10 TEMPLATES

Kalıp-Parametrik Çok Şekillilik Nedir?

Sınıflardaki fonksiyonların gövdeleri incelendiğinde, yapılan işlemler çoğu zaman, üzerinde işlem yapılan verinin tipinden bağımsızdır. Bu durumda fonksiyonun gövdesi, verinin tipi cinsinden, parametrik olarak ifade edilebilir:

```
int abs(int n) {
    return (n<0) ? -n : n;
}
long abs(long n) {
    return (n<0) ? -n : n;
}
float abs(float n) {
    return (n<0) ? -n : n;
}
```


- **C**

  Her tip için farklı adlarda fonksiyonlar.
  örnek mutlak değer fonksiyonları:
  abs(), **fabs**(), **fabsl**(), **labs**(), **cabs**(), ...

- **C++**

  Fonksiyonlara işlev yükleme bir çözüm olabilir mi?
  İşlev yüklenen fonksiyonların gövdeleri değişmiyor !
  Gövdeler tekrar ediliyor ⇒ Hata !

- **Çözüm**

  Tipi parametre kabul eden bir yapı : Template

Fonksiyon Kalıbı Tanımlamak


```
template <class T>
inline T const& max (T const& a, T const& b){
      return a<b?b:a;
 }
int main(){
  int i = 42;
  std::cout << "max(7,i): " << ::max(7,i) << std::endl;
  double f1 = 3.4;
  double f2 = -6.7;
  std::cout << "max(f1,f2): " << ::max(f1,f2) << std::endl;
  std::string s1 = "mathematics"; std::string s2 = "math";
  std::cout << "max(s1,s2): " << ::max(s1,s2) << std::endl;
}
```



```
max(4,7) // Tamam: Her iki argüman int
max(4,4.2) // Hata: ilk T int, ikinci T double

max<double>(4,4.2) // Tamam
max(static_cast<double>(4),4.2) // Tamam
```


Örnek


```
template <class T>
  void printArray(T *array,const int size){
       for(int i=0;i < size;i++)
            cout << array[i] << " " ;
   cout << endl ;
  }
```

```
int main() {
      int      a[3]={1,2,3} ;
      double   b[5]={1.1,2.2,3.3,4.4,5.5} ;
      char     c[7]={'a', 'b', 'c', 'd', 'e' , 'f', 'g'} ;

      printArray(a,3)      ;
      printArray(b,5)      ;
      printArray(c,7)      ;
  return 0 ;
}
```



```
void printArray(int *array,cont int size){
        for(int i=0;i < size;i++)
            cout << array[i] << " " ;
   cout << endl ;
  }
void printArray(char *array,cont int size){
        for(int i=0;i < size;i++)
            cout << array[i] << "" ;
   cout << endl ;
  }
```


template'in İşleyişi

Gerçekte derleyici template ile verilmiş fonksiyon gövdesi için herhangi bir kod üretmez. Çünkü template ile bazı verilerin tipi parametrik olarak ifade edilmiştir. Verinin tipi ancak bu fonksiyona ilişkin bir çağrı olduğunda ortaya çıkacaktır. Derleyici her farklı tip için yeni bir fonksiyon oluşturacaktır. template yeni fonksiyonun verinin tipine bağlı olarak nasıl oluşturulacağını tanımlamaktadır.

```
int intVar1 = 5;

cout << "abs(" << int << ")=" << abs(intVar1);
```


template'in İşleyişi

- Programı ister template yapısı ile oluşturalım ister de template yapısı olmaksızın oluşturalım, programın bellekte kaplayacağı alan değişmeyecektir.
- Değişen, kaynak kodun boyu olacaktır. template yapısı kullanılarak oluşturulan programın kaynak kodu, daha anlaşılır ve hata denetimi daha yüksek olacaktır. Çünkü template yapısı kullanıldığında değişiklik sadece tek bir fonksiyon gövdesinde yapılacaktır.

Template Parametresi bir Nesne Olabilir

```
class TComplex {      /* A class to define complex numbers */
   float real,imag;
  public:
        :   // other member functions
   bool operator>(const TComplex&) const ;
};
bool TComplex::operator>(const TComplex& z) const {
   float norm1 = real * real + imag * imag;
   float norm2 = z.real * z.real + z.imag * z.imag;
   if (norm1 > norm2) return true;
     else return false;
}
```



```
template <class T>
const T & max(const T &v1, const T & v2)
{
   if (v1 > v2) return v1;
     else       return v2;
}
int main(){
   int i1=5, i2= -3;
   char c1='D', c2='N';
   float f1=3.05, f2=12.47;
   TComplex z1(1.4,0.6), z2(4.6,-3.8);
   cout << ::max(i1,i2) << endl;
   cout << ::max(c1,c2) << endl;
   cout << ::max(f1,f2) << endl;
   cout << ::max(z1,z2) << endl;
}
```

Çoklu template Parametreli Argümanlar

Templates 10

```
template <class atype>
int find(const atype *array, atype value, int size) {
   for(int j=0; j<size; j++)
      if(array[j]==value) return j;
   return -1;
}

char chrArr[] = {'a', 'c', 'f', 's', 'u', 'z'};    // array
char ch = 'f'; // value to find
int intArr[] = {1, 3, 5, 9, 11, 13};
int in = 6;
double dubArr[] = {1.0, 3.0, 5.0, 9.0, 11.0, 13.0};
double db = 4.0;
```


Templates 10

```
int main()
{
   cout << "\n 'f' in chrArray: index=" << find(chrArr, ch, 6);
   cout << "\n 6 in intArray: index=" << find(intArr, in, 6);
   cout << "\n 4 in dubArray: index=" << find(dubArr, db, 6);
}
```


Örnek

Templates 10

```
template <class T>
void swap(T& x, T& y) {
      T temp ;
      temp = x ;
      x = y ;
      y = temp ;
}
char str1[100], str2[100] ;
int i,j ;
TComplex c1,c2;
swap( i , j ) ;
swap( c1 , c2 ) ;
swap( str1[50] , str2[50] ) ;
swap( i , str[25] ) ;
swap( str1 , str2 ) ;
```

hata

Templates 10

```
void swap(char* x, char* y) {
      int max_len ;
      max_len = (strlen(s1)>=strlen(s2)) ? strlen(s1):strlen(s2);
      char* temp = new char[max_len+1];
      strcpy(temp,s1);
       strcpy(s1,s2);
      strcpy(s2,temp);
      delete []temp;
}
```


Tipsiz Template Parametreleri

Templates 10

```
template <typename T, int VAL>
T addValue (T const& x) {
  return x + VAL;
}
template <double VAT>
double process (double v) {
      return v * VAT;
}
```

double'a izin verilmez

İşlev Yüklemede Eşleme Önceliği

- Kalıp dışında aynı imzaya sahip fonksiyon
- Kalıp ile tanımlanmış aynı imzaya sahip fonksiyon

Templates 10

```
char str1[100], str2[100] ;
...
swap( str1 , str2 ) ;
```

```
template <class T>
void swap(T& x, T& y) {
      T temp ;
      temp = x ;
      x = y ;
      y = temp ;
}
```

```
void swap(char* x, char* y) {
   ...
}
```

Çoklu template Parametreli Yapılar

Templates 10

►Template parametre sayısı birden fazla olabilir:

```
template <class atype, class btype>
 btype find(const atype* array, atype value, btype size) {
      for (btype j=0; j<size; j++)
          if(array[j]==value) return j;
      return (btype)-1;
  }
```

►Bu durumda, derleyici sadece farklı dizi tipleri için değil aynı zamanda aranan elemanın farklı tipte olması durumunda da farklı bir kod üretecektir:

```
short int result,si=100;
int invalue=5;
result = find(intArr, invalue,si) ;
```


Sınıf template Yapısı

Templates 10

```
class Stack {
    int st[MAX];          // array of ints
    int top;              // index number of top of stack
  public:
    Stack();              // constructor
    void push(int var);   // takes int as argument
    int pop();            // returns int value
};

class LongStack {
    long st[MAX];         // array of longs
    int top;              // index number of top of stack
  public:
    LongStack();          // constructor
    void push(long var); // takes long as argument
    long pop();           // returns long value
};
```


Templates 10

```
template <class Type,int maxSize>
class Stack{
    Type st[maxSize];       // stack: array of any type
    int top;                // number of top of stack
  public:
    Stack(){top = 0;}       // constructor
    void push(Type );       // put number on stack
    Type pop();             // take number off stack
};
template<class Type>
void Stack<Type>::push(Type var) // put number on stack
{
  if(top > maxSize-1)       // if stack full,
    throw "Stack is full!"; // throw exception
  st[top++] = var;
}
```


Templates 10

```
template<class Type>
Type Stack<Type>::pop() {      // take number off stack
  if(top <= 0)                 // if stack empty,
    throw "Stack is empty!";   // throw exception
  return st[--top];
}
```

```
int main()
{
  // s1 is object of class Stack<float>
  Stack<float,20> s1;
  // push 2 floats, pop 2 floats
  try{
    s1.push(1111.1);
    s1.push(2222.2);
    cout << "1: " << s1.pop() << endl;
    cout << "2: " << s1.pop() << endl;
  }
  // exception handler
  catch(const char * msg) {
    cout << msg << endl;
  }
```

```
  // s2 is object of class Stack<long>
  Stack<long,10> s2;
  // push 2 longs, pop 2 longs
  try{
    s2.push(123123123L);
    s2.push(234234234L);
    cout << "1: " << s2.pop() << endl;
    cout << "2: " << s2.pop() << endl;
  }
  // exception handler
  catch(const char * msg) {
    cout << msg << endl;
  }
}   // End of program
```


Sınıf template Yapısının Farkı

Templates 10

►Template fonksiyonları için template parametresinin ne olacağını **derleyici** çağrı yapılan fonksiyon için imzaya bakarak karar verir.

►Template sınıflar için tanımlandığında template parametresini **programcı** verir.

```
Stack<float,20> s1;
Stack<long,10> s2;
```

```
swap( c1 , c2 ) ;
swap( str1[50] , str2[50] ) ;
```


Neler Template Parametresi Olamaz?

Templates 10

```
template <typename T> class List { … };
typedef struct { double x, y, z; } Point;
typedef enum { red, green, blue } *ColorPtr;      enum types
int main() {
    struct Association { int* p; int* q; };       local types
    List<Assocation*> error1;
    List<ColorPtr> error2;
    List<Point>;
}
```

Static Polymorphism×Dynamic Polymorphism

- Run-time Polymorphism vs. Compile-time Polymorphism
- Run-time Polymorphism:
  - Inheritance & virtual functions
- Compile-time Polymorphism
  - templates


```
class GeoObj {
 public:
   virtual void draw() const = 0;
   virtual Coord center_of_gravity() const = 0;
};
class Circle : public GeoObj {
 public:
   virtual void draw() const;
   virtual Coord center_of_gravity() const;
   …
};
class Line : public GeoObj {
 public:
   virtual void draw() const;
   virtual Coord center_of_gravity() const;
   …
};
```


Run-time Polymorphism

```
void myDraw (GeoObj const& obj)
{
   obj.draw();
}

int main()
{
   Line l;
   Circle c, c1, c2;

   myDraw(l);          // myDraw(GeoObj&) => Line::draw()
   myDraw(c);          // myDraw(GeoObj&) => Circle::draw()
```



```
// concrete geometric object class Circle
// - not derived from any class
class Circle {
 public:
   void draw() const;
   Coord center_of_gravity() const;
   …
};
// concrete geometric object class Line
// - not derived from any class
class Line {
 public:
   void draw() const;
   Coord center_of_gravity() const;
   …
};
```


Compile-time Polymorphism

```
template <typename GeoObj>
void myDraw (GeoObj const& obj)
{
   obj.draw();
}

int main()
{
   Line l;
   Circle c, c1, c2;

   myDraw(l);   // myDraw<Line>(GeoObj&)=>Line::draw()
   myDraw(c);   // myDraw<Circle>(GeoObj&)=>Circle::draw()
```


Advantages & Disadvantages

Dynamic polymorphism in C++:

- Heterogeneous collections are handled elegantly.
- The executable code size is potentially smaller (because only one polymorphic function is needed, whereas distinct template instances must be generated to handle different types).
- Code can be entirely compiled; hence no implementation source must be published (distributing template libraries usually requires distribution of the source code of the template implementations).

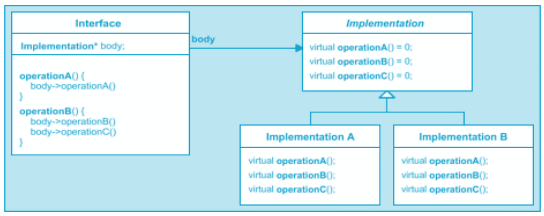
Advantages & Disadvantages

In contrast, the following can be said about static polymorphism in C++:

- Collections of built-in types are easily implemented. More generally, the interface commonality need not be expressed through a common base class.
- Generated code is potentially faster (because no indirection through pointers is needed a priori and nonvirtual functions can be inlined much more often)
- Concrete types that provide only partial interfaces can still be used if only that part ends up being exercised by the application.

New Approaches–Design Patterns

- A Design Pattern
  - "Bridge Pattern"
- Inheritance based implementation



New Approaches–Design Patterns

- Implementation with template