# Funksiya va sinf shablonlari.

Shablonlar haqida Ushbu darsda C++ dasturlash tilining asosiy hususiyatlaridan bo’lgan ***shablonlar*** bilan tanishamiz. Shablonlar yordamida umumiy funksiyalar va umumiy classlar yaratish imkoniyati mavjud.

Umumiy funksiya va umumiy classlar har xil ma’lumot toifalaridan ularni ***overload qilmasdan*** (ko’p kod yozmasdan) foydalanish imkoniyatini beradi. Ya’ni bunda biz ***har bir toifa uchun aloxida funksiya yozishimiz shart bo’lmaydi***. Shablonlar ikki xil bo’ladi:

* + ***Funksiya*** shabloni (function template)
  + ***Sinf*** shabloni (class template)

# Funksiya shablonini (function template) yaratish

Funksiya shabloni **template** kalit so’zidan foydalangan holda amalga oshiriladi.

Quyida funksiya shablonini yaratish formasi keltirilgan:

template <class *TOIFA*> ***qaytarish-tipi*** *funk-nomi* (*arg-lar*)

{

***// funksiya tanasi***

}

Bu yerda *TOIFA* funksiya tomonidan joriy holda ishlatiladigan ma’lumot tipi. Ushbu toifani kompilyator avtomatik ravishda funksiyaga kelayotgan ma’lumot tipi bilan almashtirib qo’yadi.

Bu yerda **class** va **template** lar funksiya shablonini yaratish uchun ishlatiladigan kalit so’zlardir. Lekin ba’zi hollarda **class** kalit so’zi o’rniga **typename** kalit so’zini ishlatishimiz mumkin. Quyidagi misol hohlagan tipda berilgan ikkita o’zgaruvchi o’rnini almashtirib berish uchun xizmat qiladi va bunda har bir toifa uchun alohida funksiya yozishimizga zaruriyat qolmaydi.

# Funksiya shabloniga misol

#include <iostream> using namespace std;

*// Funksiya shabloni e’lon qilinishi...*

template <class X>

void swapargs(X &a, X &b)

{

X temp; temp = a; a = b;

b = temp;

}

int main()

{

int i=10, j=20;

double x=10.1, y=23.3; char a='x', b='z';

cout << "Original i, j: " << i << ' ' << j << '\n'; cout << "Original x, y: " << x << ' ' << y << '\n'; cout << "Original a, b: " << a << ' ' << b << '\n';

swapargs(i, j); // swap funksiyasi butun toifa uchun (int) swapargs(x, y); // swap funksiyasi haqiqiy toifa uchun (float) swapargs(a, b); // swap funksiyasi simvol toifa uchun (char) cout << "Swapped i, j: " << i << ' ' << j << '\n';

cout << "Swapped x, y: " << x << ' ' << y << '\n'; cout << "Swapped a, b: " << a << ' ' << b << '\n';

return 0;

}

Dastur izohi:

# Umumiy funksiyaning boshqacha ko’rinishi

Quyidagi misolda **swapargs( )** funksiyasi boshqacharoq ko’rinishda e’lon qilingan. Ya’ni shablon birinchi satrda funksiya esa alohida satrda joylashgan.

template <class X>

void swapargs(X &a, X &b)

{

X temp; temp = a; a = b;

b = temp;

}

Lekin bu ko’rinishda birinchi va ikkinchi satr o’rniga bironta kod yozilsa xatolik beradi

template <class X> int c // ERROR

void swapargs(X &a, X &b)

{

X temp; temp = a;

a = b;

b = temp;

}

# Funksiya shablonini override (qayta yozish) qilish.

template <class X>

void swapargs(X &a, X &b)

{

X temp; temp = a; a = b;

b = temp;

cout << "swapargs funksiya shabloni chaqirildi.\n";

}

***// Bunda swapargs() funksiyasi faqatgina int tipi uchun ishlaydi.***

void swapargs(int &a, int &b)

{

int temp; temp = a; a = b;

b = temp;

cout << " int tipi uchun maxsus swapargs funksiyasi.\n";

}

int main()

{

int i=10, j=20;

double x=10.1, y=23.3; char a='x', b='z';

cout << "Original i, j: " << i << ' ' << j << '\n'; cout << "Original x, y: " << x << ' ' << y << '\n'; cout << "Original a, b: " << a << ' ' << b << '\n';

swapargs(i, j); *// calls explicitly overloaded swapargs()*

swapargs(x, y); *// calls generic swapargs()* swapargs(a, b); *// calls generic swapargs()* cout << "Swapped i, j: " << i << ' ' << j << '\n';

cout << "Swapped x, y: " << x << ' ' << y << '\n'; cout << "Swapped a, b: " << a << ' ' << b << '\n'; return 0;

}

Dastur natijasi:

*Original i, j: 10 20*

*Original x, y: 10.1 23.3 Original a, b: x z*

**int tipi uchun maxsus swapargs funksiyasi*.***

swapargs funksiya shabloni chaqirildi*.*

swapargs funksiya shabloni chaqirildi*.*

***Swapped i, j: 20 10***

***Swapped x, y: 23.3 10.1 Swapped a, b: z x***

# Funksiya shablonini Override qilish yangi usuli

template<> void swapargs<int>(int &a, int &b)

{

int temp; temp = a; a = b;

b = temp;

cout << " int tipi uchun maxsus swapargs funksiyasi.\n";

}

**Funksiya shablonini overload qilish.**

*// Oddiy funksiyalardek, funksiya shablonini ham overload qilish mumkin.*

#include <iostream> using namespace std;

*// f() funksiya shablonining birinchi turi.*

template <class X> void f(X a)

{

cout << "Inside f(X a)\n";

}

*// f() funksiya shablonining ikkinchi turi.*

template <class X, class Y> void f(X a, Y b)

{

cout << "Inside f(X a, Y b)\n";

}

int main()

{

f(10); *// calls f(X)*

f(10, 20); *// calls f(X, Y)*

return 0;

}

# Funksiya shablonining kamchiligi

* Umumiy funksiyalar funksiya overloadining o’rnini bosishi mumkin.
* Lekin bu yerda bitta kamchilik mavjud.
* Biz oddiy funksiyani overload qilganimizda, har xil ma’lumotlar tipi uchun

***funksiya tanasini har xil*** qilib yozishimiz mumkin.

* Lekin umumiy funksiyada har xil tip qabul qila olgani bilan funksiya tanasi har doim bir xil bo’ladi, chunki bitta funksiyaga murojaat bo’ladi.
* Faqatgina ***ma’lumotlar tipi*** har xil bo’la oladi.

# Umumiy sinflar (sinf shabloni)

Sinf shablonini e’lon qilishning umumiy formasi:

**template <class *TOIFA*> class *sinf\_nomi*{**

**...**

**}**

Yoki quyidagi ko’rinishda e’lon qilish mumkin

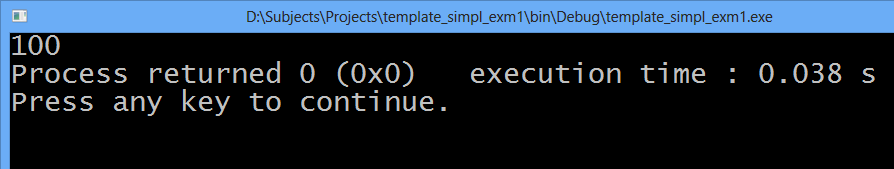
**template <class *TOIFA*> class *sinf\_nomi* {**

**...**

**}**

Sinf shablonini ishlatish

***sinf\_nomi* <tip> obyekt;**

Sinf shabloni uchun oddiy misol.

#include <iostream> using namespace std; template <class T> class mypair {

T a, b; public:

mypair (T first, T second)

{a=first; b=second;} T getmax ();

};

template <class T>

T mypair<T>::getmax ()

{

T retval;

retval = a>b? a : b; return retval;

}

int main () {

mypair <int> myobject (100, 75); cout << myobject.getmax(); return 0;

}

**Sinf shablonida ikki xil toifadan foydalanish**

#include <iostream> using namespace std;

template <class Type1, class Type2> class myclass

{

Type1 i; Type2 j; public:

myclass(Type1 a, Type2 b) { i = a; j = b; } void show() { cout << i << ' ' << j << '\n'; }

};

int main()

{

myclass<int, double> ob1(10, 0.23);

myclass<char, char \*> ob2('X', "Assalomu alaykum."); ob1.show(); *// show int, double*

ob2.show(); *// show char, char \**

return 0;

}

Dastur natijasi:

***10 0.23***

***X Assalomu alaykum.***

# Maxsuslashtirilgan sinf shabloni

template<> konstruktori maxsusashtirilgan sinf shabloni uchun ishlatiladi

template <class T> class myclass { T x;

public:

myclass(T a) {

cout << "Inside generic myclass\n"; x = a;

}

T getx() { return x; }

};

*// int toifasi uchun maxsuslashtirilgan sinf shabloni.*

template <> class myclass<int> { int x;

public:

myclass(int a) {

cout << "Inside myclass<int> specialization\n"; x = a \* a;

}

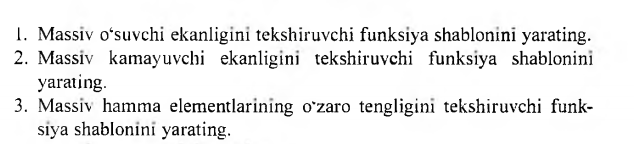
int getx() { return x; }

};

# Shablonning asosiy xususiyatlari

* ***reusable*** kod yozish imkonini beradi.
* Shablonlar yordamida framework lar yaratish mumkin
* Dastur kodi egiluvchanlik xususiyatiga ega bo’ladi.
* Turli xil tipdagi ma’lumotlar ustida ishlash uchun kod yozishda vaqtni tejash
* C++ dagi STL lar (Standard Shablon Kutubxonalar), nomidan ko’rinib turibdiki, shablonlar yordamida yaratilgan

**Topshriqlar**



4. Show\_array funksiyasi shablonidan foydalanib massiv elementlarini chiqarish.

5. Ma’lum bir ishni bajaruvchi bitta va bir nechta konstruktordan iborat sinflar yaratish va ularni qo’llashga doir misollar