

# Function overloading.

## Recursive functions

# Reja:

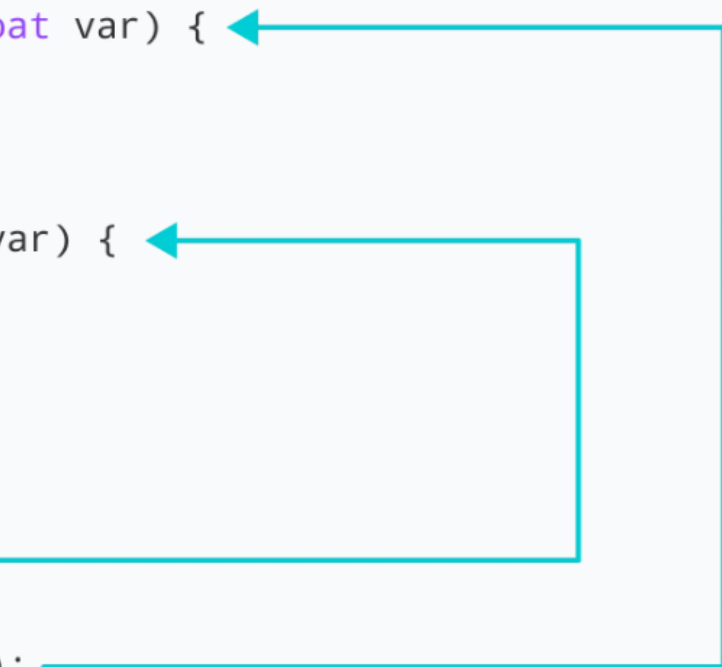
- Function overloading
- Recursive functions
- Amaliy mashqlar

# Function overloading

Dasturlashda nomlari bir xil, parametrlarining turi va soni bilan farq qiladigan funksiyalar **qayta yuklanuvchi funksiyalar** deyiladi.

## Example 1

```
float absolute(float var) {  
    // code  
}  
  
int absolute(int var) {  
    // code  
}  
  
int main() {  
    absolute(-5);  
    absolute(5.5f);  
    ...  
}
```



The diagram consists of three blue arrows. One arrow originates from the `absolute(-5);` line in the `main()` function and points to the `int absolute(int var) {` line. A second arrow originates from the `absolute(5.5f);` line in the `main()` function and points to the `float absolute(float var) {` line. A third arrow originates from the `absolute(5.5f);` line and points to the `int absolute(int var) {` line, illustrating that the same function name is used for different data types.

```
// function with float type parameter
float absolute(float var){
    if (var < 0.0)
        var = -var;
    return var;
}

// function with int type parameter
int absolute(int var) {
    if (var < 0)
        var = -var;
    return var;
}
```

```
int main() {  
  
    // call function with int type parameter  
    cout << "Absolute value of -5 = " << absolute(-5) << endl;  
  
    // call function with float type parameter  
    cout << "Absolute value of 5.5 = " << absolute(5.5f) << endl;  
    return 0;  
}
```

## Output

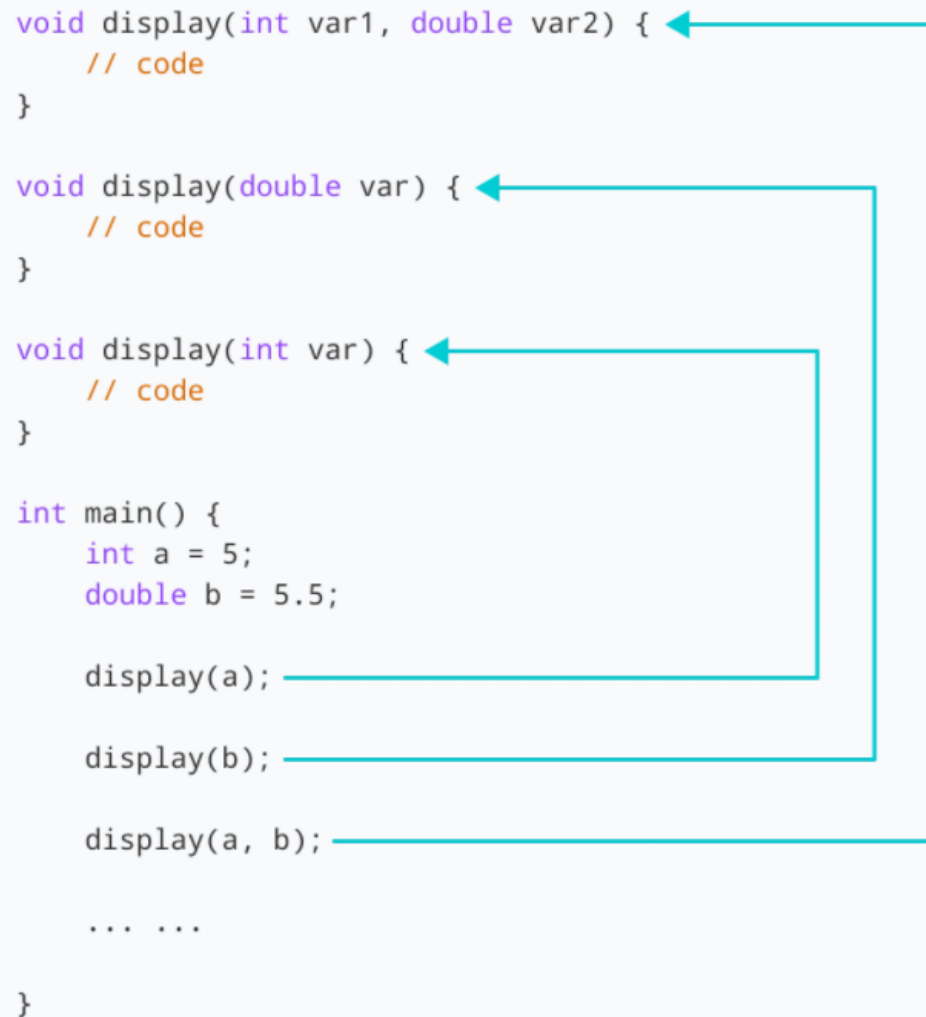
```
Absolute value of -5 = 5
```

```
Absolute value of 5.5 = 5.5
```



## Example 2

```
void display(int var1, double var2) {  
    // code  
}  
  
void display(double var) {  
    // code  
}  
  
void display(int var) {  
    // code  
}  
  
int main() {  
    int a = 5;  
    double b = 5.5;  
  
    display(a);  
    display(b);  
    display(a, b);  
  
    ... ..  
}
```



The diagram illustrates the function calls from the `main` function to the `display` functions. Three blue lines originate from the function calls in `main` and point to the corresponding function definitions:

- A line from `display(a);` points to the `display(int var)` function.
- A line from `display(b);` points to the `display(double var)` function.
- A line from `display(a, b);` points to the `display(int var1, double var2)` function.

```
// function with 2 parameters
void display(int var1, double var2) {
    cout << "Integer number: " << var1;
    cout << " and double number: " << var2 << endl;
}

// function with double type single parameter
void display(double var) {
    cout << "Double number: " << var << endl;
}

// function with int type single parameter
void display(int var) {
    cout << "Integer number: " << var << endl;
}
```

```
int main() {  
  
    int a = 5;  
    double b = 5.5;  
  
    // call function with int type parameter  
    display(a);  
  
    // call function with double type parameter  
    display(b);  
  
    // call function with 2 parameters  
    display(a, b);  
  
    return 0;  
}
```

## Output


```
Integer number: 5
```

```
Float number: 5.5
```

```
Integer number: 5 and double number: 5.5
```

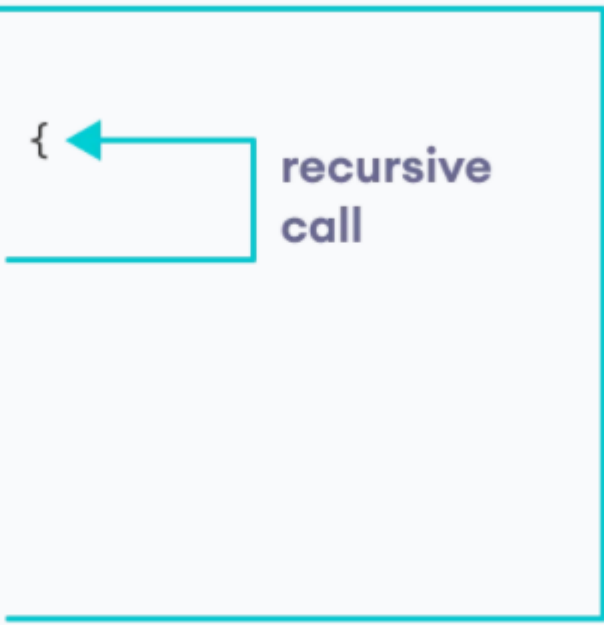
# Recursive functions

Funksiya o'ziga o'zi to'g'ridan-to'g'ri yoki qandaydir vosita orqali murojaat qilish jarayoniga **rekursiya** deyiladi va bunday funksiya **rekursiv funksiya** deb ataladi.



```
void recurse() {  
    ... ..  
    recurse();  
    ... ..  
}
```

recursive  
call



```
int main() {  
    ... ..  
    recurse();  
    ... ..  
}
```

function  
call

Har qanday to'g'ri tuzilgan rekursiya asosini ikkita shart tashkil qiladi:

- ✓ funksiyaning o'ziga o'zi murojaat qilishi;
- ✓ rekursiyaning to'xtash sharti.



```
// Factorial of n = 1*2*3*...*n

int factorial(int n) {
    if (n > 1) {
        return n * factorial(n - 1);
    } else {
        return 1;
    }
}
```

```
int main() {  
    int n, result;  
  
    cout << "Enter a non-negative number: ";  
    cin >> n;  
  
    result = factorial(n);  
    cout << "Factorial of " << n << " = " << result;  
    return 0;  
}
```

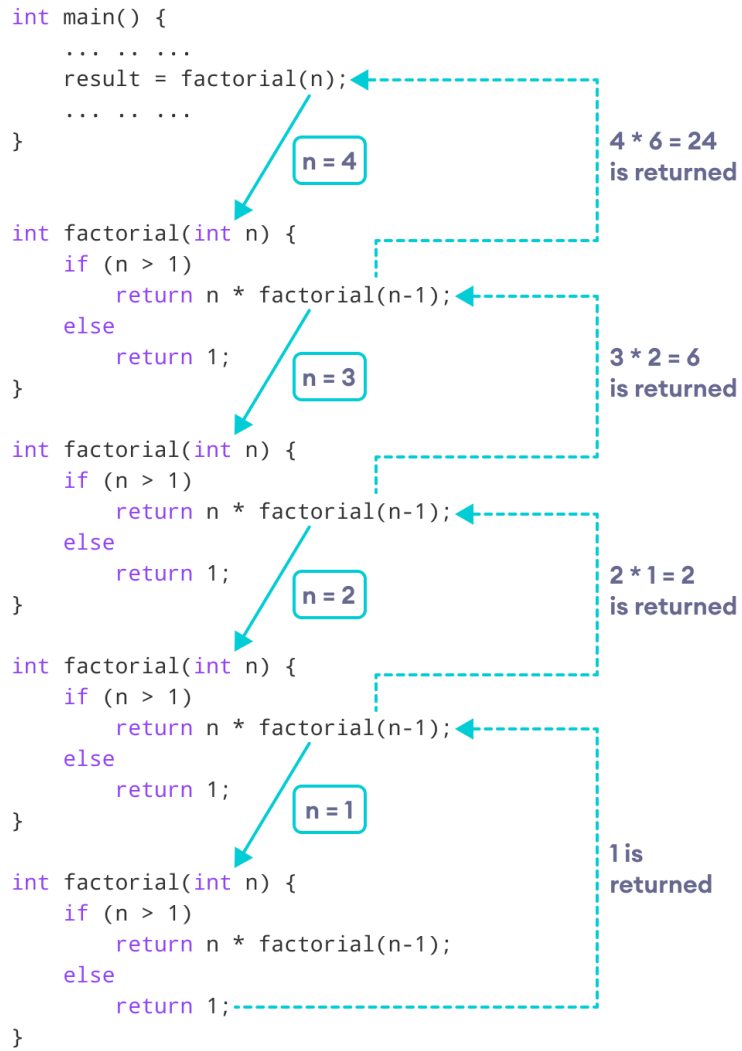
```
int main() {
    ... ..
    result = factorial(n);
    ... ..
}

int factorial(int n) {
    if (n > 1)
        return n * factorial(n-1);
    else
        return 1;
}

int factorial(int n) {
    if (n > 1)
        return n * factorial(n-1);
    else
        return 1;
}

int factorial(int n) {
    if (n > 1)
        return n * factorial(n-1);
    else
        return 1;
}

int factorial(int n) {
    if (n > 1)
        return n * factorial(n-1);
    else
        return 1;
}
```



**n = 4**

**4 \* 6 = 24**  
is returned

**n = 3**

**3 \* 2 = 6**  
is returned

**n = 2**

**2 \* 1 = 2**  
is returned

**n = 1**

**1 is**  
returned

## Output

```
Enter a non-negative number: 4  
Factorial of 4 = 24
```

# Ama1iy mashqlar

**Function overloading** yordamida quyidahi **add()** funksiyasini yozing:

```
float add(float, float);
```

```
float add(float, float, float);
```

```
string add(string, string)
```

Parametr sifatida 2 ta son berilsa ham, 3 ta son berilsa ham ularning kattasini topib, qaytaruvchi **findMax()** funksiyasini "**function overloading**" yordamida yozing.

2 ta sonni parametr sifatida qabul qilganda ularning yig'indisini, 3 ta sonni parametr sifatida qabul qilganda esa ularning ko'paytmasini hisoblab, natija sifatida qaytaruvchi **calculate()** funksiyasini yozing.



1 dan N gacha bo'lgan natural sonlarning yig'indisini rekursiv funksiya yordamida hisoblang.

Berilgan  $N$  natural sonning raqamlari yig'indisini rekursiv funksiya yordamida hisoblang.

n-Fibonachi sonini rekursiv funksiya yordamida aniqlovchi dastur tuzing.

Bunda:

$$f(0) = 1, f(1) = 1, f(n) = f(n-1)+f(n-2), n \geq 2.$$

Kompyuter tomonidan 1 dan 10 gacha oraliqdagi tasodifiy natural son generatsiya qilinadi. Ushbu tasodifiy sonni foydalanuvchi tomonidan kiritilgan son bilan solishtiruvchi rekursiv funksiya yozing va bu funksiya foydalanuvchi tasodifiy sonni to'g'ri topmagunicha rekursiv tarzda chaqirilsin.

**E`tiboringiz uchun  
rahmat!**