# Datu pamattipi (32 bitu kompilators)

| Tips | Atmiņas apjoms, baiti | Vērtību diapazons |
|---|---|---|
| char | 1 | -128 ... 127 |
| int | 4 | -2 147 483 648 ... 2 147 483 647 |
| short | 2 | -32 768 ... 32767 |
| long | 4 | -2 147 483 648 ... 2 147 483 647 |
| unsigned char | 1 | 0 ... 255 |
| unsigned [int] | 4 | 0 ... 4 294 967 295 |
| unsigned short | 2 | 0 ... 65535 |
| unsigned long | 4 | 0 ... 4 294 967 295 |
| float | 4 | +/- $1.18*10^{-38}$ ... +/- $3.40*10^{38}$ (7 cipari) |
| double | 8 | +/- $2.23*10^{-308}$ ... +/- $1.79*10^{308}$ (15 cipari) |
| long double | 10 | +/- $3.37*10^{-4932}$ ... +/- $1.18*10^{4932}$ (19 cipari) |
| enum | 4 | -2 147 483 648 ... 2 147 483 647 |



# Datu pamattipi (16 bitu kompilators)

| Tips | Atmiņas apjoms, baiti | Vērtību diapazons |
|---|---|---|
| char | 1 | -128 ... 127 |
| int | 2 | -32 768 ... 32767 |
| short | 2 | -32 768 ... 32767 |
| long | 4 | -2 147 483 648 ... 2 147 483 647 |
| unsigned char | 1 | 0 ... 255 |
| unsigned [int] | 2 | 0 ... 65535 |
| unsigned short | 2 | 0 ... 65535 |
| unsigned long | 4 | 0 ... 4 294 967 295 |
| float | 4 | +/- $1.18*10^{-38}$ ... +/- $3.40*10^{38}$ (7 cipari) |
| double | 8 | +/- $2.23*10^{-308}$ ... +/- $1.79*10^{308}$ (15 cipari) |
| long double | 10 | +/- $3.37*10^{-4932}$ ... +/- $1.18*10^{4932}$ (19 cipari) |
| enum | 2 | -32 768 ... 32767 |

## Datu pamattipi `char` un `unsigned char`

```
char cs;
unsigned char cu;

cs = 'ā';   // iekšējais kods: 0xE2
            // skaitliskā vērtība: -30

cu = 'ā';   // iekšējais kods: 0xE2
            // skaitliskā vērtība: 226

if (cs > 'a')
   cout << "cs ir lielāks par \'a\' !" << endl;

if (cu > 'a')
   cout << "cu ir lielāks par \'a\' !" << endl;

   // 'a' iekšējais kods: 0x61
   // 'a' skaitliskā vērtība: 97
```



## Konstantes

- Speciālās `char` konstantes
  - \n - 10
  - \r - 13
  - \t - 9
  - \a - 7
  - \f - 12
  - \0 - 0
  - \xhh - heksadecimāls skaitlis (00...FF)
  - \ooo - oktāls skaitlis (000...777)

**`'A'`** **`'\x41'`** un **`'\101'`** iekšējie kodi ir vienādi: 0100 0001

```
int          25     -13    24000
long   45000  24000L 13l
double 0.5    5e-1   5E-1   1e24
float  0.5f   5e-1f
```

# Tips enum

```
enum color {BLACK, BLUE, GREEN};

color cc;
int a;

cc = GREEN;
cout << cc;        // izvada 2
// cc = 12;           // ERROR!
// cc = GREEN + BLUE; // ERROR!
cout << cc + BLUE; // izvada 3
a = cc;

enum cards {SPADE = 1, HEART = 2, DIAMOND = 4, CLUB = 8};
```



# Tips int

```
int n = 1000000000; //  n = 1 miljards
int k;

k = 3 * n;          //  k = 3 miljardi
cout << k << endl;

----------------------------------------
int n = 20000;
int k;

k = 3 * n;
cout << k << endl;
```

**Kāds būs rezultāts, ja šo pašu programmu kompilēsim 16 bitu videi?**
**-5536**

# Tipu pārveidošana

```
int n = 20000;
long k;
k = 3 * n;
cout << k << endl;
```

Šīs operācijas rezultāta tips ir int, jo rezultātam vienmēr ir tāds pats tips kā operandiem!

**Kāds būs rezultāts 16 bitu vidē?**

**-5536**

**Tipa pārveidošanas operācija (typecasting)** `(tips)izteiksme`

```
int n = 20000;
long k;
k = (long) (3 * n);
cout << k << endl;
```

**Kāds būs rezultāts 16 bitu vidē?**

**-5536**

*169*

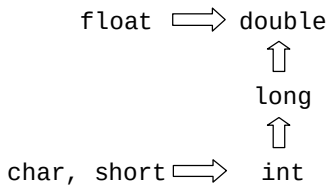
# Tipu pārveidošana

```
int n = 20000;
long k;
k = 3 * (long)n;
cout << k << endl;
```

Šīs operācijas rezultāta tips ir long

**Rezultāts arī 16 bitu vidē būs 60000**

```
int n = 20000;
long k;
k = 3L * n;
cout << k << endl;
```




**Rezultāts arī 16 bitu vidē būs 60000**

*170*

# Tipu pārveidošana

- Tipu pārveidošana notiek automātiski, ja:
  - deklarācijā sākumvērtības izteiksmes tips atšķiras no objekta tipa
    ```
    double x = 1;
    ```
  - funkcijas faktiskā argumenta tips atšķiras no parametra tipa funkcijas deklarācijā
    ```
    long maxSize(int, long, double);
    ...
    k = maxSize(12, 25, 0.5);
    ```
  - izteiksmes tips operatorā return atšķiras no funkcijas deklarācijā norādītā funkcijas tipa
    ```
    long maxSize(int n, long k, double w)
    {   ...
       return k * n;
    }
    ```
  - izteiksmē divu operandu tipi ir atšķirīgi
    ```
    long max;
    ...
    k = max + 1;
    ```



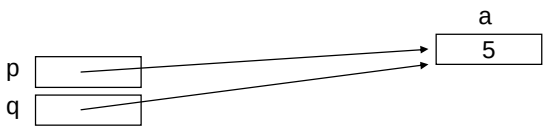
# Tipu pārveidošana

```
MyString a("ABC");
MyString b = a;          //strādā kopijas konstruktors
MyString x = "ABC";      //notiek tipa pārveidošana
MyString z = 13;         //notiek tipa pārveidošana !!!

x = (MyString)"123";     //notiek tipa pārveidošana
```

- Lai pārveidotu tipu klases objektam, tiek izmantots konstruktors, kura parametra tips sakrīt ar pārveidojamās izteiksmes tipu.

# Rādītāji

```
int a;
int *p;
int* q;

a = 5;
p = &a;
q = p;
```




```
*p = 13;  // mainīgajam a netieši piešķir vērtību 13
*q = -1;  // mainīgajam a netieši piešķir vērtību -1

double *r;
r = &a;   // KĻŪDA! Varētu pārveidot tipu:  r = (double*)&a;

char c, *pc;
c = 'A';
pc = &c;
*p = *pc;  // mainīgajam a netieši piešķir vērtību 65
```

*173*

# Masīvi

```
int m[5];  // atmiņā aizņem 20 baitus
```

| m[0] | m[1] | m[2] | m[3] | m[4] |
|---|---|---|---|---|

```
float a[2][3]; // atmiņā aizņem 24 baitus
```

| a[0][0] | a[0][1] | a[0][2] | a[1][0] | a[1][1] | a[1][2] |
|---|---|---|---|---|---|

```
int n = 10;
int beta[n]; // kļūda!

const int MaxSize = 12;
int gamma[MaxSize]; // OK

Sākumvērtību piešķire:
int  m[5] = {0, 0, 1, 1};
```

| m[0] | m[1] | m[2] | m[3] | m[4] |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | ? |

*174*

# Rādītāji un masīvi

Masīva identifikators bez indeksa ir rādītājs uz masīva pirmo elementu

```
int mas[6];
```

**mas un &mas[0]**
**vērtības ir vienādas – tā ir masīva sākuma elementa adrese**

```
int *p;
p = &mas[0];
*p = 10;   // masīva elementam mas[0] piešķir vērtību 10
*mas = 20; // masīva elementam mas[0] piešķir vērtību 20
```

**mas + n ir masīva n-tā elementa adrese**

```
*(mas + 1) = 30; // mas[1] piešķir vērtību 30
*(mas + 2) = 40; // mas[2] piešķir vērtību 40
*(p + 4) = 50;   // mas[4] piešķir vērtību 50
```

*175*

# Rādītāji un masīvi

**pieraksts mas[i] ir ekvivalents pierakstam *(mas+i)**

```
char t[100];
char *pc = t;

*(pc + 4) = 'Z';

double v[100];
double *pd = v;

*(pd + 4) = 0.5;

int *pm[5]; // rādītāju masīvs
```

*176*

# Simbolu masīvi

```
char n[10] = { 'A', 'i', 'v', 'a', 'r', 's', '\0' };
```

| 'A' | 'i' | 'v' | 'a' | 'r' | 's' | '\0' | ?? | ?? | ?? |
|---|---|---|---|---|---|---|---|---|---|

```
char name[10] =  "Aivars";
char message[] = "Ievadi savu vārdu: ";

int len, size;
size = sizeof(name);    // 10
size = sizeof(message); // 20
len = strlen(name);     // 6
len = strlen(message);  // 19

// name = "Leontīne";         // Kļūda!
strcpy(name, "Magdalēna");
strcpy(name, "Kristofers");      // !!!
```

177

**Simbolu virknes konstantes tips ir char***

# Simbolu virkņu masīvi

```
char list[3][10] = { "aaa", "BB", "123" };
```

| 'a' | 'a' | 'a' | '\0' | ?? | ?? | ?? | ?? | ?? | ?? |
|---|---|---|---|---|---|---|---|---|---|
| 'B' | 'B' | '\0' | ?? | ?? | ?? | ?? | ?? | ?? | ?? |
| '1' | '2' | '3' | '\0' | ?? | ?? | ?? | ?? | ?? | ?? |

```
list[0][1] = '\0';
```

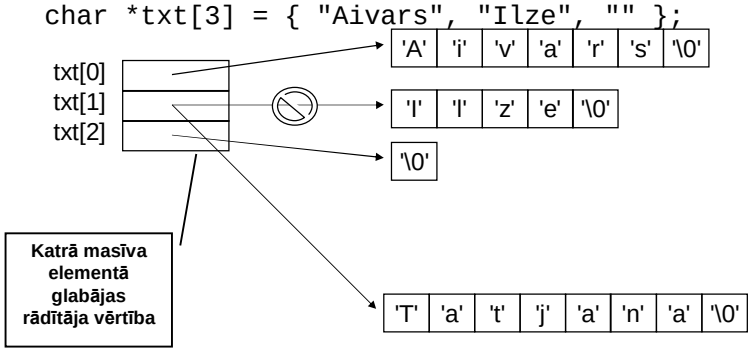
| 'a' | '\0' | 'a' | '\0' | ?? | ?? | ?? | ?? | ?? | ?? |
|---|---|---|---|---|---|---|---|---|---|
| 'B' | 'B' | '\0' | ?? | ?? | ?? | ?? | ?? | ?? | ?? |
| '1' | '2' | '3' | '\0' | ?? | ?? | ?? | ?? | ?? | ?? |

```
list[0][1]  ir ekvivalents *(list[0] + 1)
list[0]     ir pirmās rindas sākuma adrese
list[2] = "456"; // Kļūda!
```

178

# Simbolu virkņu masīvi

```
char *txt[3] = { "Aivars", "Ilze", "" };
```



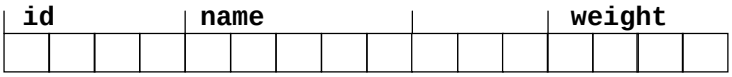

```
txt[1] = "Tatjana";
txt[0][2] = 'g'; // *(txt[0] + 2) = 'g';
cout << txt[0] << endl; // izvadīs Aigars
```

`txt` **tips ir** `char**`



# Struktūras

```
struct person
{
   int id;
   char name[5];
   float weight;
};
...
cout << sizeof(person) << endl; // 16!
```




#pragma pack(*n*), kur n = 1, 2, 4, 8, 16

# Struktūras

```
#pragma pack(1)
...
struct person
{
   int id;
   char name[5];
   float weight;
};
...
cout << sizeof(person) << endl;                // 13
```