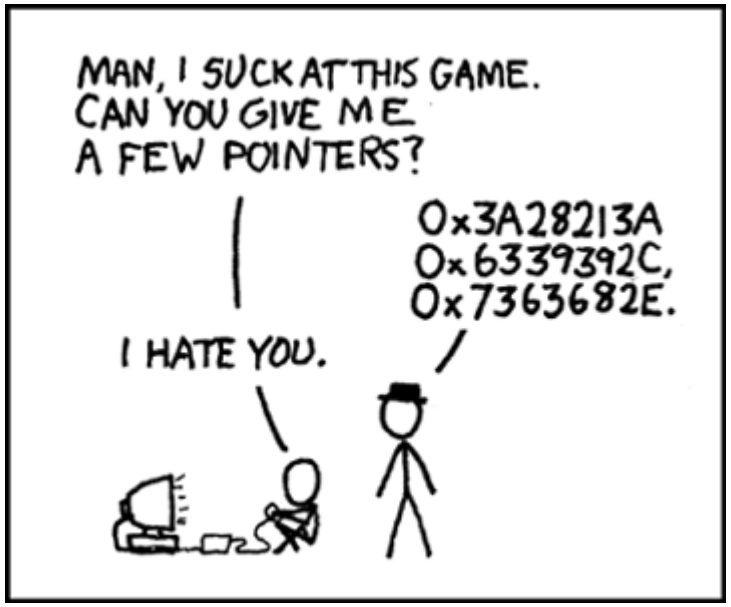


https://xkcd.com/138/

Viidad (*pointers*)

VIIT MUUTUJALE

VIIDAD JA FUNKTSIOONID

Mis on viit?

Viit on olemuselt **aadress**. See **viitab asukohale arvuti mälus**

- Aadress võib olla füüsiline või virtuaalne

Viitade ehk mäluaadresside salvestamiseks kasutame **viitmuutujaid**

Tegu on **täisarvulise väärtusega**, mis viitab positsioonile / asukohale mälus

Viitmuutujal on **andmetüüp**, mis vastab viidatavatele andmetele

```
int *pList;
char *pKey;
void *pMember;
```

Võid kujutada seda ette nt sisukorra või majajuhina

Viit ja viidatavad andmed on eraldiseisvad – üks ei taga ega eelda teist!

Aadresse kuvatakse tavaliselt 16ndsüsteemi arvudena (0x0, 0x0fff1a9b)

Kasutuse valdkonnad

Riistvara ligipääs (operatsioonisüsteemid, ajurid)

Abstraktsioon

Dünaamiline mälujaotus

Otsene andmete poole pöördumine

Andmestruktuurid (puud, ahelloendid, järjekorrad, …)

Funktsiooniviidad

Memory mapped IO, failid, …

jne

NULL-viit

Eritähendusega viit - ei viita mitte millelegi

Kasutamine põhjustab määramata käitumist (nt kokkujooks)

Kasutatakse sageli puuduva objekti või ebaõnnestunud operatsiooni tähistamiseks

- Ahelloendi lõpp (*linked list*)
- Mälu küsimise ebaõnnestumine (*memory allocation*)
- Faili avamise ebaõnnestumine
- Alamsõne otsimisel tulemuse puudumine (`strstr()` teegist `string.h`)
- Muutujate antakse `NULL` väärtus vältimaks tahtmatut pöördumist vabastatud ressurssidele

`NULL` on viit, `0` on täisarv. `NULL` viit on enamasti väärtusega 0 (eksisteerib erandeid)

Viitmuutuja deklareerimine

Viitmuutuja nimi algab enamasti väikse p tähega (*p – pointer)*
- `p`, `pData`, `pGreatestMember`, `pStartofLastName`, jne.

Deklareerime viitmuutuja täisarvule
- `int *p;`

Deklareerime viitmuutuja reaalarvule
- `float *p;`

Viitmuutujad ja tavamuutujad deklareeri **eraldi ridadel**!

Deklareerime kaks viitmuutujat ühel real
- `int *p1, *p2;`

Deklareerime `void`-tüüpi viida (tüüp määramata, kasutamiseks vaja tüübiteisendust, kasutatakse abstraktsioonideks)

```
void *p;
```

Viida operaatorid

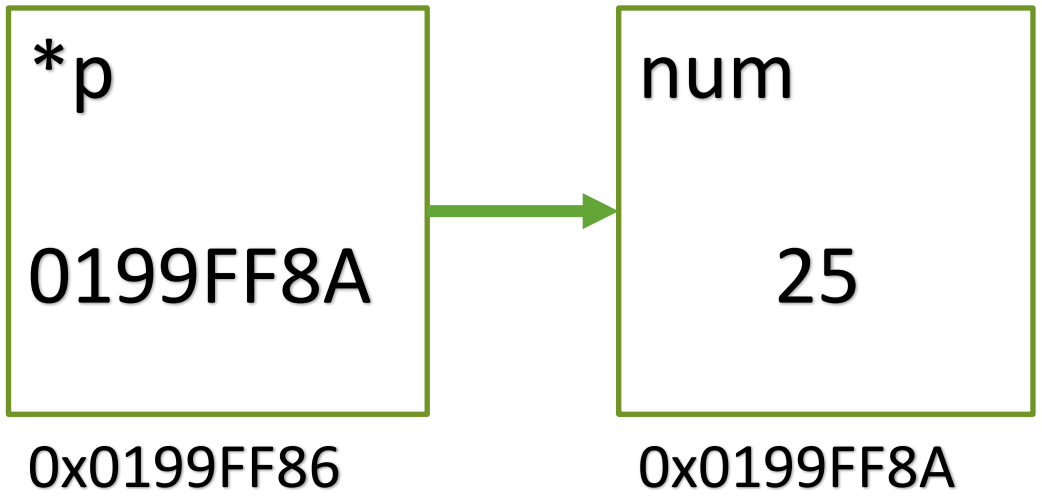
- `&` Aadressi operaator (*Address-of operator*)
- `*` Viitamise operaator (*Dereference operator*)

Enamik operaatoreid töötavad viitadega, kuid käituvad viida puhul veidi teisiti. Täpsemalt viidaaritmeetikast hiljem.

Operatsioonid viitadega

```
int *p;                 // Deklareerime viitmuutuja
&var;                   // Võtame muutuja mäluaadressi
*p;                     // Pöördume viidatava väärtuse poole
p = &var;               // Omistame muutuja var aadressi viitmuutujale p
*p = 55;                // Omistame väärtuse muutuja p poolt viidatavale aadressile
printf("%p", p);        // Väljastame muutujas salvestatud mäluaadressi
printf("%d", *p);       // Väljastame muutuja poolt viidatava väärtuse
```

Viitadest visuaalselt




```c
#include <stdio.h>

int main(void)
{
    int num = 25;
    int *p;
    p = &num;
    printf("%d", num);
    printf("%d", *p);
    return 0;
}
```

Näide 1: viit ja aadress

```c
#include <stdio.h>
int main(void)
{
    float pi = 3.14;
    float *p = &pi;
    printf("The pointer holds an address: %p\n", p);
    printf("pi variable is located at:\t\t %p\n", &pi);
    printf("Dereferencing p we can access the value behind it\t %.2f\n", *p);
    return 0;
}
```

```
The pointer holds an address:                   0x7ffdab04ed50
pi variable is located at:                      0x7ffdab04ed50
Dereferencing p we can access the value behind it       3.14
```

64-bit programm
64-bit Linux

```
The pointer holds an address:           0028FF00
pi variable is located at:              0028FF00
Dereferencing p we can access the value behind it       3.14
```

32-bit programm
64-bit Windows

Näide 2: Viit ja scanf()

```c
#include <stdio.h>

int main(void)
{
    int num;
    int *p;                 // Declare a pointer variable
    p = &num;               // Assign the address of num to p
    scanf("%d", p);         // Why didn't I use & here?
    printf("%d\n", *p);     // Printing by dereferencing the pointer p
    *p = 55;                // Assigning a value using the pointer
    printf("%d\n", num);    // Which number will print?
    return 0;
}
```

Viida edastamine funktsiooni

Meenuta, mis edastati funktsiooni koopiana, mis viitena originaalile!

- Muutuja puhul edastatakse **väärtuse koopia** *(by value)*
- Massiivi puhul edastatakse massiivi esimese elemendi mäluaadress **(viida väärtuse koopia)**

Kasutades aadressi operaatorit (&) saame funktsiooni edastada muutuja mäluaadressi

- Teades muutuja mäluaadressi saame andmeid muuta kõikjal
- Võimaldab teises funktsioonis deklareeritud muutuja väärtuse muutmist!
- Võimaldab ka mitme väärtuse muutmist funktsiooni poolt

Muutuja eluiga aadressi teadmine ei mõjuta. Kui muutuja on „hävitatud“, ei tohi selle poole enam pöörduda ka aadressi abil

Märka, et `main()`-is enam viitmuutujat ei looda!

Näide 3: funktsioonid ja viidad

```c
#include <stdio.h>

void SetVal(int *val);

int main(void)
{
    int num = 0;
    SetVal(&num);        // Passing the address (location) of the variable num
    printf("%d\n", num);
    return 0;
}

void SetVal(int *val)    // This parameter is now a pointer, we are passing an address to it
{
    *val = 55;           // Assigning a value using dereferencing
}
```

Viidad (*pointers*)

TOPELTVIIDAD

VIIDA ARITMEETIKA

VIIDAD JA MASSIIVID

Topeltviidad

Viida deklareerimine

```
int *p;
```

Topeltviida deklareerimine

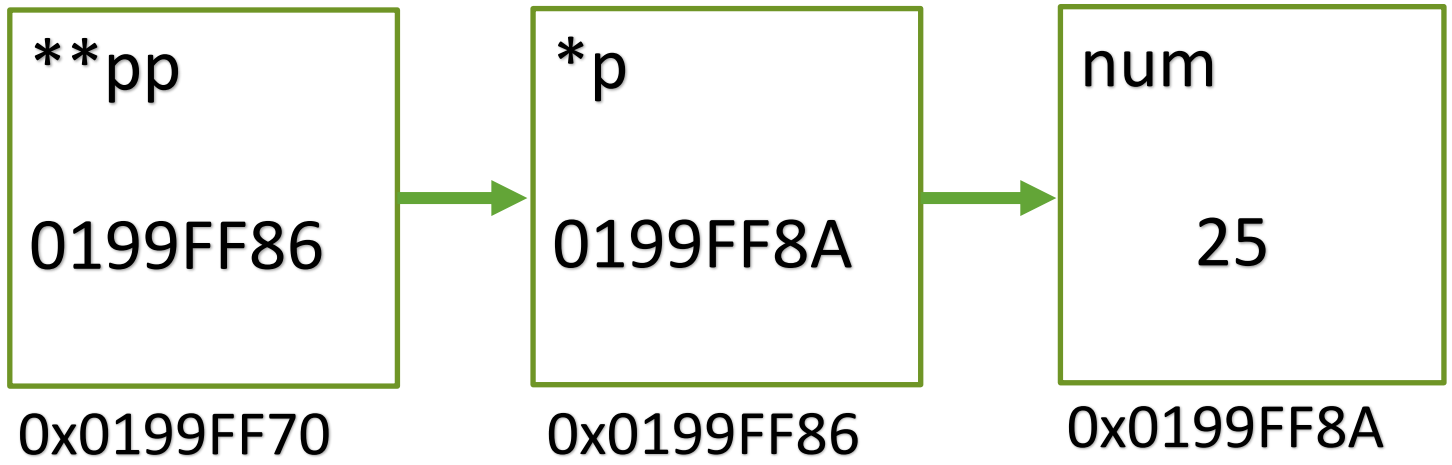
```
int **pp;
```

Topeltviit on viit viidale

- Eesmärk on hoiustada mõne teise viitmuutuja aadressi
- Topeltviit hoiustab viitmuutuja aadressi, mis viitab andmetele
- Kasutades viitamise operaatorit ühe korra (*) jõuame viidani, kasutades 2 korda (**) jõuame andmeteni
- Kasutatakse näiteks funktsiooni edastatud viida väärtuse muutmiseks funktsiooni sees
- Kasutatakse sageli funktsiooni edastatud viitmuutuja väärtuse muutmiseks

Eksisteerivad ka kolme, nelja, … kordsed viidad – **võimalusel väldi!**

Topeltviit visuaalselt




```
int num = 25;
int *p = &num;
int **pp = &p;
```

Näide 4: Topeltviit

```c
#include <stdio.h>

void FindMax(int *val1, int *val2, int **ppMax);

int main(void)
{
    int val1 = 10;
    int val2 = 6;
    int *pMaxVal;

    FindMax(&val1, &val2, &pMaxVal);
    printf("Max value is %d\n", *pMaxVal);
    return 0;
}

void FindMax(int *val1, int *val2, int **ppMax)
{
    *ppMax = (*val1 < *val2) ? val2: val1;
}
```

Kolmikoperaator lühikeste if/else lausete jaoks:

```
condition ? stmt_true: stmt_false;
```

Viidad, massiivid ja viida-aritmeetika

Massiivi saab käsitleda kui viita esimese massiivi liikmele

Massiivi liikmed on mälus järjestikused

Massiivi saab indekseerida kasutades viida-aritmeetikat. Viida andmetüüp äärmiselt oluline!

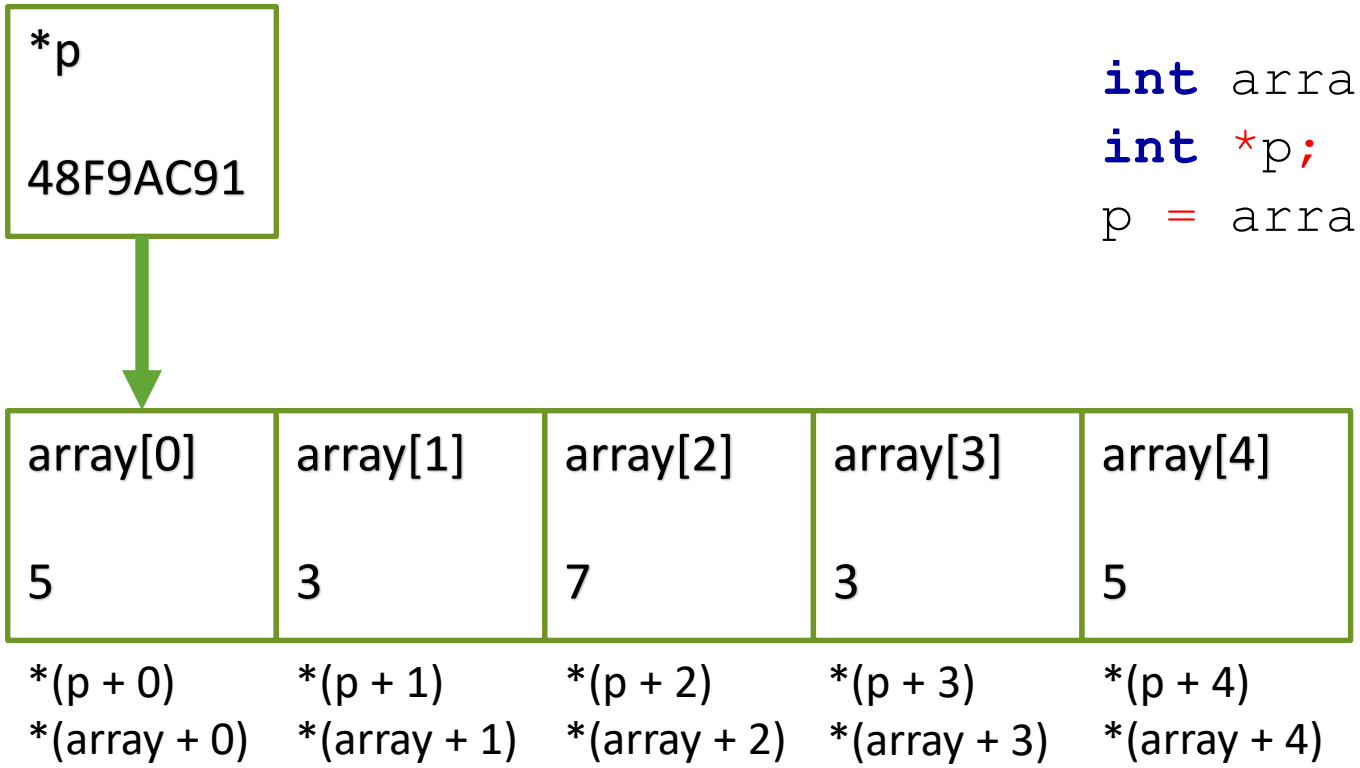
Massiivi indekseerimine taandub **alati** viida-aritmeetikale:

- `massiivi_esimese_liikme_aadress + indeks`

NB! Tehete järjekord on oluline!

```
p = &array[3] // assigns the location of the 4th member of the array to p
p = array     // assigns the start of the array to p
*(p + i)      // access (dereference) the p + i member of the array
*(array + i)  // same as previous
p++           // increment the address by 1 element (type dependent)
```

Viidad ja massiivid visuaalselt




```
int array[] = {5, 3, 7, 3, 5};
int *p;
p = array;
```

`array[ind]`
taandub kujule
`*(array + ind)`

Näide 5: Viidad ja massiivid

```c
#include <stdio.h>

int main(void)
{
    int array[] = {5, 3, 7, 3, 5};
    int *p = array;
    int i;
    printf("%p\n", p);
    printf("%p\n\n", array);
    for (i = 0; i < 5; i++)
        printf("%p, %d\n", (p + i), *(p + i));
    return 0;
}
```

```
0x7fff43605be0
0x7fff43605be0

0x7fff43605be0, 5
0x7fff43605be4, 3
0x7fff43605be8, 7
0x7fff43605bec, 3
0x7fff43605bf0, 5
```

(array + i)
töötab samuti

Viida-aritmeetika sõltub andmetüübist

```c
#include <stdio.h>

int main(void)
{
    int i;
    int ints[] = {1, 2, 3, 4, 5};
    char chars[] = {"abcde"};
    short shorts[] = {1, 2, 3, 4, 5};
    double doubles[] = {1.1, 1.2, 1.3, 1.4, 1.5};

    int *pInt = ints;
    char *pChar = chars;
    short *pShort = shorts;
    double *pDbl = doubles;

    for (i = 0; i < 5; i++, pInt++, pChar++, pShort++, pDbl++)
    {
        printf("%c @ %p \t", *pChar, (void *)pChar);
        printf("%hd @ %p \t", *pShort, (void *)pShort);
        printf("%d @ %p \t", *pInt, (void *)pInt);
        printf("%.1lf @ %p\n", *pDdbl, (void *)pDdbl);
    }
    return 0;
}
```

```
a @ 0028FF12    1 @ 0028FF08    1 @ 0028FF18    1.1 @ 0028FEE0
b @ 0028FF13    2 @ 0028FF0A    2 @ 0028FF1C    1.2 @ 0028FEE8
c @ 0028FF14    3 @ 0028FF0C    3 @ 0028FF20    1.3 @ 0028FEF0
d @ 0028FF15    4 @ 0028FF0E    4 @ 0028FF24    1.4 @ 0028FEF8
e @ 0028FF16    5 @ 0028FF10    5 @ 0028FF28    1.5 @ 0028FF00
```

Isegi kui saab, ei tähenda, et peaks

```c
#include <stdio.h>

int main(void)
{
    int arr[] = {11, 12, 13, 14, 15};
    int ind = 2;

    printf("a) %d\n", arr[ind]);
    printf("b) %d\n", *(arr + ind));
    printf("c) %d\n", *(ind + arr));

    printf("d) %d\n", ind[arr]);
    printf("e) %d\n", 0[ind + arr]);

    return 0;
}
```

Massiivi indekseerimine `arr[i]` taandub alati viida-aritmeetikale, st `*(arr + i)`

Kuniks matemaatika on korrektne….

… ja sa ei salli oma kolleege …

Või osaled *The International Obfuscated C Code Contest-il*