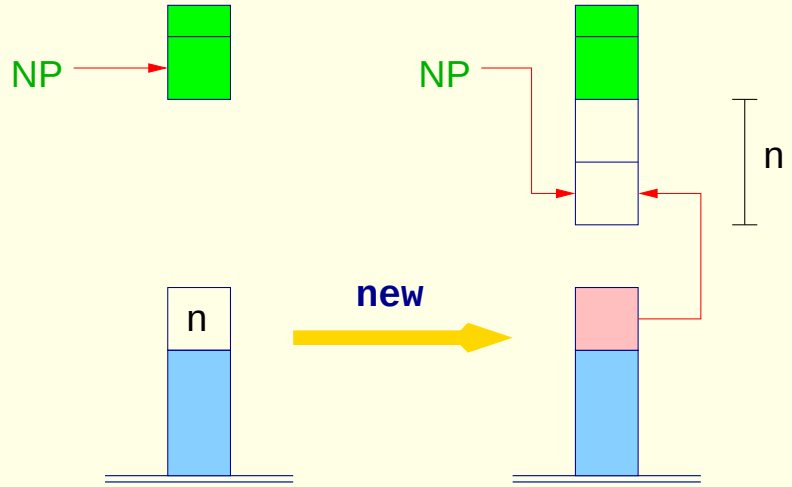
- Magasin ja kuhi kasvavad üksteise suunas, aga ei tohi kattuda (Stack Overflow).
- "Kokkupõrge"võib olla pähjustatud nii SP suurendamisest, kui NP vähendamisest.
- EP aitab meil vältida ületäitumist magasini operatsioonide korral.
- EP väärtus on võimalik määrata staatiliselt.
- Kuhjast mälu võttes tuleb ületäitumist kontrollida.

Viidad ja dünaamiline mäluhaldus

- *Viidad* võimaldavad ligipääsu anonüümsetele, dünaamiliselt loodud objektidele, mille eluiga ei allu LIFO printsiibile.
- Viitväärtuse saamiseks on kaks võimalust:
  - funktsiooni **malloc**(e) väljakutse reserveerib kuhjas mälupiirkonna mille suurus on e väärtus ning väljastab selle piirkonna algusaadressi;
  - aadressoperaatori & rakendamine muutujale väljastab selle muutuja aadressi (so. L-väärtuse).

$$\begin{aligned} \text{code}_R\ \textbf{malloc}(e)\ \rho &= \text{code}_R\ e\ \rho \\ &\quad\ \texttt{new} \\ \text{code}_R\ (\&e)\ \rho &= \text{code}_L\ e\ \rho \end{aligned}$$

Viidad ja dünaamiline mäluhaldus




```
if (NP - S[SP] ≤ EP)
    S[SP] = NULL;
else {
    NP = NP − S[SP];
    S[SP] = NP;
}
```

- `NULL` on spetsiaalne viitkonstant; samastatakse täisarvulise konstandiga 0.
- Ületäitumise korral väljastatakse `NULL`-viit.

Viidad ja dünaamiline mäluhaldus

- Viidatava väärtuse saamiseks on kaks võimalust:
  - operaatori $*$ rakendamine avaldisele e väljastab mälupesa sisu, mille aadress on avaldise e R-väärtus;
  - kirje komponendi selekteerimine läbi viida $e \to c$ on ekvivalentne avaldisega $(*e).c$.

$$
\begin{aligned}
\text{code}_L\ (*e)\ \rho &= \text{code}_R\ e\ \rho \\
\text{code}_L\ (e \to c)\ \rho &= \text{code}_R\ e\ \rho \\
&\quad \texttt{load}\ (\rho\ a) \\
&\quad \texttt{add}
\end{aligned}
$$

Viidad ja dünaamiline mäluhaldus

- Näide: olgu antud deklaratsioonid

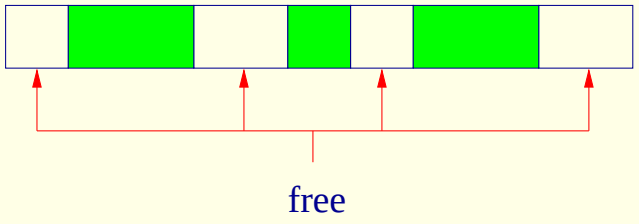
$$\textbf{struct}\ t\ \{\ \textbf{int}\ a[7];\ \textbf{struct}\ t\ *b;\ \};$$
$$\textbf{int}\ i,\ j;$$
$$\textbf{struct}\ t\ *pt;$$

- Siis $\rho = \{\ a \mapsto 0,\ b \mapsto 7,\ i \mapsto 1,\ j \mapsto 2,\ pt \mapsto 3\ \}$.
- Avaldise $((pt \to b) \to a)[i+1]$ korral emiteeritakse kood:

```
loada 3    load       loada 1    loadc 1
loadc 7    loadc 0    loadc 1    mul
add        add        add        add
```

Viidad ja dünaamiline mäluhaldus

- Mälu vabastamine toimub C-s funktsiooni **free**(e) väljakutse abil.
- Antud mälupiirkond märgistatakse kui vaba ja pannakse spetsiaalsesse vabamälu listi (free list), kust **malloc** saab vajadusel selle uuesti kasutusele võtta.
- Probleemid:
  - peale mälupiirkonna vabastamist võivad mõned, ikka veel kättesaadavad, viidad sellele viidata (dangling references);
  - mälu võib ajapikku muutuda *fragmenteerituks*;



  - vabamälu listi haldamine võib olla küllalt kulukas.

Viidad ja dünaamiline mäluhaldus

- Alternatiiv: funktsiooni **free** väljakutse korral teha mittemidagi.

$$\text{code}\ (\textbf{free}(e);)\ \rho \quad = \quad \begin{array}{l}\text{code}_R\ e\ \rho \\ \texttt{pop}\end{array}$$

- Mälu täitumisel vabastada kindlasti mitte-kättesaadav mälu automaatselt kasutades *prügikoristust* (garbage collection).
  - \+ Mälu võtmine ja vabastamine on lihtne ja väga efektiivne.
  - \+ Pole ”ripnevate” viitade probleemi.
  - \+ Mitmed prügikoristuse algoritmid defragmenteerivad kasutusel oleva mälu.
  - – Prügikoristus võib olla aeganõudev, mistõttu võivad tekkida programmi täitmises märgatavad pausid.

Avaldiste transleerimine — kokkuvõte

$$
\begin{array}{lcll}
\text{code}_L\ (e_1[e_2])\ \rho & = & \text{code}_R\ e_1\ \rho & \\
 & & \text{code}_R\ e_2\ \rho & \\
 & & \texttt{loadc}\ |t| & \\
 & & \texttt{mul} & \\
 & & \texttt{add} & e_1 \text{ on tüüpi } t \\
\text{code}_L\ (e.c)\ \rho & = & \text{code}_L\ e\ \rho & \\
 & & \texttt{loadc}\ (\rho\ c) & \\
 & & \texttt{add} & \\
\text{code}_L\ (*e)\ \rho & = & \text{code}_R\ e\ \rho &
\end{array}
$$

Avaldiste transleerimine — kokkuvõte

$$
\begin{array}{lcll}
\text{code}_L\ (e \to c)\ \rho & = & \text{code}_R\ e\ \rho & \\
 & & \texttt{load}\ (\rho\ a) & \\
 & & \texttt{add} & \\
\text{code}_L\ x\ \rho & = & \texttt{loadc}\ \ (\rho\ \ x) & \\
\text{code}_R\ (\&e)\ \rho & = & \text{code}_L\ e\ \rho & \\
\text{code}_R\ e\ \rho & = & \text{code}_L\ e\ \rho & \text{kui } e \text{ on massiiv} \\
\text{code}_R\ (\Box e)\ \rho & = & \text{code}_R\ e\ \rho & \\
 & & \texttt{op1} & \text{kus } \texttt{op1} \text{ on operaatori } \Box \text{ kood} \\
\text{code}_R\ (e_1 \oplus e_2)\ \rho & = & \text{code}_R\ e_1\ \rho & \\
 & & \text{code}_R\ e_2\ \rho & \\
 & & \texttt{op2} & \text{kus } \texttt{op2} \text{ on operaatori } \oplus \text{ kood}
\end{array}
$$

Avaldiste transleerimine — kokkuvõte

$\text{code}_R\ q\ \rho$	=	`loadc q`	q on konstant
$\text{code}_R\ (e_1 = e2)\ \rho$	=	$\text{code}_R\ e_2\ \rho$	
		$\text{code}_L\ e_1\ \rho$	
		`store`	
$\text{code}_R\ e\ \rho$	=	$\text{code}_L\ e\ \rho$	
		`load`	ülejäänud juhtudel

Funktsioonid

- Funktsiooni definitsioon koosneb:
  - funktsiooni *nimi*, mille kaudu toimub tema väljakutsumine;
  - funktsiooni *formaalsete parameetrite* spetsifikatsioon;
  - funktsiooni *resultaadi tüüp*;
  - funktsiooni *keha*.
- C-s kehtib:

$$\text{code}_R\ f\ \rho \quad = \quad \_f \quad = \quad f\text{-i koodi algaadress}$$

- Seega peab aadresskeskond haldama ka funktsioonide nimesid!

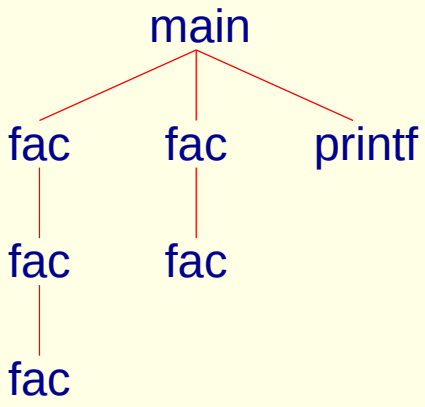
Funktsioonid

- Näide:

```
int fac (int x) {
    if (x ≤ 0) return 1;
    else return x * fac(x − 1);
}
```

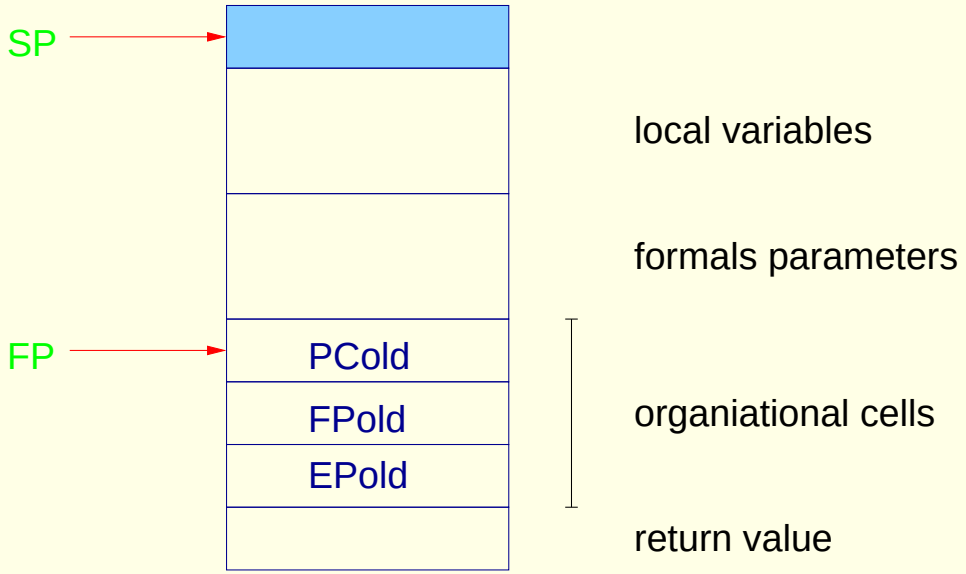
```
main () {
    int n;
    n = fac(2) + fac(1);
    printf("%d", n);
}
```

- Samast funktsioonist võib olla aktiivsed mitu eksemplari.



Funktsioonid

- Funktsiooni iga *eksemplari* formaalsed parameetrid ja lokaased muutujad tuleb hoida eraldi.
- Reserveerime magasinis spetsiaalse mälupiirkonna (Stack Frame).



- FP on register, mis viitab viimasele *organisatoorsele pesale* ja mida kasutatakse formaalsete parameetrite ja lokaalsete muutujate adresseerimiseks (Frame Pointer).

Funktsioonid

- Väljakutsuja peab peale väljakutsutud funktsiooni lõpetamist olema võimeline jätkama täitmist omas freimis.
- Seetõttu tuleb funktsiooni väljakutsel salvestada:
  - väljakutsuja freimi aadress FP;
  - koodi aadress, kust tuleb jätkata peale väljakutse lõpetamist (so. käsuregister PC);
  - väljakutsuja magasini maksimaalne võimalik aadress EP.
- Lihtsustus: eeldame, et funktsioonide resultaatväärtused mahuvad ühte mälupessa.

Funktsioonid

- Peame eristama kahte eri liiki muutujaid:
  - *globaalsed* muutujad, mis on defineeritud funktsioonidest väljaspool;
  - *lokaalsed* (ehk automaatsed) muutujad (k.a. formaalsed parameetrid), mis on defineeritud funktsioonide siseselt.
- Aadresskeskond ρ seob muutujanimedega paarid

$$(tag, a) \in \{G, L\} \times \mathbb{N}$$

- NB! Paljud keeled lubavad muutujate nähtavust kitsendada mingi ploki piiresse.
- Erinevate programmiosade transleerimine reeglina kasutab erinevaid aadresskeskondi.

Funktsioonid

```
0  int i;
   struct list {
      int info;
      struct list *next;
   } *l;

1  int ith (struct list *x, int i) {
      if (i ≤ 1) return x → info;
      else return ith(x → next, i−1);
   }

2  main () {
      int k;
      scanf("%d", &i);
      scanlist(&l);
      printf("%d", ith(l, i));
   }
```

---

0 globaalne aadresskeskond

ρ_0 :

i	$\mapsto$	$(G, 1)$
l	$\mapsto$	$(G, 2)$
ith	$\mapsto$	$(G, \_ith)$
$main$	$\mapsto$	$(G, \_main)$

Funktsioonid

```
0   int i;
    struct list {
      int info;
      struct list *next;
    } *l;

1   int ith (struct list *x, int i) {
      if (i <= 1) return x -> info;
      else return ith(x -> next, i-1);
    }

2   main () {
      int k;
      scanf("%d", &i);
      scanlist(&l);
      printf("%d", ith(l, i));
    }
```

---

1 keskond fun-s ith

ρ_1 :

x	$\mapsto$	$(L,1)$
i	$\mapsto$	$(L,2)$
l	$\mapsto$	$(G,2)$
ith	$\mapsto$	$(G,\_ith)$
$main$	$\mapsto$	$(G,\_main)$

Funktsioonid

0
```
int i;
struct list {
    int info;
    struct list *next;
} *l;
```

1
```
int ith (struct list *x, int i) {
    if (i ≤ 1) return x → info;
    else return ith(x → next, i−1);
}
```

2
```
main () {
    int k;
    scanf("%d", &i);
    scanlist(&l);
    printf("%d", ith(l, i));
}
```

2 keskond fun-s main

$$\rho_2 : \quad \begin{array}{rcl} k & \mapsto & (L,1) \\ i & \mapsto & (G,1) \\ l & \mapsto & (G,2) \\ ith & \mapsto & (G,\_ith) \\ main & \mapsto & (G,\_main) \end{array}$$