DUOMENŲ STRUKTŪROS IR ALGORITMAI

Tiesinės dinaminės duomenų struktūros:

tiesinis sąrašas, stekas, eilė, dekas

Praeitos paskaitos santrauka

- Duomenys, duomenų tipai
- Duomenų struktūros apibrėžimas
- Algoritmo apibrėžimas
- Masyvai
  - Savybės (elementai, indeksai, dydis)
  - Naujo elemento pridėjimas
  - Elemento trinimas
  - Elemento paieška

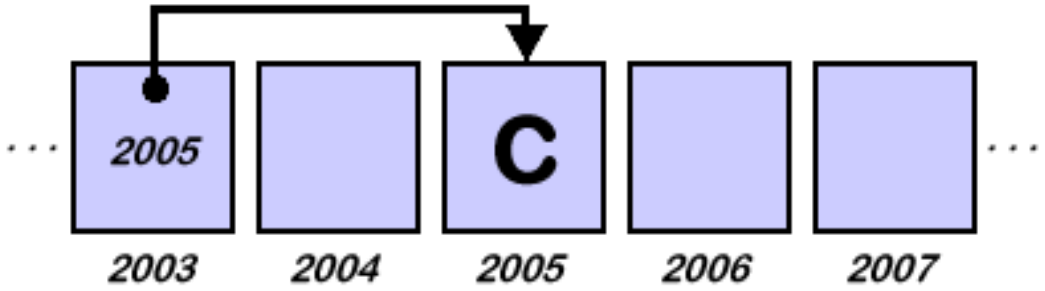
Prieš pradedant naują temą...

Rodyklė (Pointer)

Norint atvaizduoti duomenų struktūras, ypač netiesines, operatyviojoje atmintyje, kuri laikoma tiesine struktūra, reikia naudoti rodykles.

Rodyklė – tai atminties ląstelė, kurioje saugomas kitos ląstelės adresas. Naudojant rodykles, netiesiogiai pasiekiami duomenys, esantys ląstelėje, kurios adresą žino rodyklė.

Naudojant rodykles, galima gauti keleto lygių netiesioginį kreipimąsi.



Kodėl dinaminės duomenų struktūros?

Masyvo trūkumai:

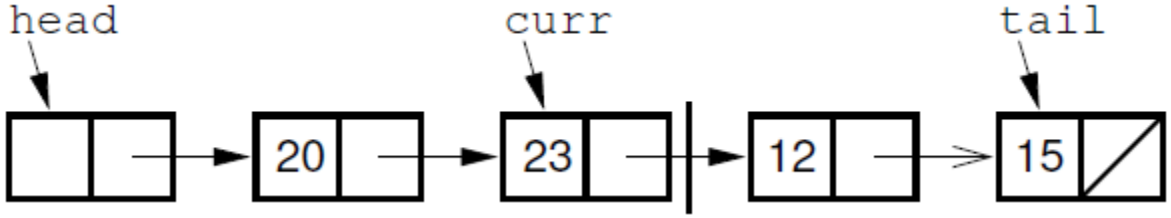
- Saugomų elementų skaičius nėra iš anksto žinomas arba jis keičiasi uždavinio sprendimo metu, todėl neaišku, kokio dydžio masyvą reikia apibrėžti;
- Nauji elementai gali būti įterpiami į masyvo pabaigą;
- Bet kurio (išskyrus paskutinio) elemento šalinimas yra problemiškas.

Tiesinis sąrašas

Tiesinis dinaminis sąrašas (angl. *list*)– tai pradžios rodyklė ir rodyklėmis susietų vienodų elementų aibė.

Surūšiuotas sąrašas panašus į studentų sąrašą išdėstytą abėcėlės tvarka. Pridedant naują įrašą, jis talpinamas į reikiamą vietą, todėl sąrašas visada išlieka išrūšiuotas.

Sąrašą sudaro elementai. Elementai susideda iš duomenų ir rodyklės. Sąrašo pradžios rodyklė turi pirmojo sąrašo elemento adresą, kuriuo pradedant galima nuosekliai pereiti per visus sarašo elementus.

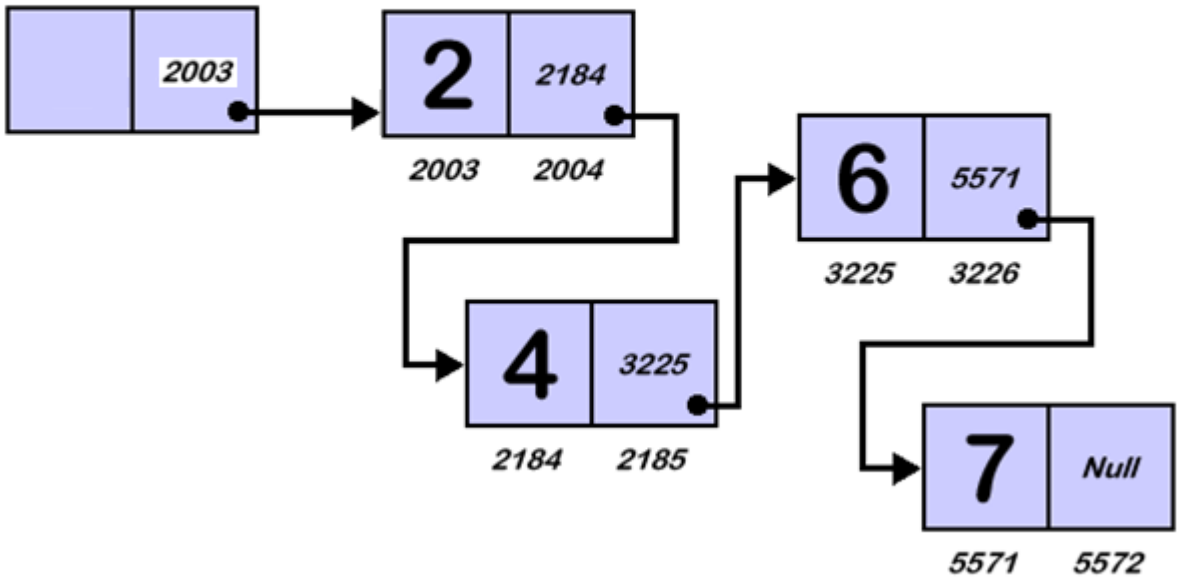


Sąrašo savybės

Tiesinio dinaminio sąrašo savybės:

- Visi elementai yra to paties duomenų tipo;
- Kiekvienas elementas turi rodyklę, rodančią į vieną kaimyninį elementą;
- Paskutinio sąrašo elemento rodyklė yra tuščia. Tai sąrašo pabaigos požymis.
- Elementai saugomi bet kurioje atminties vietoje;
- Sąrašas yra laikomas tuščias, kai neturi nei vieno elemento.
- Sąrašo ilgis – tai elementų skaičius sąraše.

Sąrašo pavyzdys



Paskutinio sąrašo elemento rodyklė tuščia. Tai požymis, kad pasiektas paskutinis sąrašo elementas.

Sąrašo tipai

Tiesinio dinaminio sąrašo tipai:

- Vienkryptis tiesinis sąrašas;
- Dvikryptis tiesinis sąrašas;
- Vienkryptis ciklinis sąrašas;
- Dvikryptis ciklinis sąrašas

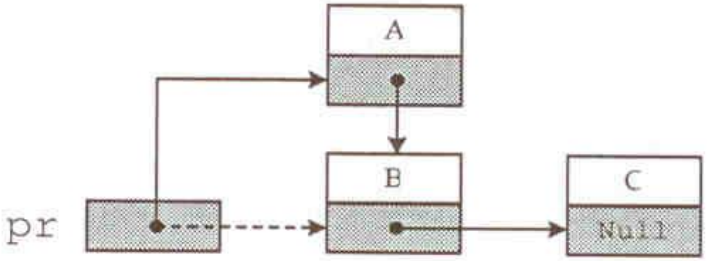
Sąrašo veiksmai

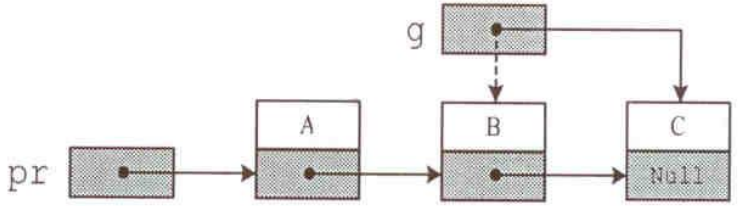
Tiesinis dinaminis sąrašas gali atlikti tokias operacijas:

- Įterpti elementą bet kurioje sąrašo vietoje
- Ištrinti elementą bet kurioje sąrašo vietoje
- Surasti bet kurį elementą pagal jo reikšmę
- Ištuštinti sąrašą
- Pasiekti bet kurį elementą tik nuo sąrašo pradžios, paeiliui perėjus visus prieš jį esančius elementus.

Naujo sąrašo sudarymas

Naujas sąrašas gali būti sudaromas naudojant tiesioginės ir atvirkštinės kelties algoritmus.





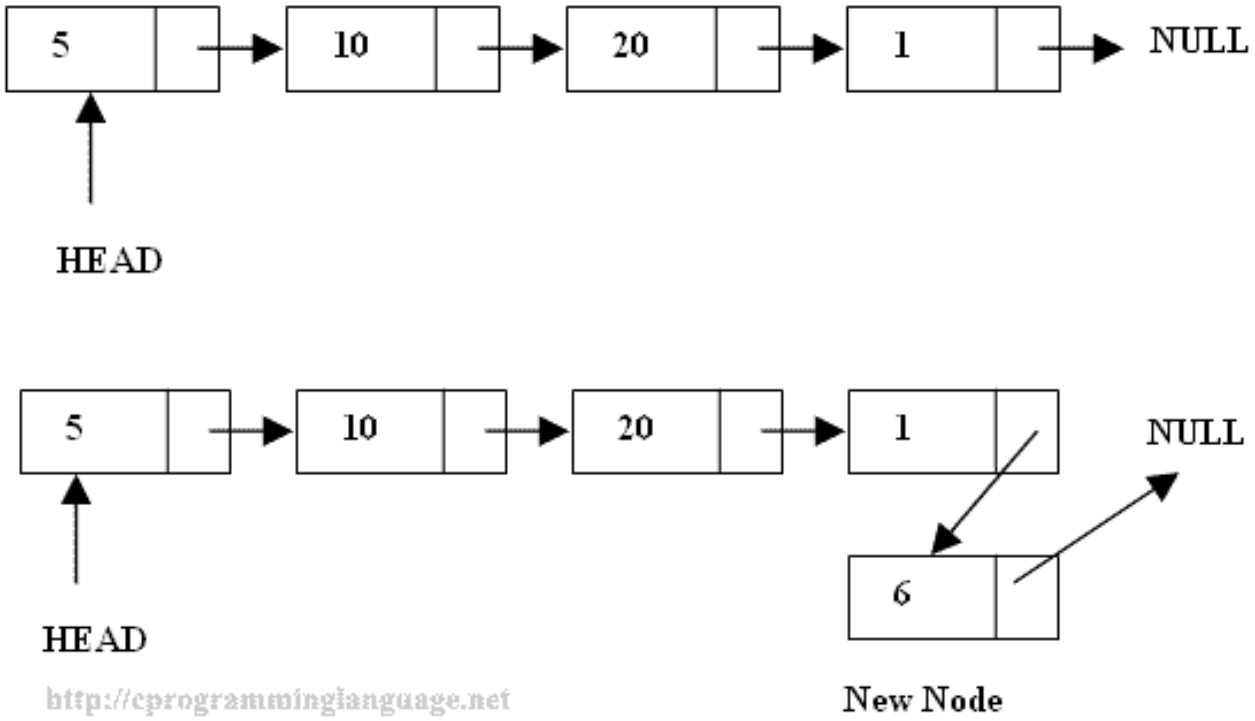
Atvirkštinė keltis
(naujas elementas įrašomas į pradžią)

Tiesioginė keltis
(naujas elementas į pabaigą)

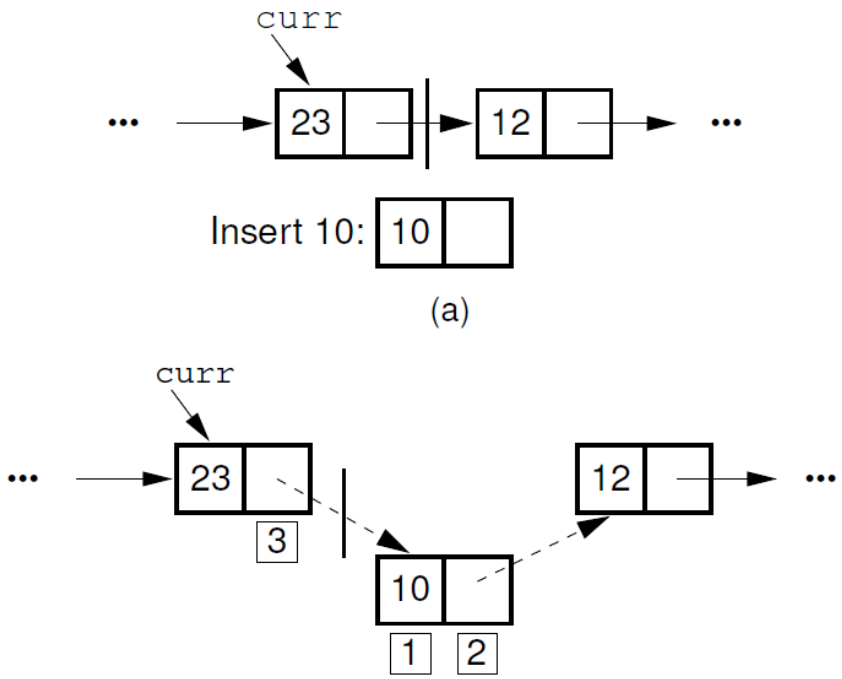
pr – sąrašo pradžios rodyklė

g – sąrašo galo rodyklė

Naujo elemento pridėjimas



Naujo elemento įterpimas



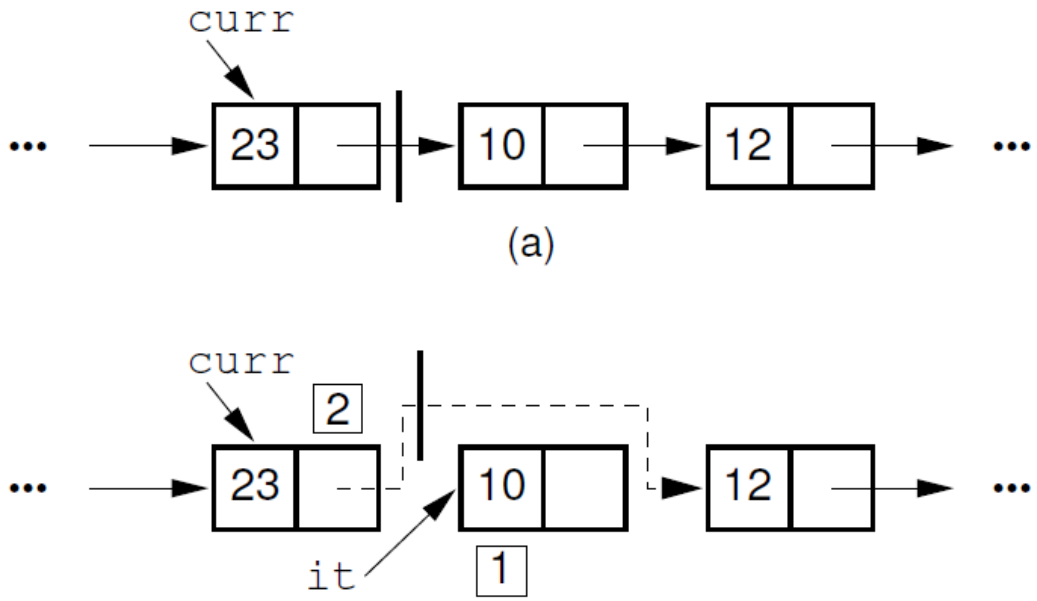
- Surandamas elementas, po kurio bus įterpiamas naujas elementas (pavyzdyje **23**)
- Elemento rodyklei priskiriama naujo elemento (pvz. **10**) adresas
- Naujo elemento rodyklei priskiriamas sekančio elemento adresas (pvz. **12**)

Elemento paieška

- Atliekant elemento paiešką, tikrinama ar atitinka elemento duomenų lauko reikšmė ieškomai reikšmei.
- Judėjimui per elementus, naudojamas ciklas, kuriame rodyklei priskiriama vis kito elemento adresas. Ciklas baigiamas, kai randamas skaičius arba pasiekiama sąrašo pabaiga.

```
while (node-> next  != NULL )
  { if ( node->data == skaicius)
        {  cout << “Elementas rastas”<< endl;
             exit (0);
          }
    node = node-> next;
  }
```

Elemento šalinimas



- Surandamas elementas, kurį reikia ištrinti (pvz. **10**) ir pasižymima nuoroda į tą elementą (pvz. **it**)
- Surandamas prieš jį buvęs elementas (pvz. **23**) ir jo rodyklės lauko reikšmę pakeičiama į sekančio elemento (pvz. **12**) reikšmę.

Sąrašo elementas

```
struct node {
        int data;
        struct node *next;
};
```

```
class node {
        private: int data;
                 node *next;
        public:
                 node()
                 { }
    };
```

Pavyzdys

```
void add (struct node *head, int data){
        struct node *tmp;
        if (head  ==  NULL){
                head = (struct node *) malloc(sizeof(struct node));
                if (head == NULL){
                        printf (“Klaida! Truksta atminitiesn");
                        exit(0);    }
                head-> data = data;
                head-> next = head;
        }else {
                tmp = head;
                while (tmp-> next  != NULL)
                        tmp = tmp-> next;
                tmp-> next = (struct node *)malloc(sizeof(struct node));
                if(tmp -> next == NULL)
                {       printf(" Klaida! Truksta atminities \n");
                        exit(0);
                }
                tmp = tmp-> next;
                tmp-> data = data;
                tmp-> next = NULL;
        }  }
```

Pavyzdys

```
void printlist(struct node *head)
{
        struct node *current;
        current = head;
        if(current != NULL)
        {
        do
        {  printf("%d\t",current->data);
           current = current->next;
        }
             while (current != NULL);

        printf("\n");
        }
        else
        printf("The list is empty\n");
 }
```

```
 void destroy(struct node *head)
{
    struct node *current, *tmp;

     current = head->next;
     head->next = NULL;

      while(current != NULL) {
          tmp = current->next;
          free(current);
          current = tmp;
       }
}
```

```
void remove (struct  node *e)
{  struct node *d = e->next;
    if ( d != NULL )
        e->next = d->next;
     free(d);
}
```

Pavyzdys

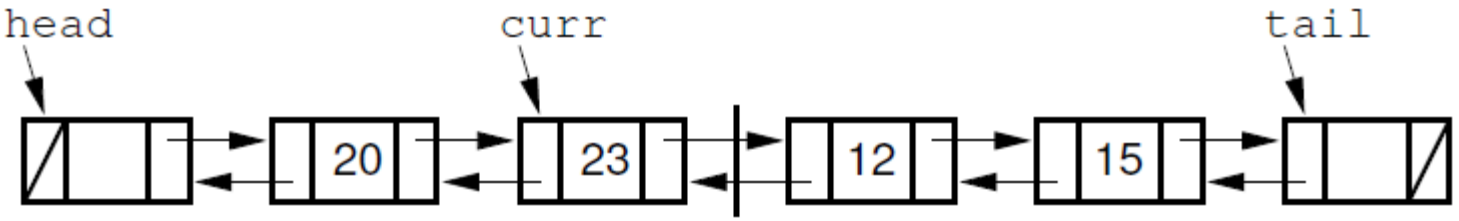
```
void main()
{
        struct node *head = NULL;
        head = add(head,1);       /* 1 */
        printlist(head);

        head = add(head,20);      /* 1 20 */
        printlist(head);

        head = add(head,10);      /* 1 20 10 */
        printlist(head);

        head = add(head,5);       /* 1 20 10 5*/
        printlist(head);

        destroy(head);
        getchar();
}
```

Dvikryptis tiesinis sąrašas

Dvikryptis dinaminis tiesinis sąrašas nuo vienkrypčio skiriasi tuo, kad kiekvienas elementas turi dvi rodykles, rodančias į prieš ir už jį esančius elementus.

Dvikryptis sąrašas gali būti įsivaizduojamas kaip priešingomis kryptimis suklijuotus du vienkrypčius sąrašus.

Realizuojant dvikryptį sąrašą, reikia turėti sąrašo pradžios ir pabaigos rodykles.



Ciklinis vienkryptis sąrašas

Ciklinis vienkryptis sąrašas – tai vienkryptis sąrašas, kurio paskutinis elementas yra gretimas pirmam.

Toks sąrašas naudojamas aprašant žiediniu maršrutu judančio objekto sustojimų aibę. Baigus ratą, vėl bus pereinama prie pirmojo sustojimo.

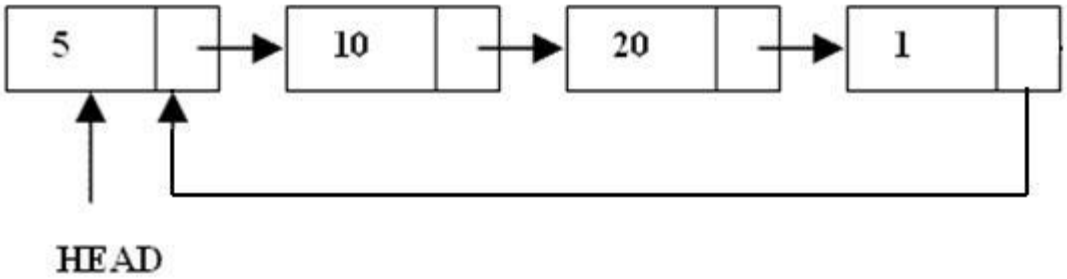
Ciklinio sąrašo paskutinio elemento rodyklė sutampa su sąrašo pradžios rodykle ir yra sąrašo pabaigos požymis. Skaičiavimų pabaiga nustatoma, lyginant tarpinę rodyklę P su sąrašo pradžios rodykle.

Pavyzdys

Kuris vaikas bus pasirinktas paskutinis?

Tarkime, kad N vaikų sustoja ratu, jie atsitiktinai pasirenka skaičių ir vaiką, nuo kurio bus pradėta skaičiuotė. Tada laikrodžio judėjimo kryptimi suskaičiuojamas vaikas, kuris palieka ratą. Skaičiuotė˙ tęsiama nuo sekančio vaiko.

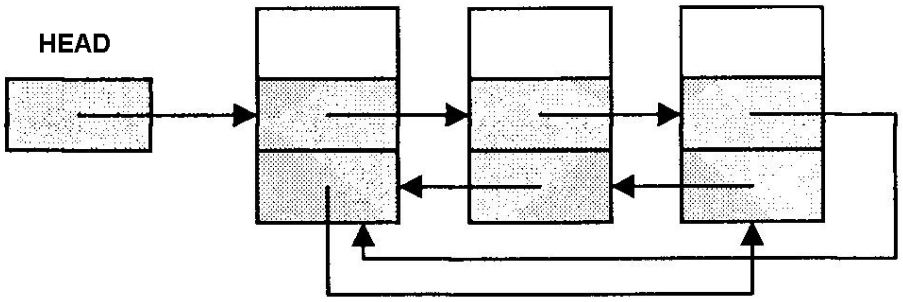
Ciklinis sąrašas naudojamas kaip duomenų struktūra eilei realizuoti.



Dvikryptis ciklinis sąrašas

Kiekvienas dvikrypčio sąrašo elementas turi dvi rodykles: *kitas* ir *ankstesnis*. Tokio sąrašo paskutinio elemento rodyklė *kitas* rodo į pirmąjį, o pirmojo rodyklė *ankstesnis* – į paskutinį elementą.

Dvikrypčio ciklinio sąrašo pradžiai surasti naudojamas fiktyvus elementas, kurio rodyklė *kitas* rodo į sąrašo pirmąjį elementą, o rodyklė *ankstesnis* – į pabaigą.



Stekas

Stekas (angl. stack) arba dėklas – tai vienkrypčio sąrašo tipo duomenų struktūra, kurioje elementas gali būti pridėtas ir pašalintas tik iš galo principas "paskutinis įeina, pirmasis išeina“ (LIFO).

Stekas buityje:

- kaladėlių bokšto statyba ir griovimas;
- knygų krovimas į dėžę ir išėmimas iš jos ir t.t.

Steko realizacija

Stekas gali būti realizuotas panaudojus masyvą arba tiesinį dinaminį sąrašą.

- **Stekas-masyvas** naudoja indeksą, kad nustatyti viršutinį elementą. Taip pat **papildomą kintamąjį**, kuris parodo elementų skaičių steke.

- **Stekas-sąrašas** naudoja tokio pat tipo elementus, kaip tiesinis vienkryptis sąrašas, o steko **pirmojo elemento rodyklė rodo į viršutinį steko elementą**. Steko – sąrašo paskutinio elemento rodyklė yra tuščia.

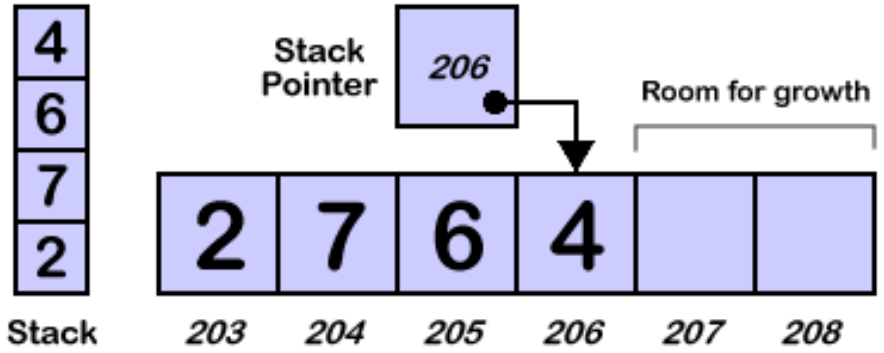
Stekas

Stekas pasiekiamas per viršutinį sąrašo elementą t.y. **viršūnę**, todėl steko rodyklė rodo į paskutinį elementą (dinaminiame sąraše).

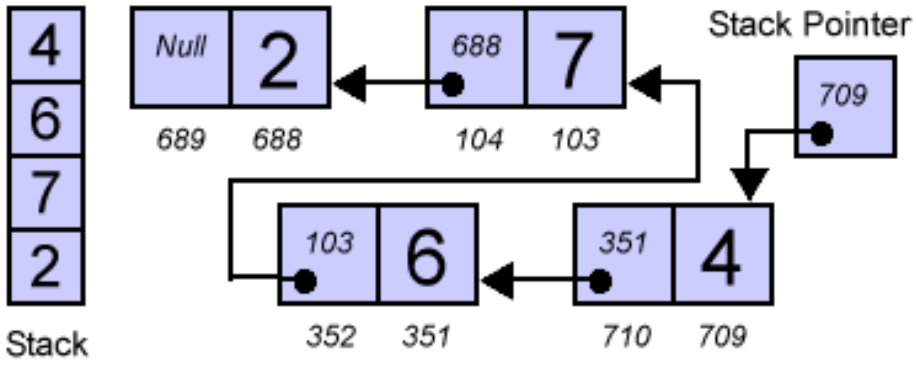
Jei stekui naudojamas masyvas, viršūnė randama naudojant **paskutinio elemento indeksą**.

Veiksmai su steku:

- Elemento pridėjimas (**push**)
- Elemento trinimas (**pop**).



Stekas



Steko, realizuoto kaip **tiesinis sąrašas**, elementai susideda iš reikšmės lauko ir rodyklės į gretimą elementą.

Steko rodyklė rodo į steko viršūnę.

Steko veiksmai (funkcijos)

push(stack, new-item)
Pridėti naują elementą.

item **top**()
Grąžina steko viršūnės elementą.

item **pop**()
Ištrina steko viršūnės elementą.

bool **is-empty**()
Grąžina teisybė, jei tuščias.

bool **is-full**()
Grąžina teisybė, jei pilnas ir daugiau negali būti pildomas.

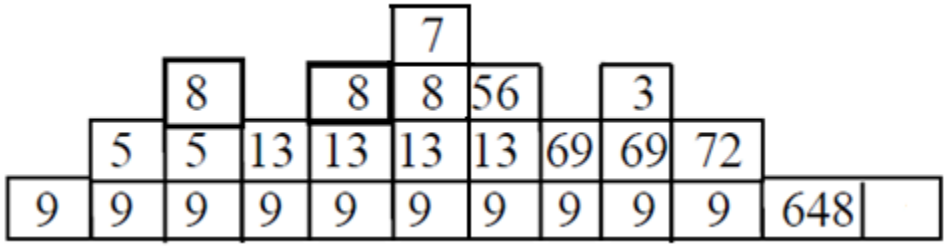
int **get-size**()
Grąžina steko elementų skaičių.

Steko pavyzdys

Suskaičiuokime išraišką:

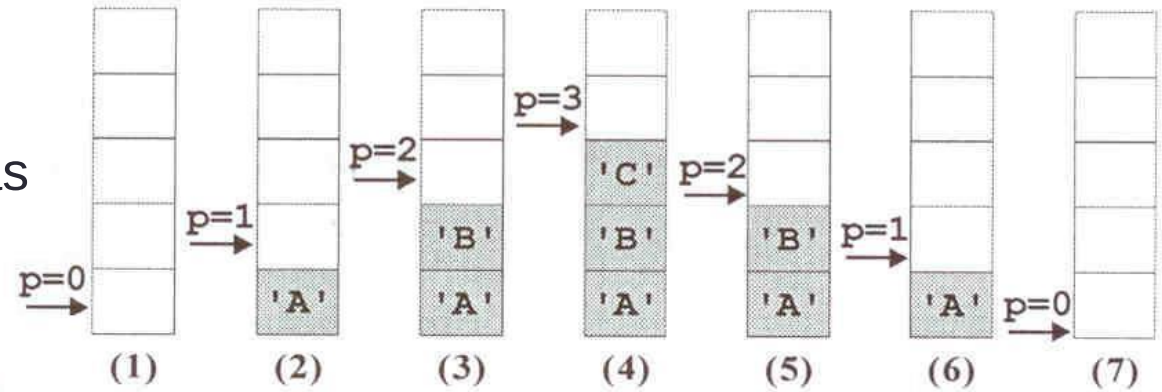
9*(((5+8)+(8*7))+3)

- *push(9);*
- *push(5);*
- *push(8);*
- *push(pop+pop);*
- *push(8);*
- *push(7);*
- *push(pop*pop);*
- *push(pop+pop);*
- *push(3);*
- *push(pop+pop);*
- *push(pop*pop);*
- *write(pop).*

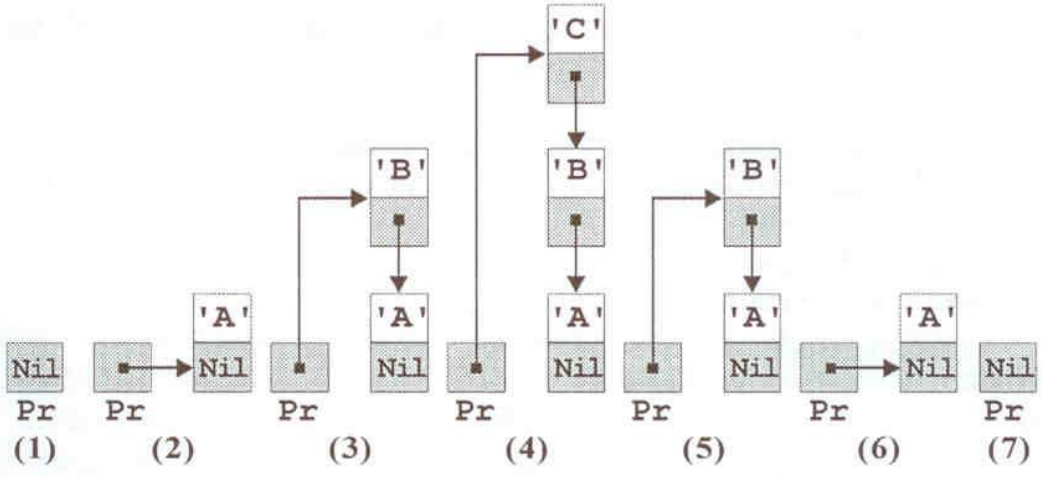


Steko sukūrimas

Stekas realizuotas masyve



Stekas, realizuotas vienkrypčiame sąraše



Steko realizacija masyve

Stack.h

```
void push (int *s,int* top, int element);
int pop (int *s,int *top);
int full (int *top,const int size);
int empty (int *top);
void init (int top);
```

Steko realizacija masyve

```
//  Steko rodyklė t.y. viršūnės indekso numeris
void init (int *top)
{
    * top = 0;
}


//  Steko pildymas
void push (int *s, int *top, int element)
{  if (*top < N)
       s[*top] = element;
       *top++;
}
```

Steko realizacija masyve

```
/* Elemento šalinimas */
int pop (int *s, int *top)
{ if (*top > 0)
   return s[*top--];
}

/* Tikrinama, ar stekas pilnas (size – steko max elementų skaičius  */
int full (int *top, const int size)
{
   return *top == size ? 1 : 0;
}

/* Tikrinama, ar stekas tuščias   */
int empty (int *top)
{
   return *top == 0 ? 1 : 0;
}
```

Steko realizacija masyve

```
#include <stdio.h>
#include <stdlib.h>
#include "stack.h"
#define size 3
void main()
{
   int top, element;
   int stack[size];
   // steko inicializacija
   init (&top);

   // pridedamas naujas elementas
   while (! full(&top,size) ) {
            element = rand();
            printf („Elementas %d pridetas\n", element);
            push(stack,&top,element);
            getchar();
    }
```

```
// elemento šalinimas iš steko

while (!empty(&top)){
    element = pop(stack,&top);
    printf(“Elementas %d
            pasalintas\n",element);
   getchar();
}

printf (“Stekas tuscias\n");
getchar();
}

}
```

Steko realizacija sąraše

Steko pavyzdys, naudojant tiesinį dinaminį sąrašą.

Elementas:

```
struct node
{       int data;
        struct node *next;
};
```

Steko Pildymas:

```
void push (int skaicius) {
    node *v;
    v = new node;
    v->data = skaicius;
    if ( listStart == NULL)    // Pirmas el.
            v->next = NULL ;
    else
            v->next = listStart->next;
    listStart = v;
}
```

Steko realizacija sąraše

Steko pavyzdys, naudojant tiesinį dinaminį sąrašą.

Šalinimas:

```
int pop() {
node *v;
   if ( listStart == NULL)
            return 1 ;
    else {
             v = listStart;
             listStart = v->next;
             delete (v);
             }
return 0;
}
```

Skaityti pirmą elementą:

```
int isvesti() {
node *v;
v = listStart;
while ( v != NULL)
{     printf("%d ", v ->data);
      v = v->next;
}
  else
       return EXIT_FAILURE;

return 0;
}
```

Eilė

Eilė (*angl. queue*) – tai tiesinė, sąrašo tipo duomenų struktūra, kurioje nauji elementai pridedami į sąrašo galą, o šalinami iš sąrašo pradžios.

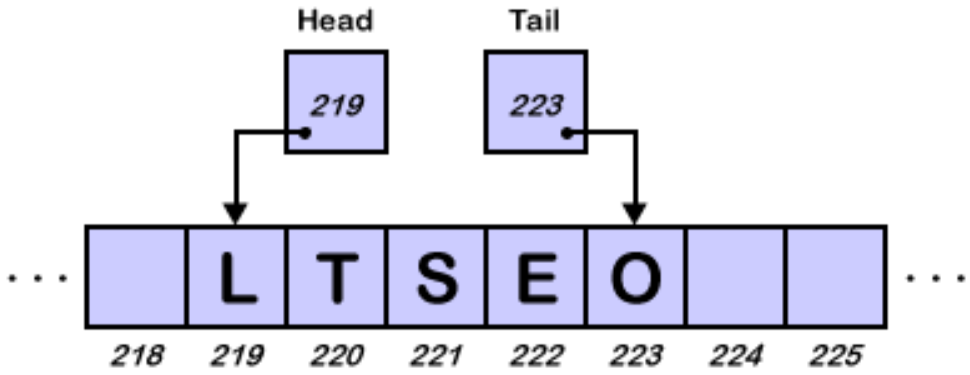
Eilės tipo struktūrą apibūdina principas "First-In, First-Out" arba **FIFO**.

Eilės pavyzdys:

Eilę galime įsivaizduoti pirkėjų eilę stovinčią prie kasos. Naujai atėjęs pirkėjas stoja į eilės galą, o aptarnaujamas pirkėjas, esantis eilės pradžioje.

Eilė gali būti realizuojama **cikliniame masyve** arba naudojant **tiesinį sąrašą.** Bet kuriuo atveju reikalingos dvi rodyklės t.y. į eilės pradžią ir pabaigą. Naudojant ciklinį sąrašą, galima naudoti tik vieną sąrašo pradžios rodyklę, bet galo rodyklę pasiekiama per rekursiją.

Eilė



Eilės veiksmai:

- PridetiElementa (queue, node);
- node TrintiElementa (queue);

PridetiElementa (queue, node) – prideda naują elementą į eilės galą.

TrintiElementa (queue) – ištrina elementą iš eilės priekio ir grąžina tą elementą.

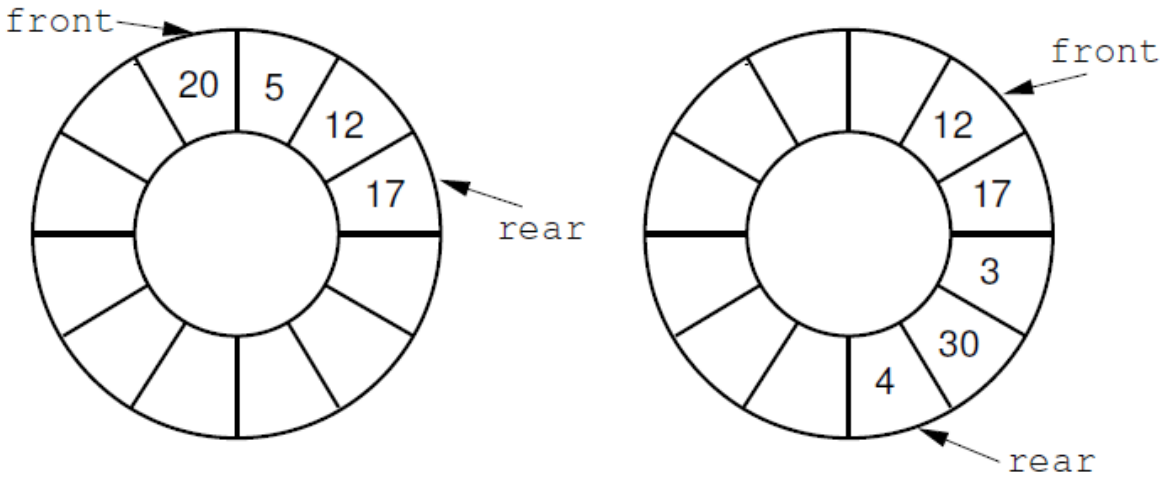
Eilės realizacija (masyvas)

Sakykim, kad turime **n** elementų eilėje. Tegul elementai bus saugomi pirmosiose **n** masyvo pozicijose.

Elementai įterpiami į galą, o šalinami iš pradžios, todėl eilė juda masyve. Tam tikslui turime sujungti masyvo galus į ciklą t.y. po elemento *data[N-1]* eina *data[0]*.

Toks eilės tipas dar vadinamas cikline eile.

Ciklinė eilė



Front – eilės pradžia

Rear – eilės galas

Eilės realizacija (masyvas)

queue.h

```
void enqueue (int *q,int *tail, int element);
int dequeue (int *q,int *head);
int full (int tail,const int size);
int empty (int head,int tail);
void init (int *head, int *tail);
```

Eilės realizacija (masyvas)

```
//Inicializacija
void  init ( int *head, int *tail) {
*head = *tail = 0;}

/* Pridedame elementą į galą, netikrinama, ar eilė pilna */
void  enqueue ( int *q,int *tail, int element) {
 q[(*tail)++] = element;}

/* Trinamas elementas iš pradžios*/
int  dequeue ( int *q,int *head) {
return q[(*head)++];}

/* Tikrinama ar eilė pilna:  1 – pilna,  0 – nepilna */
int  full ( int tail,const int size){
return tail == size ? 1 : 0;}

Tikrinama ar eilė tuščia:  1 – tuščia,  0 – netuščia */
int  empty ( int head, int tail ) {  return head == tail ? 1 : 0; }
```

Eilės realizacija (masyvas)

```
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"
#define size 3

void main(){
int head,tail,element; int queue[size];
// inizializacija
init(&head,&tail);
// pridedami elmenetai
while(full(tail, size) != 1) {
        element = rand();
        printf("enqueue element %d\n",element);
        enqueue(queue,&tail,element);
         printf("head=%d,tail=%d\n",head,tail);
       getchar(); }
printf("queue is full\n");
```

```
// trinami elementai
while(!empty(head,tail)){
    element =  queue[head];
     dequeue(queue,&head);
     printf("dequeue element %d
                \n",element);
        getchar();
}
printf("queue is empty\n");
getchar();
}
```

Dvipusė eilė

Dvipusė eilė (deque) arba dekas yra tiesinis sąrašas, į kurį elementai įrašomi arba trinami viename ir kitame gale. Dekas realizuojamas naudojant dvikrypčius sąrašus arba vienkrypčius sąrašus su dvejomis rodyklėmis: pradžios ir pabaigos.

Deko veiksmai:

InsertFront (Item)

InsertRear (Item)

DeleteFront()

DeleteRear()

Dvipusės eilės realizacija

Dvipusės eilės pavyzdys, naudojant tiesinį dinaminį sąrašą.

Elementas:

```
struct node{
        int data;
        struct node *next;
        struct node *prev;
};

// pr – pradžios rodyklė
// g  – galo rodyklė
```

```
void InsertFront (struct node *e,
struct node *pr, struct node *g)
{
if (pr == NULL)
    pr = g = e;
else {
        e->next = pr;
        pr = e;
}
```

Dvipusės eilės realizacija

```
void RemoveRear (struct node *pr, struct node *g)
{
 struct  node *p = g;
 struct  node *d = pr;

while (d->next->next != NULL )
        d= d->next;

  d->next = NULL;
  g = d;
  free (p);
}
```

Pratybos

Nr.1 Turime masyvą, kurio elementai: { 2; 23; 15; 5; 9 }. Parašyti programą, kuri ištrintų elementą su reikšme 15, pridėtų į galą elementą su reikšme 17, pakeistų elemento 23 reikšmę į 44.

Nr.2 Turime tiesinį sąrašą, kurio elmentai: { 2; 23; 15; 5; 9 }. Parašyti programą, kuri ištrintų elementą su reikšme 15, pridėtų į galą elementą su reikšme 17, pakeistų elemento 23 reikšmę į 44.

Savarankiškas darbas: parašyti programą, kuri atliktų tas pačias operacijas, kai naudojamas stekas ir eilė.