**Viidatehnika**

Massiivi korral on tavaline, et elementide poole pöördumisel ei kasutata mitte indekseid, vaid viitmuutujat. C/C++-s realiseeritud aadressaritmeetika teeb selle mitte ainult võimalikuks, vaid ka mugavaks. Viitmuutuja massiivi elementidele deklareeritakse sama tüüpi, kui on massiiv. Järgnevalt deklareeritud viitmuutuja ‘raha’ võib (kuid ei pruugi!) hakata viitama massiivi ‘euro’ elementidele.

int euro[8]={1,2,5,10,50,100,200,500};

int \*raha;

**Viida väärtus**

Viida väärtustamine toimub tavaliselt omistuslausega. Kuna viitamisel massiivile on viida esimene väärtus tavaliselt massiivi esimese elemendi (indeks 0) aadress, siis programmi kirjutamise mugavuse nimel on omistusel massiivi nimi samaväärne tema esimese elemendi aadressiga. Järgmise kahe omistuslause tulemus on sama:

raha=&euro[0];

raha=euro;

Massiivi teistele elementidele viitamiseks kasutatakse aadresstehet ja/või indeksit, nii viitab viitmuutuja ‘raha’ peale järgmise omistuslause täitmist massiivi ‘euro’ seitsmendale elemendile:

raha=&euro[6];

Aadressaritmeetika lubab teha viitmuutujatega aritmeetikatehteid. NB! Liitmisel või lahutatamisel on viitmuutujate jaoks ühikuks alati andmetüübi mäluvälja pikkus! Viitmuutujate kasutamine massiivis olevate andmete poole pöördumisel muutub programmeerijale arusaadavaks ja loomulikuks alates sellest hetkest, kui mõistetakse tava- ja viitmuutuja tehete erinevust. Viitmuutuja suurendamine ühe võrra tähendab viitamist järgmisele massiivi elemendile. Viidale kahe liitmine tähendab viita ülejärgmisele elemendile, mida kinnitab ka koodilõik:

raha=&euro[3];

printf("Aadressil %p on %d-eurone\n",

raha, \*raha);

raha+=2; // liidetakse kahe mäluvälja pikkus,

 // mitte arv '2'!

printf("Aadressil %p on %d-eurone\n",

raha, \*raha);

Viida vormingutüübina printf-funktsioonis on tavaliselt ‘%p’ – viida väärtus 16ndsüsteemis. Ülaltoodud kaks printf-funktsiooni väljastavad

Aadressil 0022FF44 on 10-eurone

Aadressil 0022FF4C on 100-eurone

Viitmuutuja õigeks kasutamiseks on tähtsad veel kaks momenti. Esiteks vahe tegemine viitmuutuja väärtuse (milleks on aadress) ja viidatava väärtuse vahel. Viimase poole pöördumisel on viitmuutuja nime ette vaja kirjutada tärn (\*). Teiseks tuleb jälgida tehete prioriteete ehk nende täitmisjärjekorda. Tehete puhul viitmuutujatega on tähtis see, kas tehe tehakse kõigepealt aadressiga või siis viidatava väärtusega või vastupidi. Alati on parem lisada avaldises kindluse mõttes pigem enam sulge, kui jätta vajalikud avaldisest välja. C/C++ enamasti ei hoiata võimalikust veast, vaid täidab laused nii, nagu kirja pandud. Kui eelnev suurendustehe oleks selline:

\*raha+=2;

tähendab see viitmuutuja ‘raha’ sisaldaval aadressil oleva väärtuse suurendamist kahe võrra.

#include <stdio.h>

 #include <stdlib.h> //rand(), srand()

 #include <time.h> //time()

#define ARR\_LEN 15

**void arr\_in(int\*, int);**

**void sort\_bubble(int\*, int);**

**void arr\_out(int\*, int);**

**int main(void**){

 int arr\_len = ARR\_LEN; //loeme jada pikkuse

 int arr\_mem[arr\_len]; // anname jada jaoks mälu

 int \*arr = arr\_mem; //viit jada esimese elemendi mälu aadressile

 arr\_in(arr, arr\_len);

 printf("V2ljasta sorteerimata jada:\n");

 arr\_out(arr, arr\_len);

 sort\_bubble(arr, arr\_len);

 printf("V2ljasta sorteeritud jada:\n");

 arr\_out(arr, arr\_len); return 0; }

**void arr\_in(int \*array, int array\_length**){

 int i;

 srand(time(NULL));

 for(i=0;i<array\_length;i++){\*(array + i) = rand() % 100; }}

**void arr\_out(int \*array, int array\_length**){

 int i;

 for(i=0;i<array\_length;i++){ printf("%d ", \*(array + i)); } printf("\n");}

**void sort\_bubble(int \*array, int array\_length**){

 int i, j, temp;

 for(i=0;i<array\_length-1;i++){

 for(j=0;j<array\_length-i-1;j++){

 if(\*(array + j) < \*(array + (j+1))){ temp = \*(array + j);

 \*(array + j) = \*(array + (j+1)); \*(array + (j+1)) = temp;

 } } } }