Dünaamiline mälujaotus 2

REALLOC()

`sizeof` suurus baitides
`strlen` tähemärkide arv

sizeof() vs strlen()

```c
#include <stdio.h>
#include <string.h>

int main(void)
{
    char arr[10] = "Hello!";
    printf("%d\n", sizeof("Hello!"));
    printf("%d\n", strlen("Hello!"));
    printf("%d\n", sizeof(arr));
    printf("%d\n", strlen(arr));
    printf("%d\n", sizeof(char));
    printf("%d\n", sizeof(char *));
    return 0;
}
```

```
risto@risto-virtual-machine ~/Desktop $ ./sizes
7
6
10
6
1
8
risto@risto-virtual-machine ~/Desktop $
```

Massiivi suurust saab leida vaid samas skoobis, kus see deklareeriti. Funktsioonis on tegu juba viidaga!

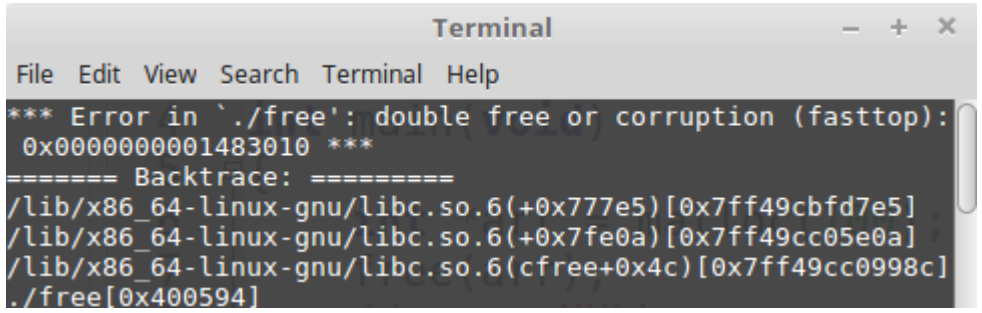
sizeof() vs strlen()

```c
int main(void){
    char arr[10] = "Hello!";
    printf("0: %d %d\n", sizeof(arr), strlen(arr));
    Print1(arr); Print2(arr); Print3(arr);
    return 0;
}
void Print1(char *arr){
    printf("1: %d %d\n", sizeof(arr), strlen(arr));
}
void Print2(char arr[]){
    printf("2: %d %d\n", sizeof(arr), strlen(arr));
}
void Print3(char arr[10]){
    printf("3: %d %d\n", sizeof(arr), strlen(arr));
}
```

```
risto@risto-virtual-machine ~/Desktop $ ./sizes2
0: 10 6
1: 8 6
2: 8 6
3: 8 6
risto@risto-virtual-machine ~/Desktop $
```

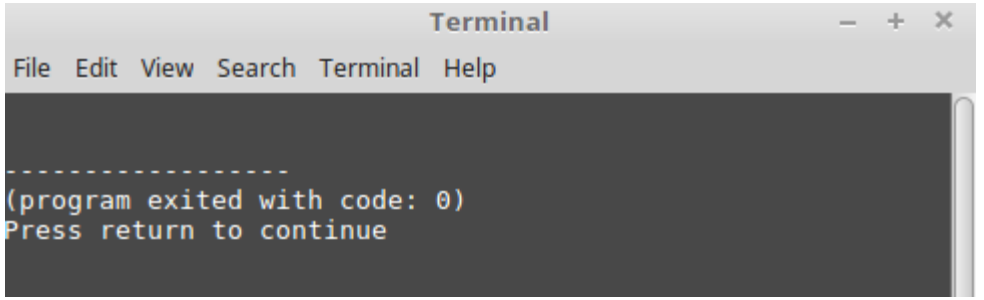
Näide: *defensive programminnig*

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int *arr = malloc(100);
    free(arr);
    free(arr);
    return 0;
}
```




```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int *arr = malloc(100);
    free(arr);
    arr = NULL;
    free(arr);
    return 0;
}
```

Ripakil viitade (*dangling pointer*) vältimine

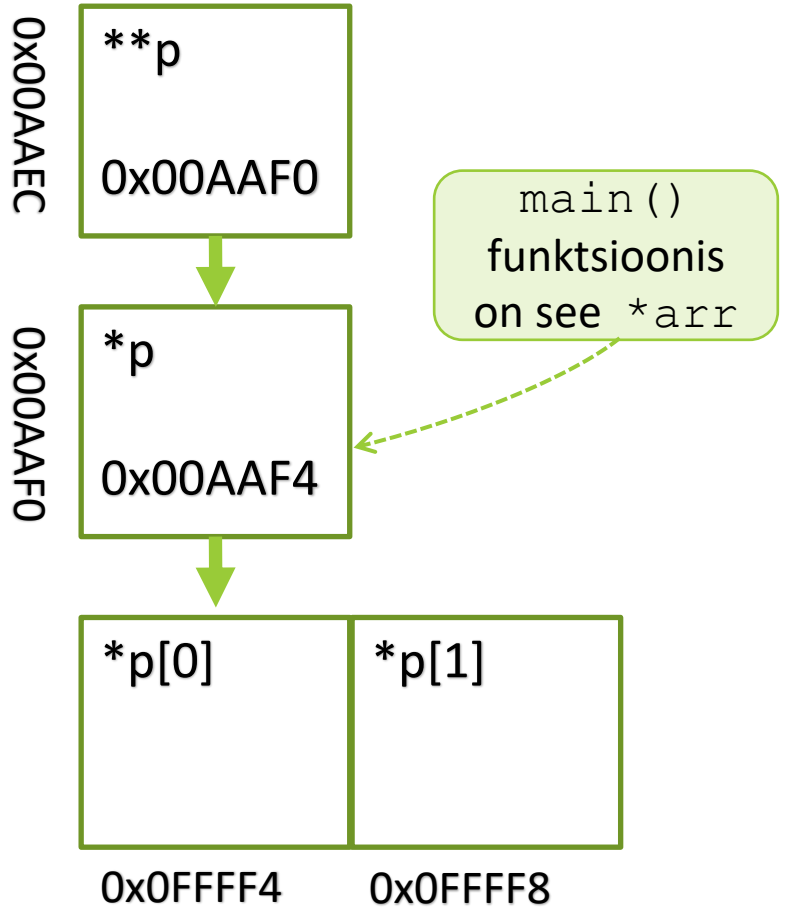


Näide 2: *defensive programming*

```c
#include <stdio.h>
#include <stdlib.h>

void SafeFree(void **p);

int main(void)
{
    int *arr = malloc(100);
    SafeFree((void *)&arr);
    return 0;
}

void SafeFree(void **p)
{
    if (p != NULL && *p != NULL)
    {
        free(*p);
        *p = NULL;
    }
}
```

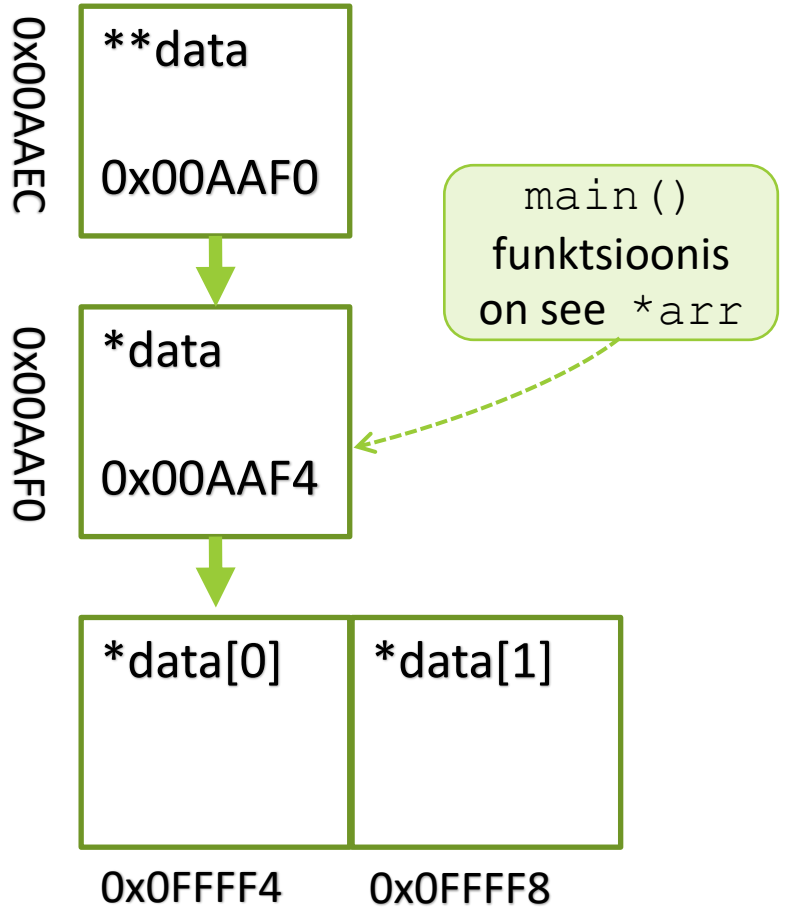
Abstraktsioon

Tehete järjekord!



Näide: topeltviida kasutamine

```c
#include <stdio.h>
#include <stdlib.h>

int AllocateData(int **data);

int main(void)
{
    int *vals;
    int length = AllocateData(&vals);
    // use data until needed
    printf("Allocated for %d\n", length);
    free(vals);
    return 0;
}

int AllocateData(int **data)
{
    int n;
    scanf("How many integers to allocate?\n");
    scanf("%d", &n);
    *data = (int *)malloc(sizeof(int) * n); // don't forget memory check!
    return n
}
```



Mälukasutuse optimeerimiseks kasutame muutuva pikkusega sõnede jaoks dünaamilist mälu

Näide: dünaamiline mälu sõnedele

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define BUFLEN 1024

int main(void)
{
    char *str;
    char buf[BUFLEN];
    scanf("%s", buf);

    str = (char *)malloc((strlen(buf) + 1) * sizeof(char));
    strcpy(str, buf);

    // use the data

    free(str);
    return 0;
}
```

Meenuta puhvri ületäitumise rünnakut!

`malloc()` **võib ebaõnnestuda**! Kontrolli enne kopeerimist!

Näide: strdup

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define BUFLEN 1024

int main(void)
{
    char *str;
    char buf[BUFLEN];
    scanf("%s", buf);

    str = strdup(buf);

    // use the data

    free(str);
    return 0;
}
```

`strdup()` sisemiselt

- Leiab sõne pikkuse
- Küsib dünaamiliselt mälu (pikkus + 1)
- Kopeerib sõne

Ebaõnnestumise korral tagastab `strdup()` `NULL`-viida.

`strdup()` **võib ebaõnnestuda**!
Kontrolli enne kasutamist!

Näide: Struktuuri liikmed dünaamiliselt

```
typedef struct {
    char *subject;
    int *grades;
    int gradeCnt;
} grade_item;

grade_item *sbj;
sbj = (grade_item *)malloc(sizeof(grade_item));
sbj->subject = strdup("Programming");
sbj->grades = (int *)malloc(sizeof(int) * n);
sbj->gradeCnt = n;

free(sbj->name);
free(sbj->grades);
free(sbj);
```

Struktuuri liikmeteks viidad

NB:Iga kord tuleb kontrollida kas mälu saadi!

Struktuuri liikmetele küsime eraldi mälu

Igale mälu küsimisele vastab vabastamine!

realloc()

`realloc()` lubab muuta dünaamiliselt saadud mälu mahtu

- Mälumahtu on võimalik suurendada
- Mälumahtu on võimalik vähendada
- Võimalik on teha ka esmane mälu küsimine, andes sisendiks `NULL`-viida (töötab nagu `malloc`)

Kui laiendamine **ebaõnnestub senises asukohas**

- tagastatakse viit uuele asukohale
- olemasolevad andmed kopeeritakse automaatselt uude asukohta
- vana viit muutub kehtetuks

Kui laiendamine **ebaõnnestub täielikult**

- tagastatakse `NULL`-viit
- vana viit jääb jätkuvalt kehtima koos seal olevate andmetega. Vana viida kaotamine tekitab mälulekke!

realloc()

```
void * realloc (void * ptr, size_t size);
```

Tagastus `(void *)` – viit saadud mälualale

Parameetrid

- `ptr` - `(void *)` – viit praegusele mälualale
- `size` - `(size_t)` – uus suurus baitides

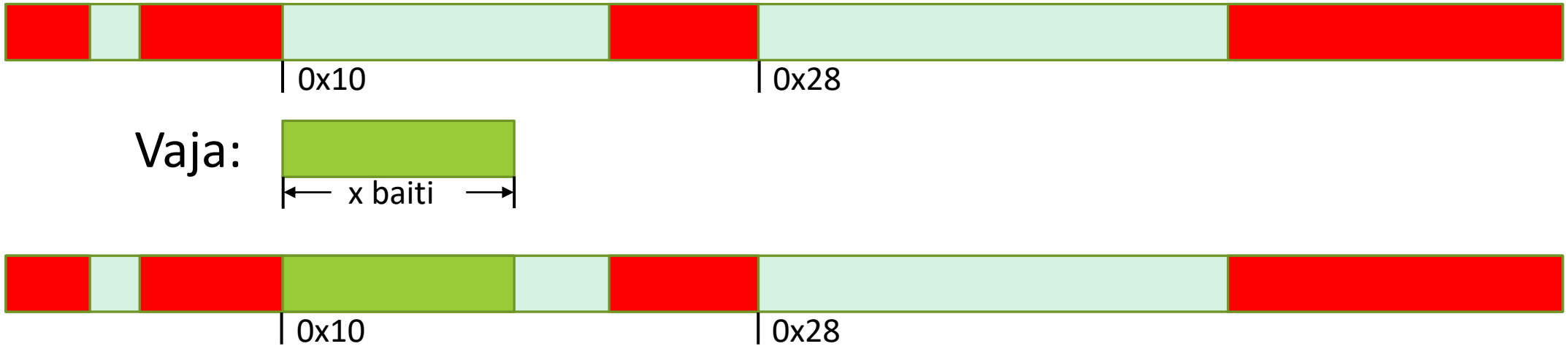
Tüübiteisendus on **valikuline** C keeles, **kohustuslik** C++ keeles

```
pTemp = (int *)realloc(pData, sizeof(int) * n)
```

Erinevad viidad mälulekke vältimiseks

Võimalikud tulemused (1/5)

`realloc` kasutamine esmaseks mälu küsimiseks



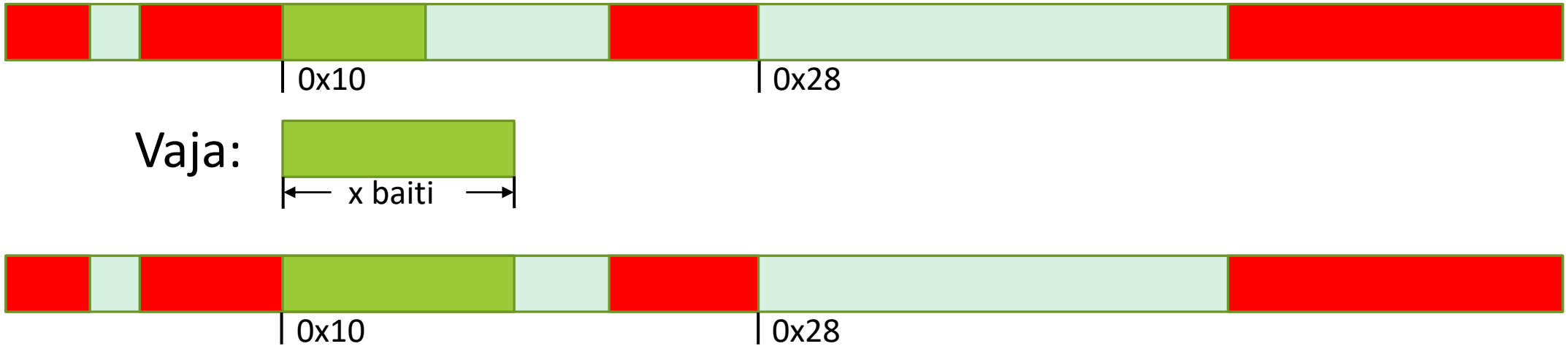

```
pTemp = realloc(pData, x)
```

Hõivatud rakenduse poolt
Blokeeritud teiste rakenduste ja süsteemi poolt
Saadaval olev mälu

```
pData:   NULL
pTemp:   0x10
```

Võimalikud tulemused (2/5)

Mäluala laiendatakse praeguses asukohas



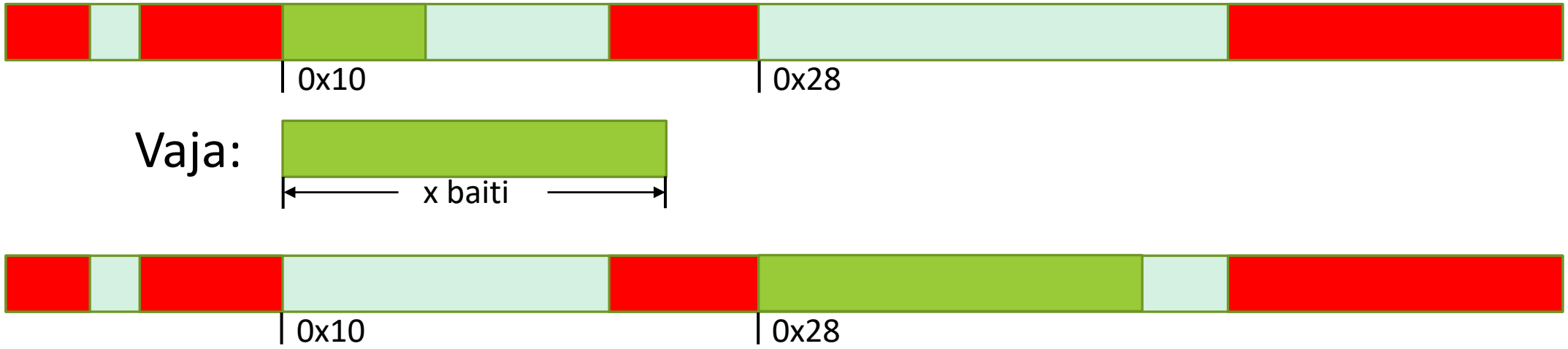

```
pTemp = realloc(pData, x)
```

Hõivatud rakenduse poolt
Blokeeritud teiste rakenduste ja süsteemi poolt
Saadaval olev mälu

```
pData:   0x10
pTemp:   0x10
```

Võimalikud tulemused (3/5)

Laiendamise käigus liigutatakse senised andmed uude kohta




```
pTemp = realloc(pData, x)
```

Hõivatud rakenduse poolt
Blokeeritud teiste rakenduste ja süsteemi poolt
Saadaval olev mälu

```
pData:   0x10
pTemp:   0x28
```

Näide: realloc muudab mälu asukohta

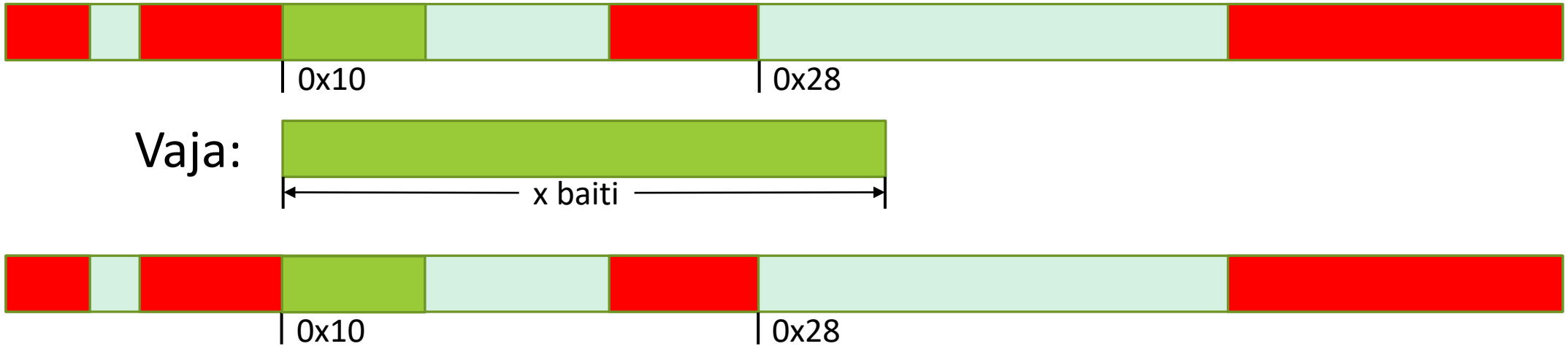
```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int *pData = (int *)malloc(sizeof(int) * 1000);
    printf("pData  = %p\n\n", pData);

    int *pTemp = (int *)realloc(pData, sizeof(int) * 50000);
    printf("pData  = %p\n", pData);
    printf("pTemp  = %p\n", pTemp);

                ...
```

```
pData  = 0x606ebf4ea2a0

pData  = 0x606ebf4ea2a0
pTemp  = 0x769bbdfb6010
```

- `pData` viit on kehtetu
- `pTemp` viitab andmetele
- `pTemp` tuleb vajadusel vabastada

Võimalikud tulemused (4/5)

Laiendamine ebaõnnestub, senised andmed säilivad




```
pTemp = realloc(pData, x)
```

```
pData:    0x10
pTemp:    NULL
```

- Hõivatud rakenduse poolt
- Blokeeritud teiste rakenduste ja süsteemi poolt
- Saadaval olev mälu

Näide: realloc võib ebaõnnestuda

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int *pData = (int *)malloc(sizeof(int) * 1000);
    printf("pData  = %p\n\n", pData);

    int *pTemp = (int *)realloc(pData, sizeof(int) * 20000000000);
    printf("pData  = %p\n", pData);
    printf("pTemp  = %p\n", pTemp);

                ...
```

```
pData  = 0x63ae603f32a0

pData  = 0x63ae603f32a0
pTemp  = (nil)
```

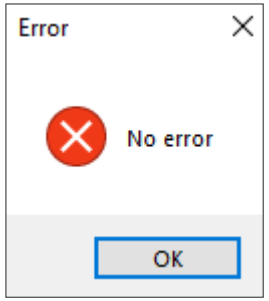
- `pData` viitab andmetele
- `pTemp` ei viita mitte millelegi
- `pData` tuleb vajadusel vabastada

0-baidine `realloc()`
Enne C23 – *implementation defined*
C23 – *undefined behavior*

Võimalikud tulemused (5/5)

Laiendame suurusele 0 baiti sooviga vabastada

Vaja: -




```
pTemp = realloc(pData, 0)
```

```
pData:    NULL
pTemp:    ?!?!?!
```

- Hõivatud rakenduse poolt
- Blokeeritud teiste rakenduste ja süsteemi poolt
- Saadaval olev mälu

Tundmatu kirjete arvu lugemine failist

Võimalus 1 : tuvasta kirjete arv enne lugemist

- Leia ridade arv (lugedes faili 2 korda ning loendades 👎 või salvestades ridade arvu faili algusse 👎 👍)
- Küsi mälu vastavalt kirjete arvule failis
- Loe kirjed mällu

Võimalus 2: Loe ühe kirje kaupa, laiendades jooksvalt mälu

- Lugemise ajal laienda mälumahtu (dünaamilist massiivi) `realloc()` funktsiooni abil
- Strateegiad: laiendus iga rea järel (n + 1), laiendame etteruttavalt (n * 2), …

Võimalus 3 : loe ühe kirje kaupa, salvesta ahelloendisse (*linked list*)

- Iga kirje jaoks küsitakse eraldi mälu lugemise ajal
- Kõik struktuurid seotakse omavahel kasutades viitasid
- Võimalusi erinevaid andmestruktuure luua on enam

Oluline mahukate ja keeruliste projektide puhul

Mõttekohad realloc() kasutamisel

Dünaamilise mälu kutsed on aeglased
- Vajadusel tuleb optimeerida `realloc()` kutseid
- Erinevad lähenemised vastavalt rakenduse iseloomust

Laienda mälu kordajaga, mitte ühe kaupa
- Kordaja võib olla lineaarne, eksponentsiaalne, logaritmiline
- Kordaja iseloom võib programmi käigus muutuda

Ära vähenda mäluala suurust ühe kaupa
- Nt Vabasta kui 20% kasutamata, 50% kasutamata, 75% kasutamata, …

Optimeeri esimest mälu küsimist
- Ära alusta ühest liikmest. Küsi alguses 10, 100, 1000, …. liikme jagu mälu
- Kasuta statistikat, isiklikku kogemust, rakenduse otstarvet, … esmase koguse leidmiseks

Erijuhtudel: loo isiklik dünaamilise mälu haldur

Kasuta varasemalt õpitud faili lugemise funktsiooni struktuuri. Kohanda tsükkel ja deklaratsioonid vastavaks.

Pseudokood (n + 1) strateegiale

Vajalik: 1 loendur, 2 struktuuriviita ja ajutised muutujad (puhvrid). Algväärtused olulised!

- `loenduri` väärtuseks `0`
- Struktuuriviitade `pTemp` ja `pData` väärtusteks `NULL`

Loe tsüklis kõik kirje väljad puhvri(te)sse. Kui kirje loeti edukalt, siis:

- Laienda oma mälu (`loendur + 1`) struktuuri jagu
- Kui laiendamine ebaõnnestus:
  - Sulge fail
  - Otsusta mida edasi teed (nt töötad sellega mis said või katkestad)
- Määra mõlemad viidad näitamaks andmete asukohale (`pData = pTemp`)
- Kopeeri kirje puhvrist struktuurimassiivi
- Suurenda `loendurit`

Need read **ei tohi** käivituda laiendamise ebaõnnestumisel!

Eelista (n * 2) strateegiat – oluliselt kiirem

Pseudokood (2 * n) strateegiale

Loo 2 loendurit (`currentRecordCount` ja `totalRecordsAllocated`), 2 struktuuriviita ja ajutised puhvrid. Algväärtusta.

Loe tsüklis kõik kirje väljad puhvrisse. Kui kirje loeti edukalt, siis:

- Kontrolli, kas mälu on juurde vaja (`currentRecordCount >= totalRecordsAllocated`)
  - Laienda oma mälu (`totalRecordsAllocated * 2`) struktuuri jagu
  - Kui laiendamine ebaõnnestub
    - Otsusta mida edasi teed
  - Uuenda `totalRecordsAllocated` väärtust
  - Määra mõlemad viidad näitamaks andmete asukohale (`pData = pTemp`)
- Kopeeri kirje puhvrist struktuurimassiivi
- Suurenda `currentRecordCount` loendurit

Need read ei tohi käivituda laiendamise ebaõnnestumisel!

Kasuta realloc() funktsiooni vajadusel mälu vähendamiseks (tagasta liigne mälu, valikuline*)