Programmētāju skola
1. līmeņa grupa

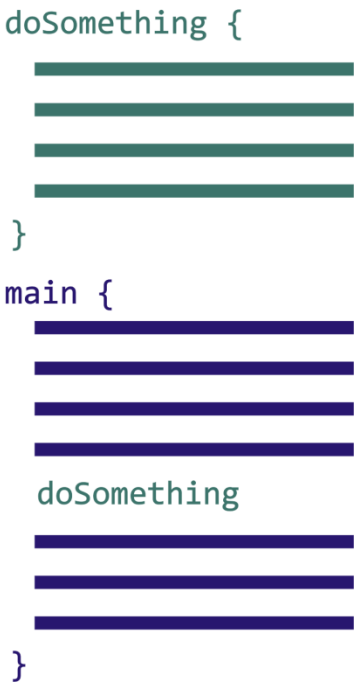
funkcijas

Apakšprogrammas

Apakšprogramma (procedūra, funkcija) – programmas koda fragments, kas ir aprakstīts atsevišķa bloka veidā un var būt izpildīts (izsaukts) jebkurā programmas vietā.

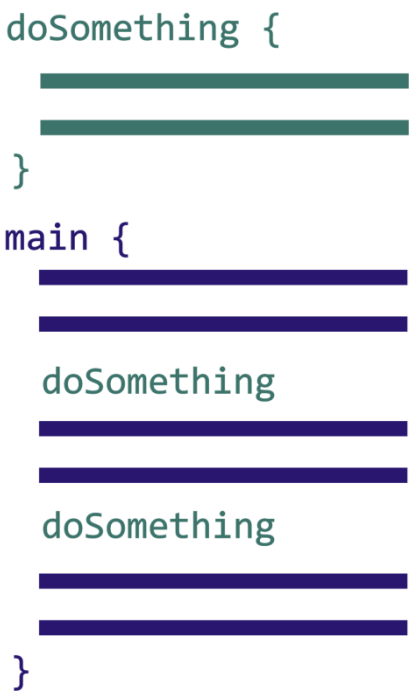
Lineāra programma

Procedūrorientēta programma

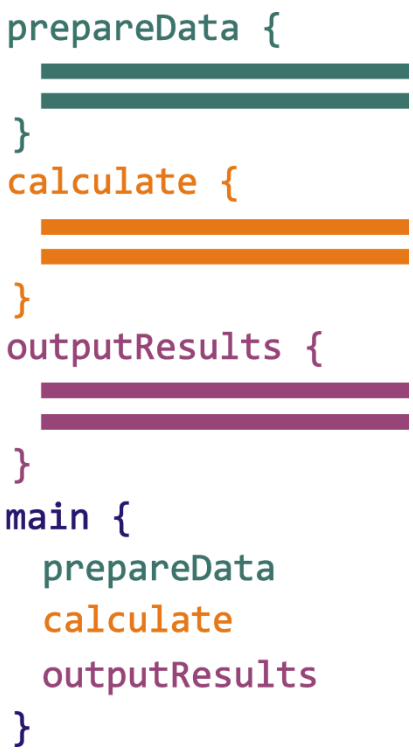


Priekš kam vajag?

1. Lai izslēgtu no koda atkārtojošos fragmentus



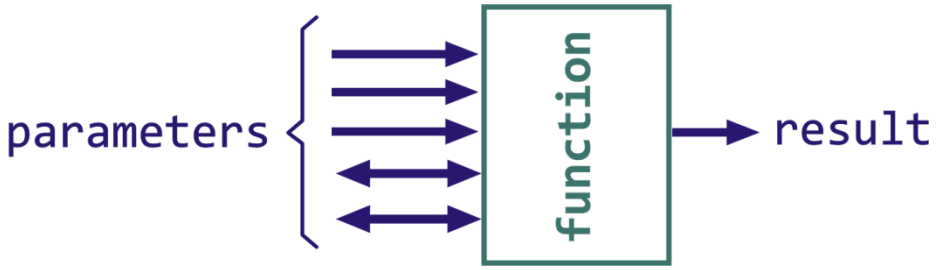
2. Lai strukturētu programmas kodu



- izstrādāšanas, atkļūdošanas un modifikācijas ērtība;
- atkārtotas koda pielietošanas iespēja (bibliotēkas);
- komandas izstrādāšanas iespēja.

Funkcijas interfeiss

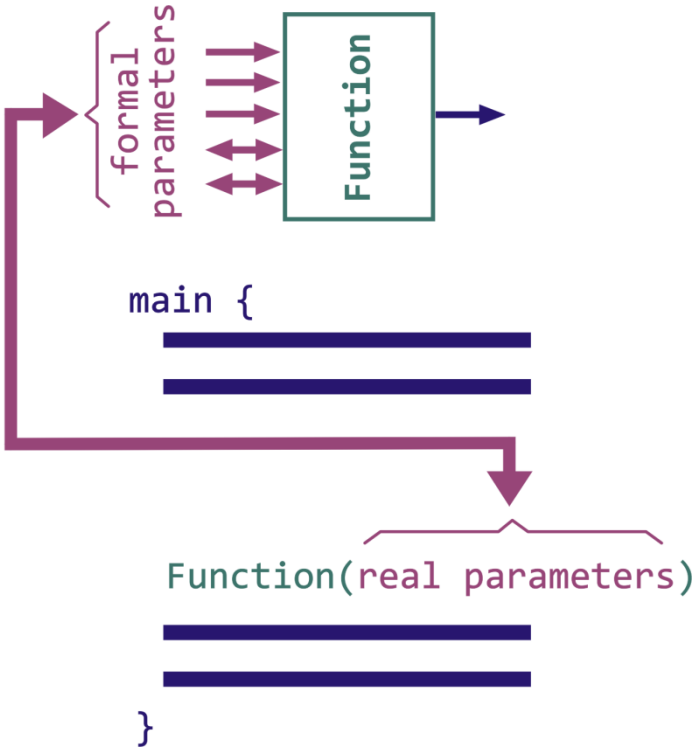
Funkcijas izsaukšanas laikā tajā var nodot ieejas datus un pēc pabeigšanas saņemt rezultātus. Tam pielieto parametrus un atgriežamo rezultātu.



Procedurālā abstrakcija – programmas strukturēšanas veids, kad programma tiek sadalīta apakšprogrammās pēc "melnās kastes" principa. Ideālā gadījumā katra apakšprogramma jābūt noformēta tā, ka, lai ar to strādātu, nepieciešams zināt tikai to, kā nodot tajā sākumdatus un saņemt rezultātus (zināt interfeisu), bet nav obligāti zināt kā tā ir uzbūvēta.

Parametri

C valodā funkcijai var būt parametru saraksts, izmantojot kuru var nodot uz funkciju ieejas datus un saņemt no tā izejas rezultātus.



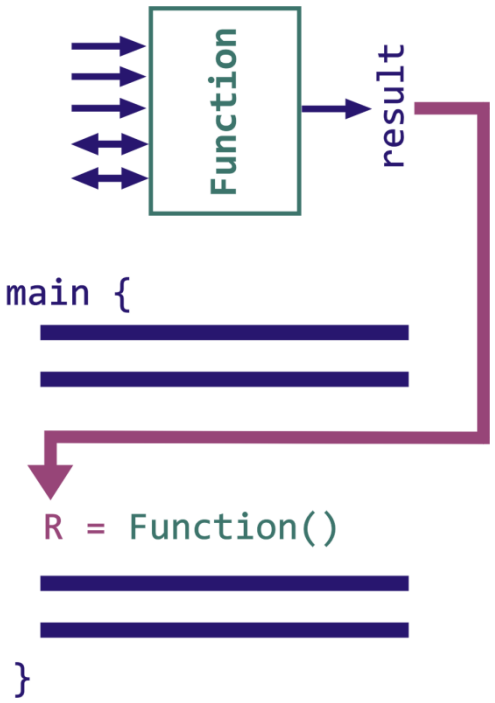

```
int main()
{
     std::cout << "1 + 2 = " << 1 + 2 << "\n";
}
```

```
void showSum()
{
    std::cout << "1 + 2 = " << 1 + 2 << "\n";
}
int main()
{
    showSum();
}
```

```
void showSum(int a, int b)
{
    std::cout << a << " + " << b << " = "
              << a + b << "\n";
    }
int main()
{
    showSum(1, 2);
}
```

Atgriežamais rezultāts

C valodā katrai funkcijai pēc darba nobeiguma ir jāatgriež rezultāts (var būt tukšais).




```
x = sin( 12 );

x = function();

function();

if ( function() > 0 ) ...

std::cout << "result = " << function() << "\n";
```

```
int getSum(int a, int b)
{
    return a + b;
}

int main()
{
    std::cout << "1 + 2 = " << getSum(1, 2);
}
```

Deklarācija

```
type name ( f_params ){ body; }
```

type atgriežama rezultāta tips, vai `void` ja funkcija neatgriež rezultātu

name vārds, kuru ir jānorāda izsaukšanas laikā

f_params neobligātais formālo parametru saraksts, kas iz jāizmanto lai nodotu uz funkciju sākumdatus un saņemt rezultātus:

$\mathbf{type}_1$ $\mathbf{name}_1$, $\mathbf{type}_2$ $\mathbf{name}_2$, ...

body ķermenis, kas būs izpildīts funkcijas izsaukšanas laikā

```
float sum( float a, float b ) {...}
void show( int value ) { ... }
float solve(int a, float b) { ... }
bool check(float data) { ... }
void do() { ... }
char get() { ... }
```

Izsaukšana

```
name ( r_params );
```

name	izsaucamas funkcijas vārds;
()	operācija "izsauc funkciju";
r_params	argumenti jeb reālo parametru saraksts, satur sākumvērtības formāliem parametriem.

```
float sum( float a, float b )
void show( int value )
float solve(int a, float b)
bool check(float data)
void do()
char get()
```

```
x = sum(1.5, 0.5);
int i = 4; show(i);
int var = 5;
if ( check( solve(var, var*2) ) ) ...
do();
std::cout << get();
```

Parametru nodošana

C++ atbalsta vismaz divus parametru nodošanas veidus: **parametri-vērtības** un **parametri-references**.

Funkcijas izsaukšanas laikā katram **parametram-vērtībai** tiks izveidots pagaidu mainīgais ar sākumvērtību no reālo parametru saraksta. Pēc funkcijas pabeigšanas visi šie mainīgie tiks izdzēsti un to vērtības tiks pazaudētas.

```
void show(int v)
{
    std::cout << v << " ";
    v *= 2;
}

int main()
{
    int a = 1;
    for (int i = 0; i < 5; i++) show(a);
}
```

1 1 1 1 1

Funkcijas izsaukšanas laikā **parametriem-referencēm** pagaidu mainīgie netiks izveidoti. Funkcijas darba laikā šo parametru vietā tiks izmantoti mainīgie no reālo parametru saraksta un tā vērtības netiks pazaudētas pēc funkcijas pabeigšanas.

```
void show(int &v)
{
    std::cout << v << " ";
    v *= 2;
}

int main()
{
    int a = 1;
    for (int i = 0; i < 5; i++) show(a);
}
```

1 2 4 8 16

Rezultātu atgriešana

Funkcijas pabeigšanai un rezultāta atgriešanai izmanto operatoru **return**.

```
return result;
```

```
int sum(int a, int b)
{
    int c = a + b;
    return c;
}
```

```
int getMax(int a, int b)
{
    if (a > b) return a;
    return b;
}
```

```
int sum(int a, int b)
{
    return a + b;
}
```

```
void showMax(int a, int b)
{
    if (a == b) return;
    std::cout << "max = " << getMax(a, b);
}
```

Piemērs

```
bool sqSolve(float a, float b, float c, float &x1, float &x2)
{
    float d = b * b - 4 * a * c;
    if (d < 0 || a == 0) return false;
    x1 = (-b + sqrt(d)) / (2 * a);
    x2 = (-b - sqrt(d)) / (2 * a);
    return true;
}

int main()
{
    float x, y;
    if (sqSolve(1, -5, 6, x, y)) std::cout << "x1=" << x << "\nx2=" << y ;
    else std::cout << "Roots not found\n";
}
```

Masīvu nodošana

C++ atbalsta masīvu nodošanu uz funkcijas tikai caur references, izmantojot speciālo sintaksi.

```
type name [Size]      type name []

type name [Rows][Cols]      type name [][Cols]
```

```
void randomArray(int array[], int count) {
    for (int i = 0; i < count; i++) array[i] = rand() % 10;
}
void showArray(int array[], int count) {
    for (int i = 0; i < count; i++) {
        std::cout << array[i] << i % 10 == 9 ? "\n" : "";
    }
}
int main()
{
    srand( time(NULL) );
    int a[100];
    randomArray(a, 100);
    showArray(a, 100);
}
```

Masīvu atgriešana

C++ nevar atgriezt masīvu kā funkcijas rezultātu.

Masīvu atgriešanai ir jāpielieto parametri-masīvi.

```
int[] get() {...}
```

```
void get( int arr[] ) {...}
```

Lokālie un globālie mainīgie

Lokālie mainīgie – mainīgie, kuri ir deklarēti iekšpus funkcijas ķermeņa un var būt izmantoti tikai iekšpus šīs funkcijas.

```
void func() { int var; ... }
```

Globālie mainīgie – mainīgie, kuri ir deklarēti ārpus funkcijas un var būt izmantoti iekšpus jebkuras funkcijas šajā programmā.

```
int var; void func() { ... }
```

Ja lokālā un globālā mainīgās vārdi sakrīt, tad kompilators pēc noklusējama izmantos lokālo mainīgo. Lai piekļūtu pie globālā mainīgā nepieciešams norādīt pirms tā `::`

```
int var;
void func() {
    int var;
    var = 1;    // local
    ::var = 2;  // global
}
```

Funkcijas prototipi

Parasti funkcija jābūt realizēta pirms izsaukšanas. Ja nepieciešams izsaukt funkciju agrāk kā tā būs realizēta, ir jāizmato funkcijas prototipu.

```
type name ( f_params );
```

```
void funcA() {
    ...
    funcB();
    ...
}

void funcB() {
    ...
    funcA();
    ...
}
```

```
void funcB();

void funcA() {
    ...
    funcB();
    ...
}

void funcB() {
    ...
    funcA();
    ...
}
```

Parametru vērtības pēc noklusējuma

C++ ļauj funkcijas vai tā prototipa deklarācijas laikā formāliem parametriem noradīt vērtības, kuri būs izmantoti, ja izsaukšanas laikā attiecīgie reālie parametri nebūs noradīti.

```
void func(int a, int b = 20, int c = 30) { ... }

      func(1, 2, 3);  // a=1, b=2, c=3
      func(1, 2);     // a=1, b=2, c=30
      func(1);        // a=1, b=20, c=30
      func();         // error
      func(1, , 3);   // error
```

Parametri ar vērtībām pēc noklusējuma jābūt pēdējiem formālo parametru sarakstā.

```
void func(int a, int b = 10, int c = 20) { ... }
void func(int a, int b, int c = 20) { ... }
void func(int a, int b = 10, int c) { ... }
```

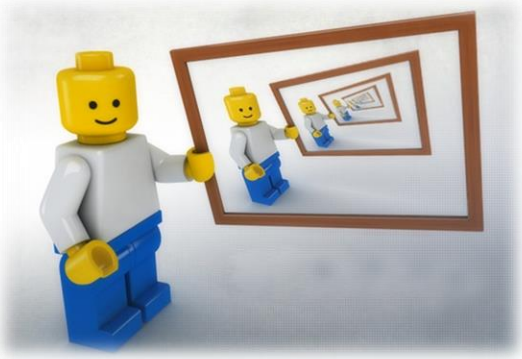
Funkciju pārdefinēšana

C++ ļauj definēt funkcijas ar vienādiem vārdiem bet ar atšķirīgiem formālo parametru sarakstiem. Šajā gadījumā kompilators noskaidro kādu funkciju izsaukt pēc reālo parametru sarakstam.

```
void func(int a) { ... }          // function 1
void func(int a, int b) { ... }   // function 2
void func(double a) { ... }       // function 3
void func() { ... }               // function 4
```

```
func(1.5);    // function 3
func(10);     // function 1
func();       // function 4
func(1, 2);   // function 3
```

Rekursija

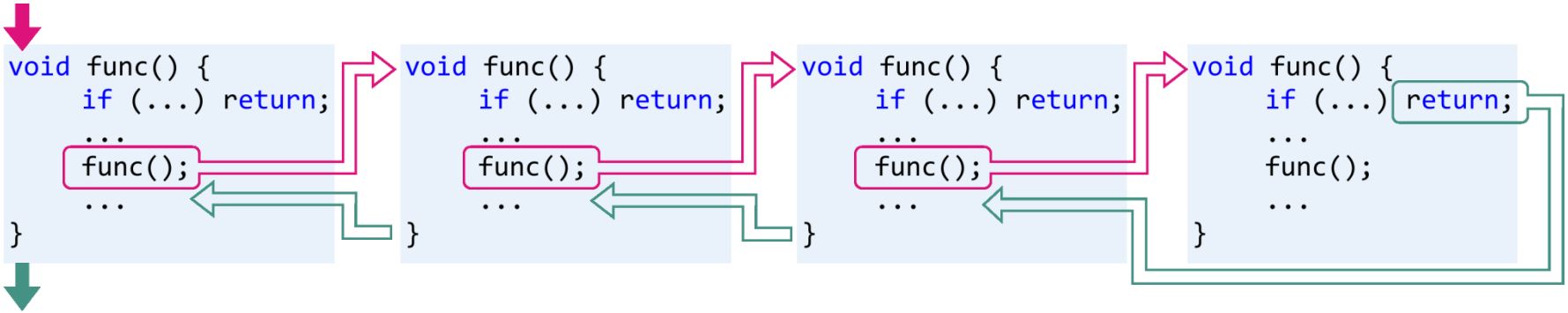


Lai saprastu rekursiju, vispirms ir jāsaprot rekursija

Для того чтобы понять рекурсию, надо сначала понять рекурсию

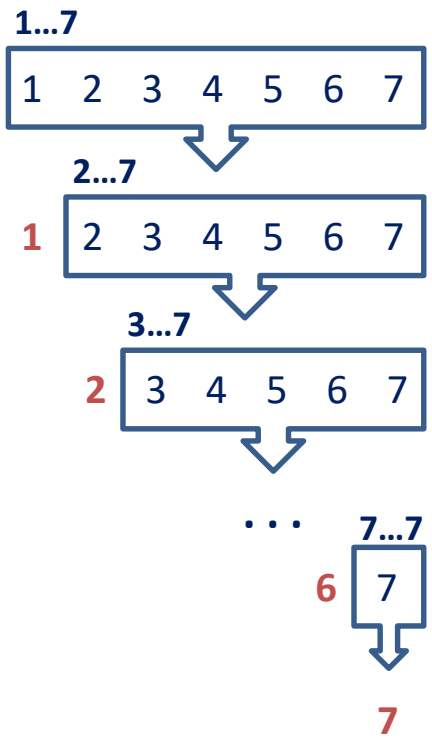
Rekursija ir programmēšanas pieeja, kad funkcija izsauc pati sevi.

```
void func() {
    if (...) return;   // izeja no rekursijas
    ...                // tiešā gaita
    func();            // ieeja uz rekursiju
    ...                // apgrieztā gaita
}
```



Rekursijas piemērs

Izvadīt uz ekrāna visus naturālus skaitļus diapazonā no Min līdz Max.




```
void showChain(int Min, int Max) {
    if (Min > Max) return;
    std::cout << Min;
    showChain(Min + 1, Max);
}

int main() { showChain(1, 7); }
```

1 2 3 4 5 6 7

```
void showChain(int Min, int Max) {
    if (Min > Max) return;
    showChain(Min + 1, Max);
    std::cout << Min;
}

int main() { showChain(1, 7); }
```

7 6 5 4 3 2 1

Faktoriāls

Aprēķināt uzdota naturāla skaitļi N faktoriālu N! = 1·2·3 ·… ·N.

7! = 6! · 7

6! = 5! · 6

5! = 4! · 5

...

1! = 0! · 1

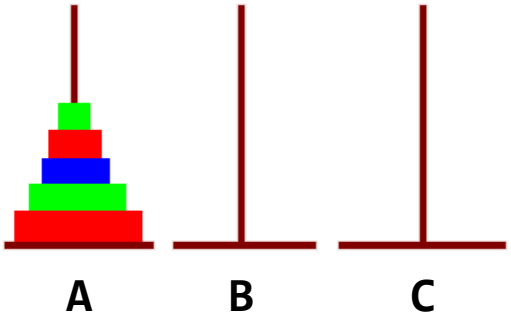
0! = 1

```
int getFactorial(int N) {
    if (N == 0) return 1;
    return N * getFactorial( N-1 );
}

int main()
{
    int n = 7;
    std::cout << getFactorial(n);
}
```

7! = 5040

Hanoja tornis



Ir trīs stieņi, uz vienu no kuriem ir ievietoti N gredzeni. Gredzeni atšķiras pēc lieluma un ir nolikti mazāki par lielākiem. Nepieciešams pārvietot piramīdu no N gredzeniem no viena stieņa uz otru par mazāko kustību skaitu. Vienmēr ir atļauts pārvietot tikai vienu gredzenu un nedrīkst ievietot lielāku gredzenu uz mazāku.

N=1: A → B

N=2: A → C, A → B, C → B

N=3: A → B, A → C, B → C, A → B, C → A, C → B, A → B

Ir iespējams pārvietot N gredzenus, spējot pārvietot N-1 gredzenus:

1. pārvietot N-1 gredzenus no sākumu uz pagaidu stieni
2. pārvietot 1 gredzenu no sākumu uz galīgu stieni
3. pārvietot N-1 gredzenus no pagaidu uz galīgu stieni

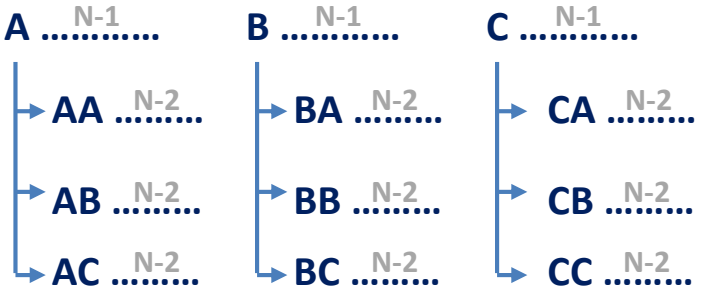
```
void Hanoi(
    int n, char from, char to, char temp
)
{
    if (n <= 0) return;
    Hanoi (n-1, from, temp, to);
    std::cout << from << "-" << to;
    Hanoi( n-1, temp, to, from);
}
```

```
Hanoi(3, 'A', 'B', 'C');
```

A->B A->C B->C A->B C->A C->B A->B

Variācijas ar atkārtojumiem

Atrast un izvadīt uz ekrāna visus vārdus, kas var salikt no burtus A, B un C un kuru garums ir N burti. Katrs burts var būt iekļauts vārdā dažas reizes.



AAA	BAA	CAA
AAB	BAB	CAB
AAC	BAC	CAC
ABA	BBA	CBA
ABB	BBB	CBB
ABC	BBC	CBC
ACA	BCA	CCA
ACB	BCB	CCB
ACC	BCC	CCC

```
void showCombinations(char Word[], int Len) {
    int len = strlen(Word);
    if (len == Len) {
        std::cout << Word <<  "\n";
        return;
    }
    Word[len + 1] = '\0';
    Word[len] = 'A'; showCombinations(Word, Len);
    Word[len] = 'B'; showCombinations(Word, Len);
    Word[len] = 'C'; showCombinations(Word, Len);
    Word[len] = '\0';
}

int main()
{
    char word[10] = "";
    showCombinations(word, 3);
}
```