6. Masīvi, norādes un references

Nodaļas saturs:

6.1. Masīvs kā funkcijas parametrs

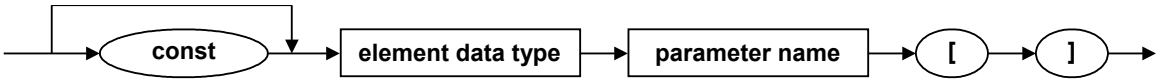
Masīvu nevar nodot caur funkcijas parametru kā vērtību (tādējādi veicot tā satura dublēšanu), bet tā nodošana caur parametru iespējama tikai tādā veidā, kā tas notiek caur parametriem-referencēm. Tam varētu būt 2 iemesli: pirmkārt, masīvs ir relatīvi apjomīga datu struktūra un nebūtu sapratīgi dublēt tik lielu informācijas apjomu, otrkārt, valodā to nosaka valodas C++ īpatnības – masīva mainīgais reprezentē nevis masīvu kopumā, bet gan tikai norāda uz tā sākumu.

Lai definētu parametru, caur kuru iespējams nodot masīvu, iespējami divi dažādi veidi, kas ir analoģiski:

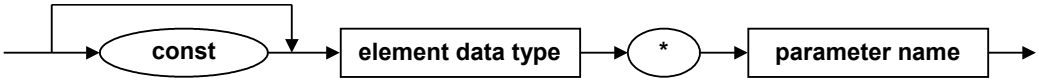
- Masīvu specifisks, izmantojot masīvu notāciju – kvadrātiekavas [] (sintakse 6.1),
- Universāls, izmantojot norāžu notāciju – zvaigznīti * (sintakse 6.2).

Tāpat kā parastu parametru deklarēšanā, arī šajā gadījumā tā līdzinās mainīgo deklarēšanai. (Ievērojiet, ka deklarēšanas gadījumā abu sintakses variantu lietošanai ir atšķirības – tie nosaka attiecīgi statisku vai dinamisku masīvu.)

Sintakse 6.1. ***array parameter declaration 1*** **(masīva parametra deklarēšana)**



Sintakse 6.2. ***array parameter declaration 2*** **(masīva parametra deklarēšana)**



Pirmkoda piemērs 6.1 parāda masīva nodošanu caur parametru. Programma demonstrē divas funkcijas, kam kā parametrs tiek padots masīvs: funkcija *printString()* izdrukā simbolu virkni uz ekrāna (*const* garantē, ka funkcija simbolu virkni nemainīs), bet funkcija *changeString()* nomaina simbolu virknē vienu simbolu. Funkcijā *printString()* masīva parametrs definēts, izmantojot masīva notāciju, bet funkcijā *changeString()* – norāžu notāciju. Pirmkoda piemērā 6.1 parādītā programma izdrukā divas simbolu virknes pirms un pēc viena simbola nomaiņas. Programma parāda, ka masīva nodošanai funkcijai neatšķiras atkarībā no tā, vai masīvs ir statisks vai dinamisks.

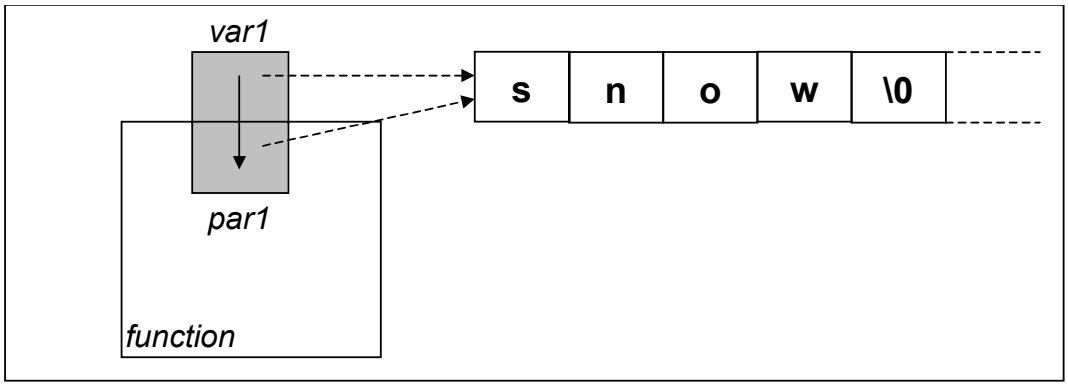
Var teikt, ka funkcija netiek nodots pats masīvs, bet gan tikai norāde uz masīvu. Arī pats masīva mainīgais pēc būtības ir norāde. Viena no masīva (vai masīva parametra) definēšanas notācijām ietver zvaigznīti, kas nozīmē norādi. **Norāde** (*pointer*) ir ļoti svarīgs jēdziens valodā C++. Tas pēc būtības ir ļoti līdzīgs jēdzienam **reference** (parametri-references tika apskatīti nodaļā par funkcijām), un abi jēdzieni smalkāk tiks apskatīti tālāk šajā nodaļā.

Pirmkods 6.1. Masīva nodošana caur funkcijas parametru (*ptr1arrpar.cpp*)

```
01 #include <iostream>
02 using namespace std;
03
04 int printString (const char s[])
05 {
06     cout << s << endl;
07 };
08
09 void changeString (char *s, int i, char c)
10 {
11     s[i] = c;
12 };
13
14 int main ()
15 {
16     char name[10];
17     strcpy (name, "snow");
18     printString (name);
19     changeString (name, 1, 'l');
20     printString (name);
21     char *name2 = new char[10];
22     strcpy (name2, "pig");
23     printString (name2);
24     changeString (name2, 2, 'n');
25     printString (name2);
26     delete[] name2;
27     return 0;
28 }
```

Programmas darbības piemērs:

```
snow
slow
pig
pin
```



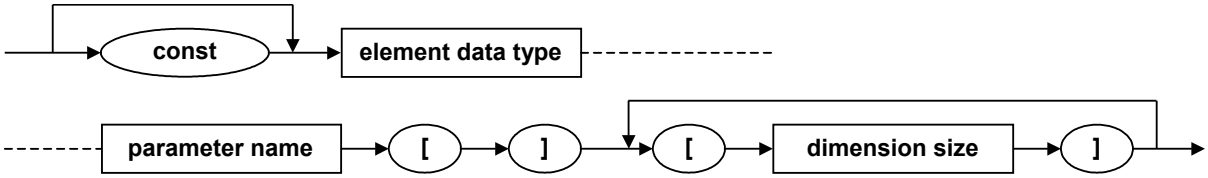
Attēls 6.1. Masīva nodošana caur parametru (tajā attēlotā funkcija atbilst funkcijai printString() pirmkoda piemērā 6.1)

6.2. Statisks vairākdimensiju masīvs kā funkcijas parametrs

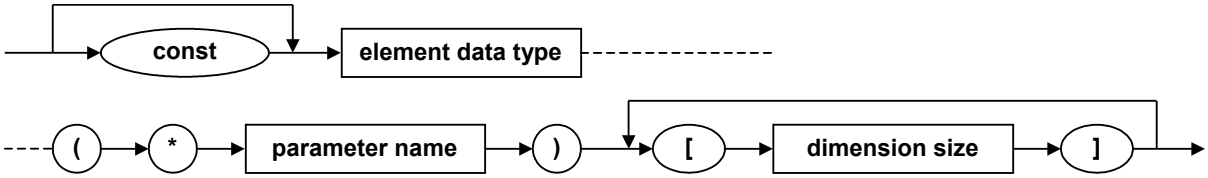
Viendimensijas masīva un vairākdimensiju masīvu padošana caur parametru sintaktiski neatšķiras, jo masīva mainīgā būtība ir vienāda abos gadījumos – norāde uz masīva sākumu.

Līdzība ar viendimensijas (statisko) masīvu nodošanai caur parametru statiskam vairākdimensiju masīvam ir tāda, ka pirmajai dimensijai izmērs nav jāuzrāda, tomēr visām pārējām ir (sintakse 6.3 un 6.4). Tapāt kā viendimensijas masīvu gadījumā, ir pieejamas divas notācijas – masīva un norāžu.

Sintakse 6.3. ***static multidimensional array parameter declaration 1*** **(statiska vairākdimensiju masīva parametra deklarēšana)**



Sintakse 6.4. ***static multidimensional array parameter declaration 2*** **(statiska vairākdimensiju masīva parametra deklarēšana)**



Norāžu notācijā (sintakse 6.4) ir jālieto papildus iekavas.

Pirmkoda piemērā 6.2 parādītā programma demonstrē divas funkcijas, kas izdrukā divu dimensiju masīvu, kurā tiek glabātas simbolu virknes.

Pirmkods 6.2. Statiska vairākdimensiju masīva nodošana caur funkcijas parametru (*ptr2marrpar.cpp*)

```
01 #include <iostream>
02 using namespace std;
03
04 int printStrings1 (const char s[][4], int count)
05 {
06     for (int i=0; i<count; i++) cout << s[i] << ' ';
07     cout << endl;
08 };
09
10 int printStrings2 (const char (*s)[4], int count)
11 {
12     for (int i=0; i<count; i++) cout << s[i] << ' ';
13     cout << endl;
14 };
15
16 int main ()
17 {
18     char arr[2][4] = {{'t', 'h', 'e', '\0'},
19                       {'p', 'i', 'g', '\0'}};
20     printStrings1 (arr, 2);
21     printStrings2 (arr, 2);
22     return 0;
23 }
```

Programmas darbības piemērs:

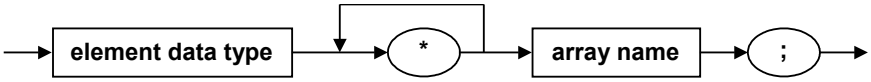
```
the pig
the pig
```

6.3. Dinamisks vairākdimensiju masīvs un tā nodošana caur parametru

Dinamiska masīva veidošana vairākdimensiju gadījumā ir sarežģītāka nekā viendimensijas masīva gadījumā, jo atmiņas izdalīšana jāveic katrai dimensijai atsevišķi.

Dinamisku vairākdimensiju masīvu deklarējot, jālieto tik daudz zvaigznīšu, cik masīvam ir dimensiju (sintakse 6.5).

Sintakse 6.5. ***dynamic multidimensional array declaration*** **(dinamiska vairākdimensiju masīva deklarēšana)**



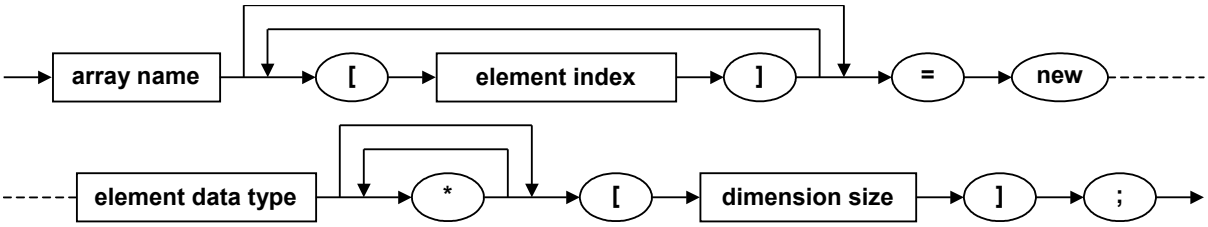
Dinamisku vairākdimensiju masīvu veido pa dimensijām, un tā izmantošana notiek šādā secībā:

- atmiņas izdalīšana dimensijām, secībā no rupjākās uz smalkāko,
- dinamiska masīva izmantošana, izmantojot parasto masīvu sintaksi,
- atmiņas atbrīvošana pa dimensijām, sākot no smalkākās un beidzot ar rupjāko.

Vairākdimensiju dinamiska masīva vienas dimensijas veidošanas sintakse aprakstīta diagrammā 6.6.

Deklarējot mainīgo, izmantojot zvaigznītes (norāžu) notāciju, jāatceras, ka zvaigznītes pieder pie datu tipa, un nosaucot vienu zvaigznīti, var izmantot vai nu vārdu masīvs vai norāde. Piemēram, divu dimensiju *int* masīva tips ir *int***, t.i. masīvs no masīviem no *int*, jeb norāde uz norādi no *int*. Katras nākošās dimensijas elementi ir ar tipu, kurā ir par vienu zvaigznīti mazāk, piemēram, masīvs ar tipu *int*** pirmajā līmenī sastāv no elementiem ar tipu *int** (salīdzināt ar attēlu 6.2).

Sintakse 6.6. ***dynamic array dimension creation*** **(dinamiska masīva vienas dimensijas izveidošana)**



Dinamiska masīva parametrus deklarē tāpat kā dinamiska masīva mainīgo – lietojot tik daudz norāžu (zvaigznīšu), cik ir dimensiju.

Pirmkoda piemērs 6.3 parāda dinamiska divu dimensiju izveidošanu, izmantošanu, kā arī nodošanu funkcijai caur parametru un iznīcināšanu. To ilustrē attēls 6.2.

Pirmkods 6.3. Dinamiska vairākdimensiju masīva izveidošana un nodošana caur funkcijas parametru (*ptr3darrpar.cpp*)

```
01 #include <iostream>
02 using namespace std;
```

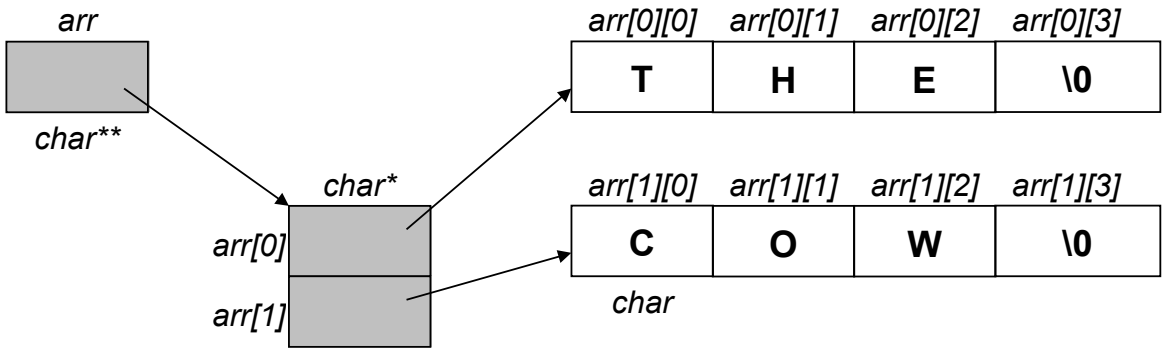
```
03
04 int printStrings (char **s), int count)
05 {
06     for (int i=0; i<count; i++) cout << s[i] << ' ';
07     cout << endl;
08 };
09
10 int main ()
11 {
12     char **darr = new char*[2];
13     for (int i=0; i<2; i++) darr[i] = new char[4];
14     strcpy (darr[0], "THE");
15     strcpy (darr[1], "COW");
16     printStrings (darr, 2);
17     for (int i=0; i<2; i++) delete[] darr[i];
18     delete[] darr;
19     return 0;
20 }
```

Programmas darbības piemērs:

```
THE COW
```

Komentāri pie pirmkoda piemēra 6.3.

- Rindā 12 masīvs *darr* tiek deklarēts (sintakse 6.5) un izveidots (sintakse 6.6; elementi: *darr[0]* un *darr[1]*). Ievērot, ka katra elementa tips ir *char**, nevis *char*.
- Rindā 13 pie abiem pirmās dimensijas elementiem tiek izveidoti otrās dimensijas masīvi *darr[0][0]..darr[0][3]* un *darr[1][0]..darr[1][3]*.
- Rindās 14 un 15 abi izveidotie otrās dimensijas masīvi tiek aizpildīti ar simbolu virknēm.
- Rindā 16 tiek izsaukta funkcija, kas izdrukā masīvu.
- Rindā 17 tiek iznīcināti otrās dimensijas masīvi (kas tika izveidoti rindā 13).
- Rindā 18 tiek iznīcināts pirmās dimensijas masīvs, kas tika izveidots rindā 12.



Attēls 6.2. Dinamisks vairākdimensiju masīvs (atbilst masīvam darr pirmkoda piemērā 6.3)

6.4. Masīvi un norādes

Valodā C++ masīviem un norādēm ir ļoti tieša saistība.

Norāde (*pointer*) ir mainīgais (vai cits mehānisms), kas glabā (nosaka) atmiņas adresi.

Varētu teikt, ka norāde satur informāciju, kas "palīdz nonākt pie citas informācijas".

Adrese (*address*) ir vērtība, kas norāda uz noteiktu vietu atmiņā.

Vizualizējot programmas darbību vai noteiktu atmiņas konfigurāciju, norāde parasti tiek attēlota kā bultiņa (sk. attēlu 6.2).

Masīvu un norāžu saistība ir tāda, ka **masīva mainīgais ir norāde**. Tādējādi var teikt, ka valodā C++ masīva mainīgais nevis reprezentē visu masīvu, bet gan tikai norāda uz tā sākumu. Kaut gan pieejamā masīva kvadrātiekavu notācija ļauj abstrahēties no šīs tehniskās detaļas, tomēr masīvu var apstrādāt arī, izmantojot norāžu notāciju.

Ieviešot jēdzienu par norādi, var izšķirt 2 veidu mainīgos:

- parastie mainīgie (glabā datus),
- norādes mainīgie (glabā adreses).

Norādes mainīgos parasti raksturo zvaigznīte (*) pie to deklarēšanas (sintakse 6.7), vienīgais izņēmums ir statiski masīvi, kas tiek deklarēti, izmantojot kvadrātiekavu notāciju, tomēr savā būtībā arī ir norādes (kaut arī ar ierobežotām izmantošanas iespējām).

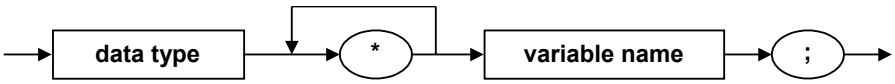
Adrese ir tehnisks jēdziens, kas domāts programmas iekšējām vajadzībām, tāpēc norāžu saturu tiešā veidā parasti neinterpretē.

Norāde savu vērtību, t.i., adresi, var iegūt šādos veidos:

- statisks masīvs kā norāde – automātiski pie deklarēšanas (pirmkods 6.4, rinda 6),
- iegūstot to no citas norādes (pirmkods 6.4, rinda 7),
- noteiktam mainīgajam – izmantojot adreses ņemšanas operatoru & (pirmkods 6.4, rindas 8,9; sintakse 6.8)

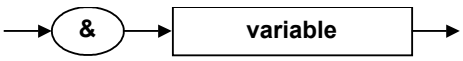
Norādes var būt vairāku pakāpju, ko nosaka vairāk nekā viena zvaigznīte pie to deklarēšanas (piemēram, pirmkods 6.3, rinda 12).

Sintakse 6.7. ***pointer variable declaration simple*** **(norādes mainīgā deklarēšana)**



Deklarējot norādi, ir svarīgi norādīt, uz kāda tipa vērtību norāda mainīgais. Tam vislielākā nozīme ir nevis pašā adrešu apstrādē, bet gan, veicot apstrādi pēc adreses iegūtajiem datiem (piemēram, masīva no veseliem skaitļiem tips ir *int**, t.i., "norāde uz *int*", kas nozīmē, ka masīva elementi jāapstrādā kā *int* tipa vērtības).

Sintakse 6.8. ***adress operator*** **(adreses ņemšanas operators)**



Pirmkoda piemērs 6.4 parāda dažādas iespējas piekļūt masīvam un tā elementiem. Piemērs parāda, ka masīva mainīgajam obligāti nav jāatbilst fiziskajam masīvam, lai izmantotu masīvu sintaksi piekļūšanai masīva elementiem. Piemēra esošās programmas darbību reprezentē darbības demonstrācija, kas parādīta zemāk.

Pirmkods 6.4. Masīvi un norādes (*ptr4ptrarr.cpp*)

```
01 #include <iostream>
02 using namespace std;
03
04 int main ()
05 {
```

```
06      int arr[5] = {11, 22, 33, 44, 55};
07      int *arr0 = arr;
08      int *arr00 = &arr[0];
09      int *arr2 = &arr[2];
10      cout << arr[1] << endl;
11      cout << arr0[1] << endl;
12      cout << arr00[1] << endl;
13      cout << arr2[1] << endl;
14      cout << &arr[3]-&arr[0] << endl;
15      cout << (char*)&arr[3]-(char*)&arr[0] << endl;
16      arr0++;
17      cout << arr0[1] << endl;
18      arr2-=3;
19      cout << arr2[1] << endl;
20      return 0;
21  }
```

Programmas darbības piemērs:

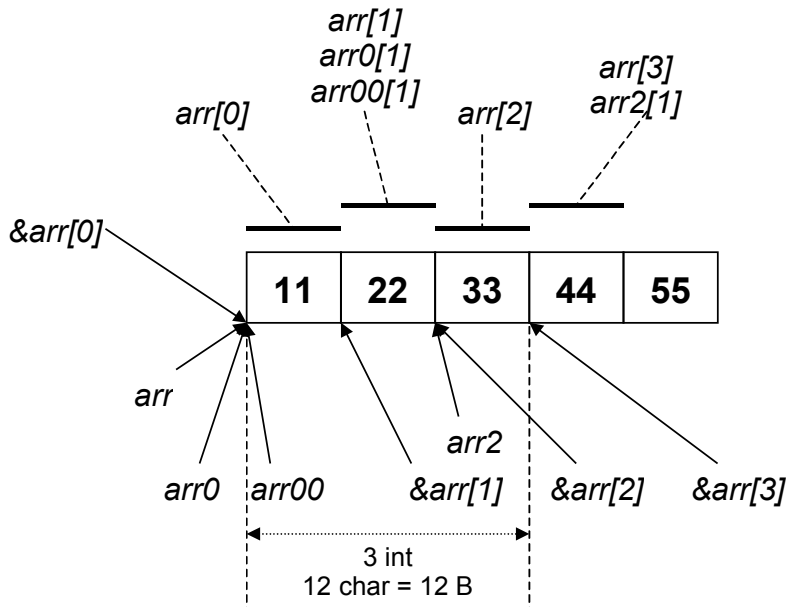
```
22
22
22
44
3
12
33
11
```

Komentāri pie pirmkoda piemēra 6.4 (pieņemot, ka *int* izmērs ir 4B).

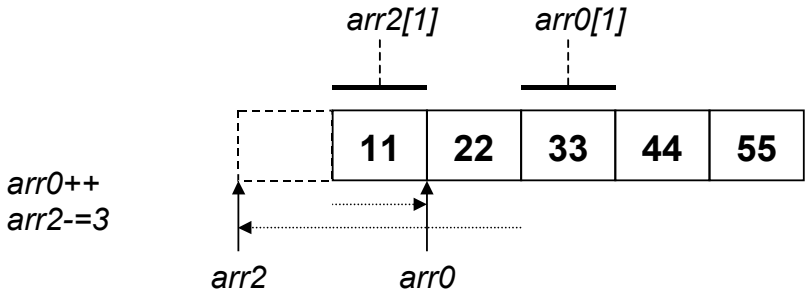
- Rindā 14 parādīta 2 adrešu starpība veselās *int* (4B) vienībās.
- Rindā 15 parādīta to pašu 2 adrešu starpība baitos (sk. arī sekojošo demonstrāciju).
- Rindās 16 un 18 adrešu aritmētika tiek veikta pa veseliem *int* apgabaliem (4B), tāpēc ++ nozīmē 4 baitus uz priekšu, bet -=3 nozīmē 12 baitus atpakaļ

Programmas darbības demonstrācija (pirmkods 6.4; pieņemot, ka *int* izmērs ir 4B).

Konfigurācija pēc rindas 15:



Konfigurācijas izmaiņa pēc rindas 19:

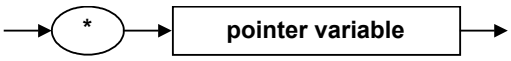


6.5. Norādes, adreses un references

Masīva mainīgais nav vienīgais norādes paveids. Norādes ir universāls līdzeklis datu pārvaldīšanai atmiņā. Saistībā ar norādēm izšķir 2 pretējas darbības:

- Adreses iegūšana noteiktam atmiņas apgabalam pēc to reprezentējošā mainīgā, izmantojot adreses ņemšanas operatoru & (sintakse 6.8);
- Vērtības iegūšana pēc adreses (sintakse 6.9; nesajaukt ar norādes mainīgā deklarēšanu!);

Sintakse 6.9. *pointer value operator* (vērtības iegūšanas operators)



Bez masīviem un simbolu virknēm norāžu mainīgos parasti izmanto dinamiskās datu struktūrās, kas tiks aprakstītas vēlāk.

Līdzīgs jēdziens norādei ir reference, kas ir pieejams tikai valodā C++.

Reference (*reference*) ir norādes paveids ar ierobežotām izmantošanas iespējām, bet ērtāku sintaksi.

Tipiskākais referenču pielietojums ir funkciju parametros, tomēr vispārīgā gadījumā to pielietojums var būt plašāks – un parasti tikai, lai uzlabotu programmas lasāmību, salīdzinot ar alternatīvo variantu, lietojot norādes.

Referenci norāda references operators &. To nedrīkst sajaukt ar adreses ņemšanas operatoru – references operators piedalās mainīgā (vai parametra) deklarēšanā.

Vēl viena references atšķirība no norādes ir tāda, ka references mainīgo nevar deklarēt bez vērtības piešķiršanas tam deklarēšanas brīdī (piemēram, pirmkods 6.5, rinda 7).

No pielietojuma viedokļa referenci varētu uzskatīt par mainīgā sinonīmu.

Norāžu un referenču lietojums parādīts pirmkodā 6.5.

Pirmkods 6.5. Norādes, adreses un references (*ptr5ptrref.cpp*)

```
01 #include <iostream>
02 using namespace std;
03
04 int main ()
05 {
06     int a = 999;
07     int &ra = a;
08     int *pa = &ra;
09     (*pa)++;
```

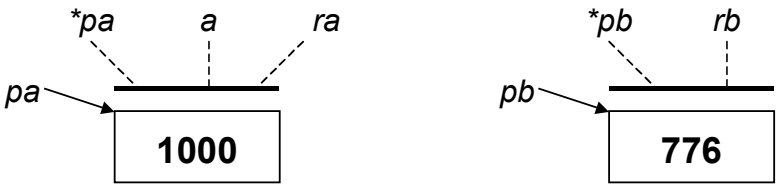
```
10      cout << a << endl;
11      int *pb = new int;
12      int &rb = *pb;
13      *pb = 777;
14      rb--;
15      cout << *pb << endl;
16      delete pb;
17      return 0;
18  }
```

Programmas darbības piemērs:

```
1000
776
```

Programmas darbības demonstrācija (pirmkods 6.5).

Konfigurācija pēc rindas 15:



Pirmkoda piemērs 6.6 parāda simbolu virkņu un norāžu saistību (pēc līdzības ar piemēru 6.4 ar *int* masīviem).

Pirmkods 6.6. Simbolu virknes un norādes (*ptr6str.cpp*)

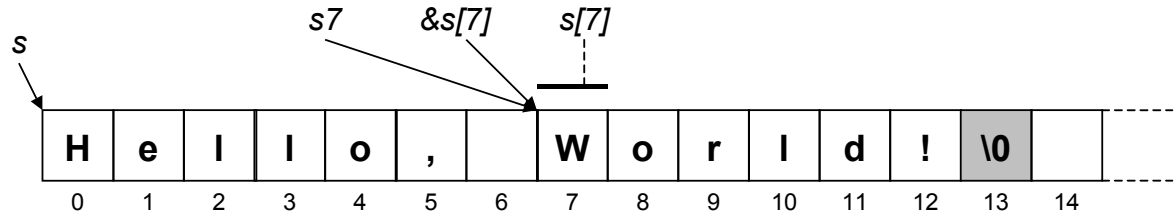
```
01 #include <iostream>
02 using namespace std;
03
04 int main ()
05 {
06      char s[20] = "Hello, World!";
07      cout << s << endl;
08      char *s7 = &s[7];
09      cout << s7 << endl;
10      s[5] = '\0';
11      cout << s << endl;
12      cout << &s[2] << endl;
13      s[5] = ',';
14      cout << &s[2] << endl;
15      return 0;
16  }
```

Programmas darbības piemērs:

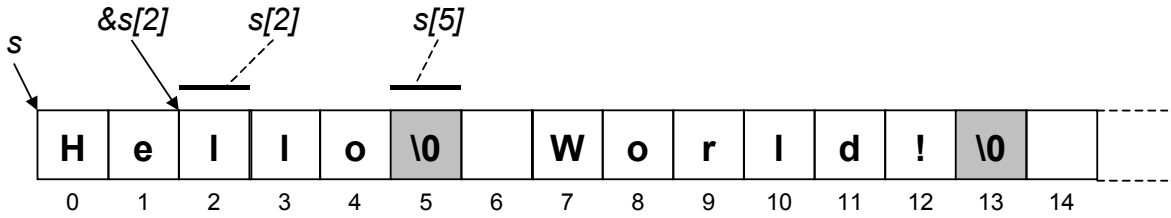
```
Hello, World!
World!
Hello
llo
llo, World!
```

Programmas darbības demonstrācija (pirmkods 6.6).

Konfigurācija pēc rindas 9:



Konfigurācija pēc rindas 12:



Konfigurācija pēc rindas 14: