

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

n 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 | 131 | |
| 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

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

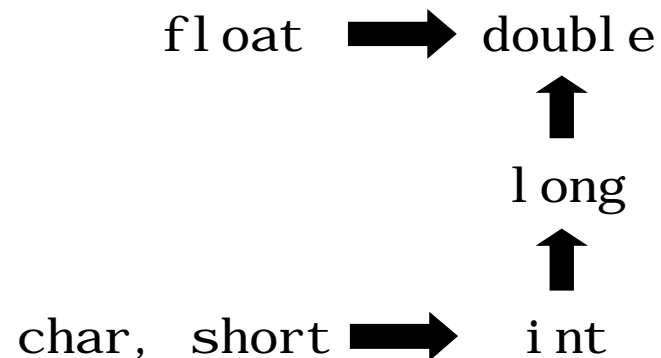
```
int n = 20000;
```

```
long k;
```

```
k = 3L * n;
```

```
cout << k << endl;
```

Rezultāts arī 16 bitu vidē būs 60000



Tipu pārveidošana

n Tipu pārveidošana notiek automātiski, ja:

§ deklarācijā sācumvērtības 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

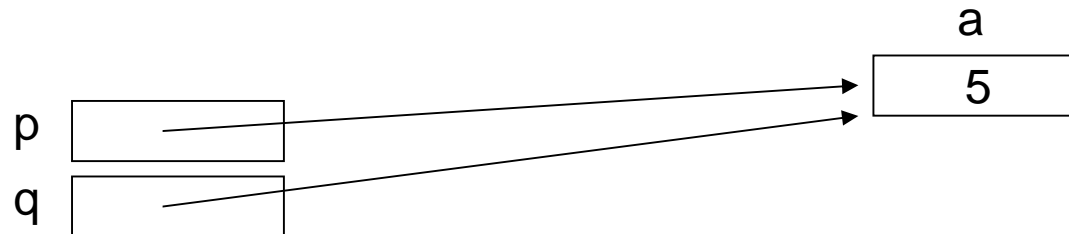
```
MyString a("ABC");  
MyString b = a;           // strādā kopijas konstruktors  
MyString x = "ABC";       // notiek tipa pārveidošana  
MyString z = 13;          // notiek tipa pārveidošana !!!  
  
x = (MyString)"123";      // notiek tipa pārveidošana
```

nLai 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
```

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
```

Sācumvērtību piešķire:

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

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

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | ? |
|---|---|---|---|---|

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
```

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
```

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");    // !!!
```

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' | ?? | ?? | ?? | ?? | ?? | ?? |

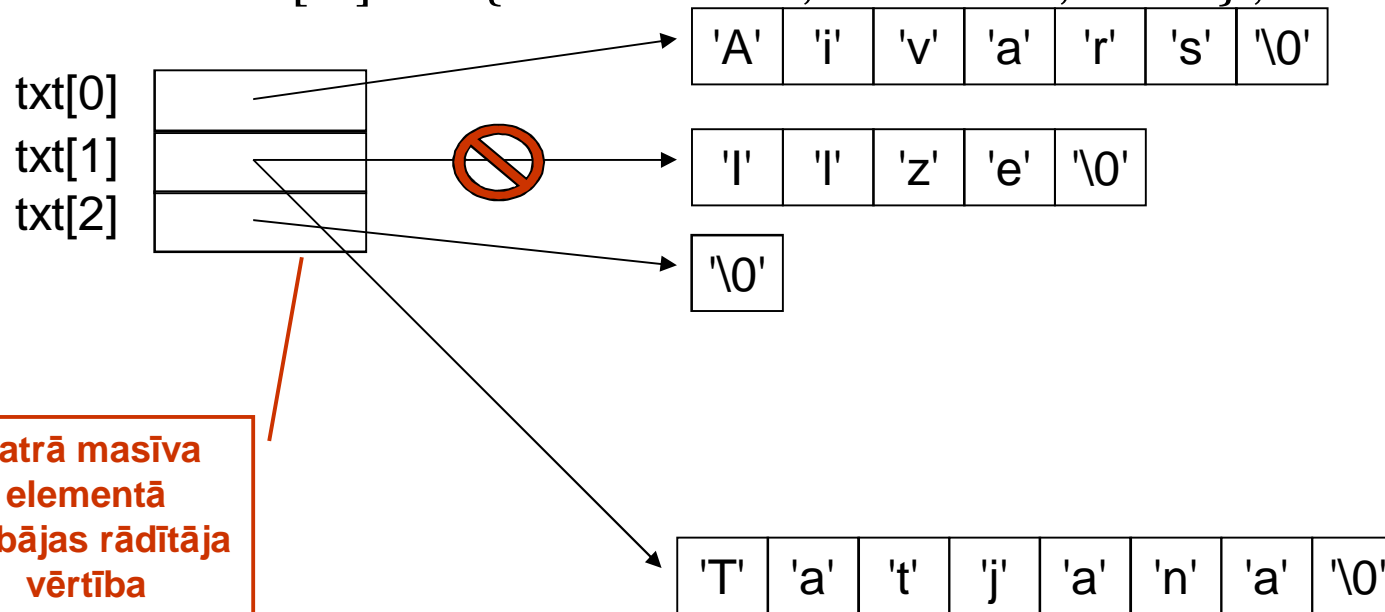
`list[0][1]` ir ekvivalents `*(list[0] + 1)`

`list[0]` ir pirmās rindas sākuma adrese

`list[2] = "456";` // Kļūda!

Simbolu virkņu masīvi

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



```
txt[1] = "Tatjana";
```

```
txt[0][2] = 'g'; // *(txt[0] + 2) = 'g';
```

```
cout << txt[0] << endl; // izvadīs Ai gars
```

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];
```

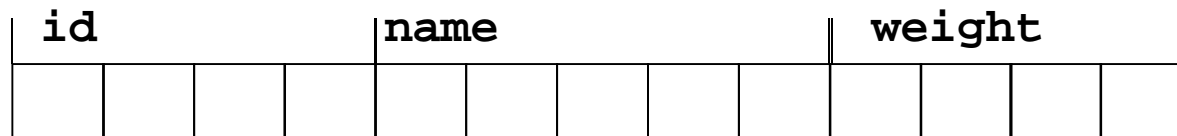
```
    float weight;
```

```
};
```

```
...
```

```
cout << sizeof(person) << endl;
```

```
// 13
```



Operācijas (angļu val. *operators*)

| # | Category | Operator | Associativity |
|-----|----------------|-------------------------------------|--------------------|
| 1. | Highest | () [] -> :: . | $\frac{3}{4}$ ® |
| 2. | Unary | ! ~ + - ++ -- & * sizeof new delete | $\neg \frac{3}{4}$ |
| 3. | Member access | . * ->* | $\frac{3}{4}$ ® |
| 4. | Multiplicative | * / % | $\frac{3}{4}$ ® |
| 5. | Additive | + - | $\frac{3}{4}$ ® |
| 6 | Shift | << >> | $\frac{3}{4}$ ® |
| 7. | Relational | < <= > >= | $\frac{3}{4}$ ® |
| 8. | Equality | == != | $\frac{3}{4}$ ® |
| 9. | Bitwise AND | & | $\frac{3}{4}$ ® |
| 10. | Bitwise XOR | ^ | $\frac{3}{4}$ ® |
| 11. | Bitwise OR | | $\frac{3}{4}$ ® |
| 12. | Logical AND | && | $\frac{3}{4}$ ® |
| 13. | Logical OR | | $\frac{3}{4}$ ® |
| 14. | Conditional | ? : | $\neg \frac{3}{4}$ |
| 15. | Assignment | = *= /= %= += -= &= ^= = <<= >>= | $\neg \frac{3}{4}$ |
| 16. | Comma | , | $\frac{3}{4}$ ® |

Operācijas ar datiem

```
int x, y;  
x = 4;  
y = x | x >> 1 > 1; // y iegūst vērtību 5
```

```
int a = 20;  
Izteiksmes 5 < a < 10 vērtība ir: 1
```

```
int i, j, m[4];  
i = 2;  
j = ++i; // i iegūst vērtību 3, j iegūst vērtību 3
```

```
i = 2;  
j = i++; // i iegūst vērtību 3, j iegūst vērtību 2
```

```
i = 0;  
m[i++] = 7; // m[0] iegūst vērtību 7, i iegūst vērtību 1
```

Operācijas ar datiem

```
char str[] = "*aa*bbb***cccc*****";
```

Noteikt, cik simbolu '*' ir simbolu virknē str.

operācijas * operands ir "vecā",
nepalielinātā s vērtība

```
int stars = 0;
```

```
char *s = str;
```

```
while (*s) stars += *s++ == '*'; // stars += (*(s++) == '*')
```

```
// "saprotamāka" realizācija
```

```
int stars = 0;
```

```
int len = strlen(str);
```

```
for (int i = 0; i < len; ++i)
```

```
    if (str[i] == '*') ++stars;
```

Operācijas ar datiem

n Divvietīgā operācijā, operandu aprēķināšanas secība nav noteikta, izņemot operācijas `||`, `&&` un `,` (komats).

```
k = 5;
```

```
alpha = ++k + MyFun(k); // MyFun() saņems 5 vai 6?
```

```
int total, sum;
```

```
total = 0;
```

```
sum = (++total) + (total = 3);
```

```
cout << total << sum << endl; // 3 6
```

```
total = 0;
```

```
sum = (total = 3) + (++total);
```

```
cout << total << sum << endl; // 4 7
```

Operācijas ar datiem

- n Loģiskajās operācijās && un || operandu vērtības aprēķina no kreisās puses uz labo.
- n Var būt gadījumi, kad ne visas operandu izteiksmes tiek aprēķinātas!

```
if (a > 0 || ++k > 10)  
    //ja a > 0, tad k nepalielinās un nepārbaudīs!
```

```
if (++x <= n && ++y <= m)  
    //ja ++x > n, tad y nepalielinās un nepārbaudīs!
```


Operācijas ar datiem

- n Operācijas "komats" operandus aprēķina no kreisās puses uz labo. Tās vērtība un tips ir pēdējās aprēķinātās izteiksmes vērtība un tips.
- n Šo operāciju visbiežāk lieto, lai secīgi aprēķinātu vairākas izteiksmes vietās, kur C++ sintakse pieļauj tikai vienu izteiksmi.

`izteiksme1, izteiksme2, ..., izteiksmen`

```
for (i = 0, j = n; i <= n; ++i, --j)
    a[i] = b[j];
```

...

```
int someFun(int, int, int);
```

...

```
int p = someFun(i, (j = 1, j + 4), k);
// someFun() saņem 3 parametrus: i, 5, k
```

Operācijas new un delete

n Operācijas new vērtība ir attiecīgā tipa rādītājs uz iedalīto atmiņu:

```
new tips  
new tips[izteiksme]
```

```
int *pp1, *pp2, *pp3;  
pp1 = new int;  
pp2 = new int[2048];  
int n = 5;  
pp3 = new int[1024 * n];
```

```
Triangle *pt = new Triangle[1000];
```

n Operācija delete atbrīvo ar new iedalīto atmiņu:

```
delete rādītājs
```

```
delete pp1;  
delete pp2;  
delete pp3;  
delete pt;
```

Operācijas new un delete

n Ieteikumi darbā ar rādītājiem:

- § Nelietot neinicializētu rādītāju;
- § Neatbrīvot ar delete atmiņu, kas nav iedalīta ar new;
- § Neizpildīt delete vairākkārtīgi vienam un tam pašam rādītājam.

```
#define NULL 0
```

```
char *buf = NULL;
```

```
...
```

```
buf = new char[size];
```

```
if (!buf) exit(1); //atmiņas iedalīšana nebija veiksmīga!
```

```
...
```

```
if (buf) { delete buf; buf = NULL; }
```

Funkcijas

n Funkcijas deklarācija:

```
tips vārds (parametru tipu saraksts);  
tips vārds (parametru deklarāciju saraksts);
```

n Funkcijas definīcija:

```
tips vārds (parametru deklarāciju saraksts)  
{ funkcijas ķermenis }
```

n Funkcijas ar mainīgu parametru skaitu:

```
int varFun(char *c, ...);
```

n Funkciju izsaucot, faktiskie parametri tiek pārveidoti atbilstoši deklarācijā norādītajiem tipiem un vērtības tiek ievietotas stekā.

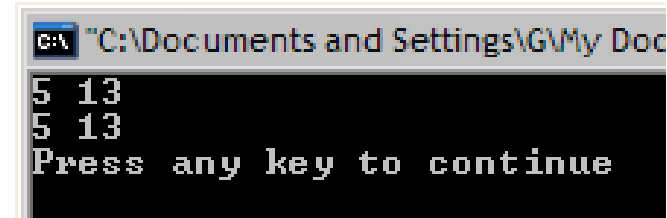
n Funkcija savā darbā parametru vērtības izmanto no steka, t.i. izmanto faktisko parametru kopijas.

Funkcijas

- n Uzdevums: Uzrakstīt funkciju, kas samaina vietām savu divu argumentu vērtības.

```
void main()
{ void swap(int, int); // deklarācija
  int x = 5, y = 13;
  cout << x << " " << y << endl;
  swap(x, y);
  cout << x << " " << y << endl;
}
```

```
void swap(int a, int b)
{
  int c = a; a = b; b = c;
}
```



A screenshot of a Windows command prompt window. The title bar shows the path "C:\Documents and Settings\G\My Doc". The command prompt displays the output of the program: "5 13" on the first line, "5 13" on the second line, and "Press any key to continue" on the third line.

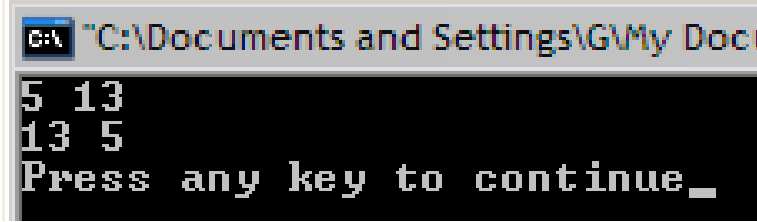
Kas tiks izvadīts uz ekrāna?

Funkcijas

- n Uzdevums: Uzrakstīt funkciju, kas samaina vietām savu divu argumentu vērtības.

```
void main()  
{  
    int x = 5, y = 13;  
    void swap(int*, int*); // deklarācija  
    cout << x << " " << y << endl;  
    swap(&x, &y);  
    cout << x << " " << y << endl;  
}
```

```
void swap(int *a, int *b)  
{  
    int c = *a; *a = *b; *b = c;  
}
```



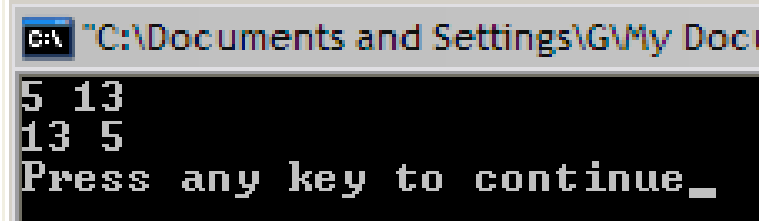
The screenshot shows a Windows command prompt window with the title bar "C:\Documents and Settings\G\My Doc...". The window contains the following text: "5 13", "13 5", and "Press any key to continue_". This represents the output of the program, showing the values of x and y before and after the swap function is called.

Funkcijas

- n Uzdevums: Uzrakstīt funkciju, kas samaina vietām savu divu argumentu vērtības.

```
void main()  
{  
    int x = 5, y = 13;  
    void swap(int&, int&); // deklarācija  
    cout << x << " " << y << endl;  
    swap(x, y);  
    cout << x << " " << y << endl;  
}
```

```
void swap(int& a, int& b)  
{  
    int c = a; a = b; b = c;  
}
```



```
C:\Documents and Settings\G\My Doc  
5 13  
13 5  
Press any key to continue_
```

Rekursīvas funkcijas

```
int pw(int n, int k)
{  if (k == 1) return n;
   else return n * pw(n, k - 1);
}
```

$$n^k = n * n^{k-1}$$

```
void main()
{
    int y;
    int x = 2;

    y = pw(x, 3);
    cout << y << endl;
}
```


Rekursīvas funkcijas

Uzlabota versija

```
int pw(int n, int k)
{
    if (k == 0)
        return 1;
    else if (k == 1)
        return n;
    else
        return n * pw(n, k - 1);
}
```