Andmestruktuur.
Lineaarne andmestruktuur.
Dünaamiline mäluhaldus.
Ahel. Pinu. Järjekord.

Andmestruktuuri mõiste

Andmestruktuur *(data structure)* on andmete talletamise ja organiseerimise viis arvuti mälus või kõvakettal.

Andmestruktuur on mudel, mis kirjeldab suure hulga andmeelementide organiseerimist ja salvestamist arvutis ning neile efektiivse juurdepääsu tagamist.

Erinevad andmestruktuurid sobivad erinevateks rakendusteks, sh mõned nendest on väga spetsiifilisteks eesmärkideks.

Üldist

Andmestruktuure saab nende kuju järgi jaotada:

- lineaarseteks;
- mittelineaarseteks.

Andmestruktuurid tuginevad arvuti võimele salvestada ja võtta andmeid muutmälust või kõvakettalt **aadressi järgi**.

Andmestruktuuriga seotakse tavaliselt ka funktsionaalsused, mida terve struktuuri või seal olevate elementidega teha saab.

Erinevad vaated

Loogiline vaade: kuidas me andmestruktuuri endale ettekujutame, teatud omadustega struktuur:

- andmete omavahelised seosed (eelmine/järgmine, ...);
- operatsioonid (elemendi lisamine, eemaldamine ..).

Realisatsioonivaade: kuidas andmestruktuur tegelikult mälus on, tema füüsiline esitus:

- sama loogilist struktuuri saab tihti realiseerida (valmis teha) mitmel erineval viisil;
- tavaliselt on realisatsioon **staatiline** või **dünaamiline**, eristamine lähtub mäluaadresside kasutamisest;
- mäluaadressid arvutatakse välja või salvestatakes koos andmetega.

Loend

Loend (list, nimistu, *ingl list)* on lõplik järgnevus, mis sisaldab 0 või enam elementi.

Loend on andmeelementide korrastatud kogum (IT terministandardi sõnastik - EVS-ISO 2382-4:1999)

Loendi elementide vahel on järgnevussuhe. Võib rääkida esimesest ja viimasest, eelmisest ja järgmisest elemendist. Elementide järjestusel võib olla eriline tähendus, aga ei pruugi.

See on andmestruktuuri loogiline vaade, mida saab realiseerida staatiliselt või dünaamiliselt.

Loendi omadused ja toimingud

Omadused:

- Loendi elementide arv on ideaalis tõkestamata.
- Kõik elemendid on ühesuguse struktuuriga.

Toimingud loendiga:

- elemendi lisamine ja kustutamine (erinevad variandid: alguses, lõpus, ...);
- loendi läbimine, poolitamine, pööramine, koopia tegemine, elementide arvu leidmine;
- mitme loendi ühendamine;
- sorteerimine ja elemendi otsimine.

Loendi realisatsioon

Lineaarloend *(ingl linear list)*

Lineaarselt järjestatud andmeelementide kogum, mille elementide järjestus on mälus järjestikpaigutusega säilitatud. (EVS-ISO 2382-4:1999)

Ahelloend *(ingl linked list)*

Loend, mille andmeelemendid võivad olla mälus hajutatud, kuid iga andmeelement sisaldab informatsiooni järgmise elemendi asukoha kohta.

(EVS-ISO 2382-4:1999)

Staatiline mäluhaldus

Deklareerides C-s muutujad:

- eraldatakse nendele vajalik kogus muutmälu (vastavalt andmetüübile);
- peetakse meeles mälupesade algusaadress;
- ka kõigile massiivi elementidele eraldatakse mälu;
- muutujate väärtused säiluvad oma pesades kuni programmi töö lõpuni (või ploki lõpuni).

Muutujaga "suhtlemiseks" (omistamine, väärtuse küsimine) on vajalik tema mäluaadress.

- Tavaliselt toimub see automaatselt muutuja nime kaudu.

Massiiv

Massiiv *(ingl array)* on madala taseme staatiline lineaarne struktuur.

- määratud suurusega (st elementide arvuga), paikneb mälus **järjestikustes mälupesades**;
- kõik elemendid on sama andmetüüpi;
- indeksi järgi otsimine ja väärtuse asendamine on kiired operatsioonid;
- elemendi lisamine vahele ja kustutamine aga aeglased operatsioonid – miks?

Massiivi abil saab ehitada **lineaarloendi**.

Dünaamiline mäluhaldus

Andmete salvestamiseks saab mälu eraldada programmi töö käigus vastavate käskudega.

Mälu saab ka vabastada, kui andmeid enam vaja ei lähe – kasulik suurte andmestike korral.

Mälu saab kasutada dünaamiliselt **viitade** *(ingl pointer)* abil.

Viit on andmeelement, mis näitab teise andmeelemendi asukohta (ISO/IEC 2382).

Mäluaadress on arvuline väärtus, mida on tavaks esitada kuueteistkümnendsüsteemis.

Ahelloend

Ahelloend *(*ingl *linked list)* on madala taseme struktuur, mis koosneb viitade abil ühendatud elementidest (sõlmedest).

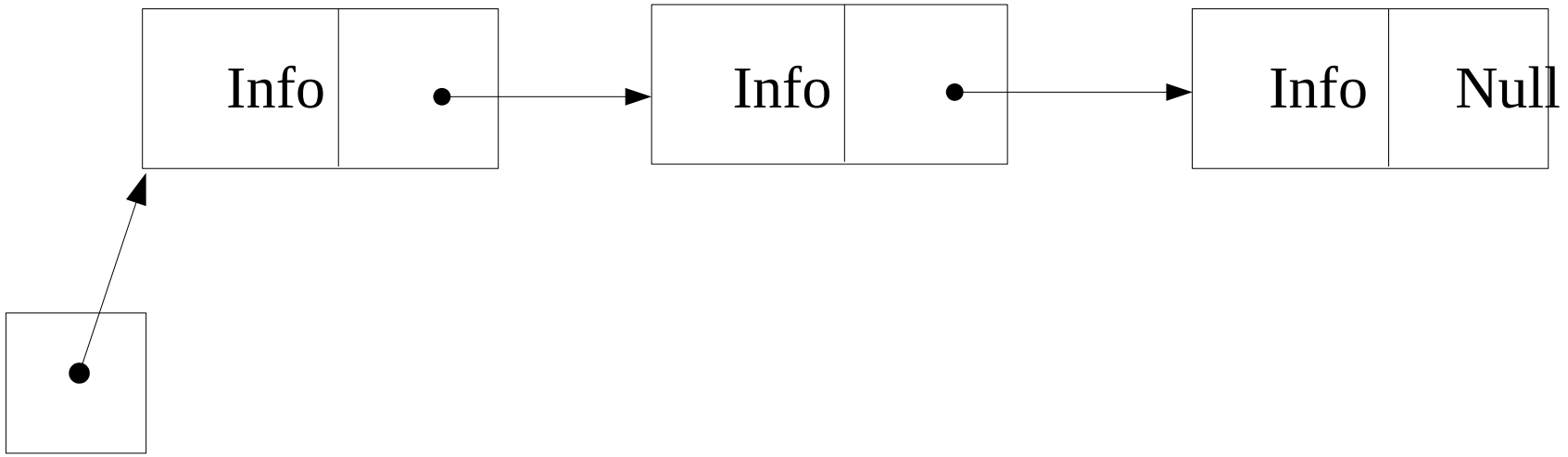
Ahela abil saame ehitada loendi dünaamiliselt.

Sõlm *(*ingl *node)* koosneb vähemalt ühest infoväljast ja vähemalt ühest aadressi- ehk viidaväljast.

Viidaväljade abil ühendatakse sõlmed ühte struktuuri: ahelloendiks.

Ahelloendi ülesehitus

Ettekujutus ühe viidaga ahelloendist (joonistuskeele idee aastast 1957):



Ristkülikud on ahelloendi sõlmed **info- ja viidaväljaga**.

Viidavälja sisuks saab olla vaid aadress.

Ühest elemendist teise näitav täpiga nool tähistab viidavälja salvestatud järgmise elemendi aadressi. Päris mäluaadressi kirjutades läheb pilt tunduvalt segasemaks.

Ahelloendi ülesehitus

Esimene sõlm: **pea** (ingl *head*)

Viimane sõlm: **saba** (ingl *tail*)

On teada ahelloendi esimese sõlme aadress (eraldi väike kast, mis võib ka jooniselt puududa).

Järgmised sõlmed on võimalik üles leida esimesest sõlmest lähtudes.

Viimase sõlme viidaväljas on "`null`"-aadress.
Selle järgi saab aru, et ahelloend lõppes ja rohkem elemente ei ole.

Ahelloendi töötlemine

Ahelloendis saab teha sarnaseid tegevusi massiiviga:

- Elemendi (sõlme) lisamine ja kustutamine ahelloendi keskel on kiired toimingud.
- Otsepöördumine vajaliku elemendi (sõlme) poole on aeglane tegevus.

Miks?

Kuidas oli massiivis?

Ahelloendi töötlemine

Nii lineaarloendi (massiiv) kui ahelloendiga saab teha mitmesuguseid operatsioone.

Ahelloend tuleb läbida iga sõlmeni jõudmiseks.

```
current = head

while current != NULL

      // Tegevus sõlmega, liikumine edasi

       current = current.next
```

Jooniseid vaata jutustavast materjalist. Samuti pseudokoodi täpsemat tähendust.

`head` – esimese elemendi aadress, `next` viidavälja nimetus, `current` – suvaline aadress.

Kuidas toimub sarnane tegevus massiivis?

Elemendi lisamine algusse

Pseudokood:

```
New(node)              // uus sõlm
node.info = X          // info sõlme
node.next = head       // viit uuest sõlmest vanale
head = node            // ahela pea uuele sõlmele
```

Uus sõlm lisatakse ahelloendi algusesse. Selleks kirjutatakse tema viidavälja ahela 1. sõlme aadress.

Kuidas lisada väärtus massiivi algusesse?

Kohtab ka sellist mõistet, nagu dünaamiline massiiv *(ingl dynamic array)*.
Mis "elukas" see veel on?

Lineaarsed struktuurid massiivi ja ahelloendit kasutades (nn abstraktsed andmetüübid). Ülesehitus ja funktsioonid.

Järjekord

Järjekord e. **rivi** (ingl *queue*) on lineaarloend, kus andmed lisatakse loendi lõppu ja kustutatakse loendi algusest.

Loend, mille ehitusviis ja hooldus on sellised, et järgmisena võetakse kõige varasemana salvestatud andmeelement. Seda meetodit iseloomustatakse väljendiga „esimesena sisse, esimesena välja„ (FIFO-printsiip) (EVS-ISO 2382-4:1999).

Järjekorra operatsioonid

Andmeelemendi lisamine järjekorda *(ingl enqueue)*

Andmeelemendi eemaldamine järjekorrast *(ingl dequeue)*

Uue tühja järjekorra loomine

Kontroll, kas järjekord on tühi

Kontroll, kas järjekord on täis

Järjekord

Realisatsioon: massiiviga või ahelloendiga.

Massiiv – vajalikud kaks indeksit, millega järjekorra algust ja lõppu meeles pidada.

Ahelloend – vajalikud kaks viita: esimesele ja viimasele sõlmele, et lisamine ja kustutamine oleks kiire ja mugav.

(vt täpsemalt pikas materjalis)

Järjekord

Kasutamine: mitmed reaalelulised olukorrad, kus mitmest allikast lähtuvad päringud tuleb nende saabumise alusel teenindada.

Eriliigid:

- Mitme teenindajaga järjekord *(ingl multiple server queue)*;
- Prioriteetidega järjekord ehk eelistusjärjekord *(ingl priority queue)*;
- Piiratud pikkusega järjekord *(ingl bounded length queue)*.

Pinu

Magasin e. **pinu** *(ingl stack)* on loend, kus elemendid lisatakse ja kustutatakse samas otsas, nn pinu **tipus** *(ingl top)*. Andmete kättesaasmine toimub reeglina sel teel, et element kustutatakse pinust.

Loend, mille ehitusviis ja hooldus on sellised, et järgmisena võetakse kõige hilisemana salvestatud andmeelement. Seda meetodit iseloomustatakse väljendiga „viimasena sisse, esimesena välja„ (LIFO-printsiip) (EVS-ISO 2382-4:1999)

Pinu operatsioonid

Andmeelemendi lisamine pinusse *(ingl push)*

Andmeelemendi eemaldamine pinust *(ingl pop)*

Uue pinu loomine

Kontroll, kas pinu on tühi

Kontroll, kas pinu on täis

Pinu

Realisatsioon: massiiviga ja ahelloendiga.

Massiiv – vajalik üks indeks, millega pinu tippu meeles pidada.

Ahelloend – vajalik üks viit esimesele sõlmele, kus elemente lisada ja eemaldada.

(vt täpsemalt pikas materjalis)

Pinu

Kasutamine

Igapäevaelus tavaliselt ei kohta. Kasutusalad peamiselt arvutiteaduses:

- Funktsioonide väljakutsete organiseerimine.
- Avaldiste teisendamine, kontrollimine, arvutamine.
- Abivahend andmete hoidmisel, kus viimati lisatud väärtust tuleb kohe töötlema hakata.

Viidad C-s.Deklareerimine

Viida tüüpi *(pointer)* muutujasse salvestatakse aadresse. Viitmuutujad on soovitavalt tüpiseeritud ehk nad on ettenähtud sisaldama teatud tüüpi väärtuse aadressi.

```
int *ptr;
```

Viidatav väärtus peab eelneva deklaratsiooni kohaselt olema täisarv `int`. Muutuja nimeks on `ptr` ja temasse saab salvestada täisarvu aadressi.

Dünaamiline mälu eraldamine:

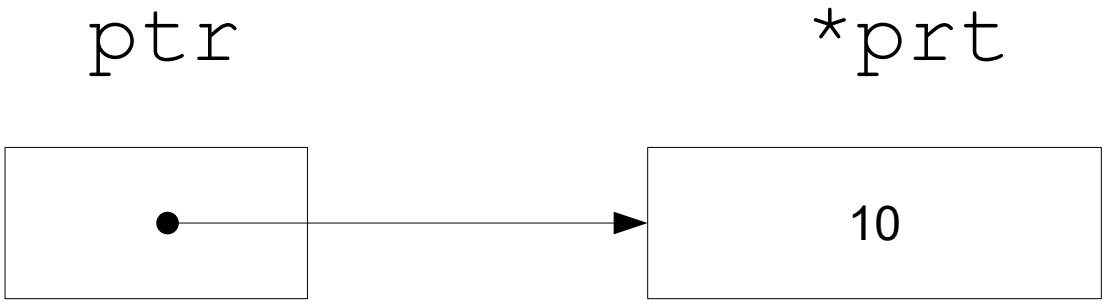
```
ptr = malloc(sizeof *ptr);
```

Eraldatakse mälu ühe `int` jagu. Aadress salvestatakse muutujasse `ptr`.

Viidad C-s.Omistamine

```
*ptr = 10;
```

Funktsiooniga `malloc()` eraldatud mäluvälja kirjutatakse väärtus 10.



Nähtavat joonist tavatsetakse kasutada viitade (viitmuutujate) visualiseerimiseks.

Viidad C-s.Aadressi küsimine.

Muutuja aadressi küsimine.

```
int arv;
```

`&arv` – muutuja `arv` mäluaadress

`ptr = &arv;` - muutujasse `ptr` omistatakse muutuja `arv` aadress.

Aadressidega saab arvutada: `ptr++`

Muutuja `ptr` väärtus suureneb 4 baidi võrra ehk aadress "nihkub" 4 baiti "edasi", sest `int`-i suurus on 4 baiti.

Mis on `&ptr` väärtuseks?

Viidad C-s.Massiiv

Massiivist:

`int arvud[10];` - massiiv 10 täisarvu jaoks (mälu eraldatakse deklaratsiooni tulemusena)

`arvud` – massiivi algusaadress (esimese elemendi aadress)

Massiivi saab deklareerida ka dünaamiliselt:

`int *arvud;` - veel ei ole massiivi, mälu on algusaadressi jaoks

```
arvud = malloc(10*sizsof(int));
```

Mälu eraldatakse massiivi 10 `int`-i jaoks alles `malloc`-lause täitmisel.

Milleks ometi on seda vaja?

Viidad on C loomulik koostisosa, võimaldades tegeleda madalama taseme programmeerimisega ja loodetavasti aitavad nad paremini mõista, mis on "karul kõhus".

Viitasid kasutades saab moodustada huvitavaid lineaarseid ja mittelineaarseid andmestruktuure. Seda me ka kursusel katsetame.

C-d (ja tema viitasid) kasutatakse teistes keeltes struktuursete andmetüüpide realiseerimiseks.