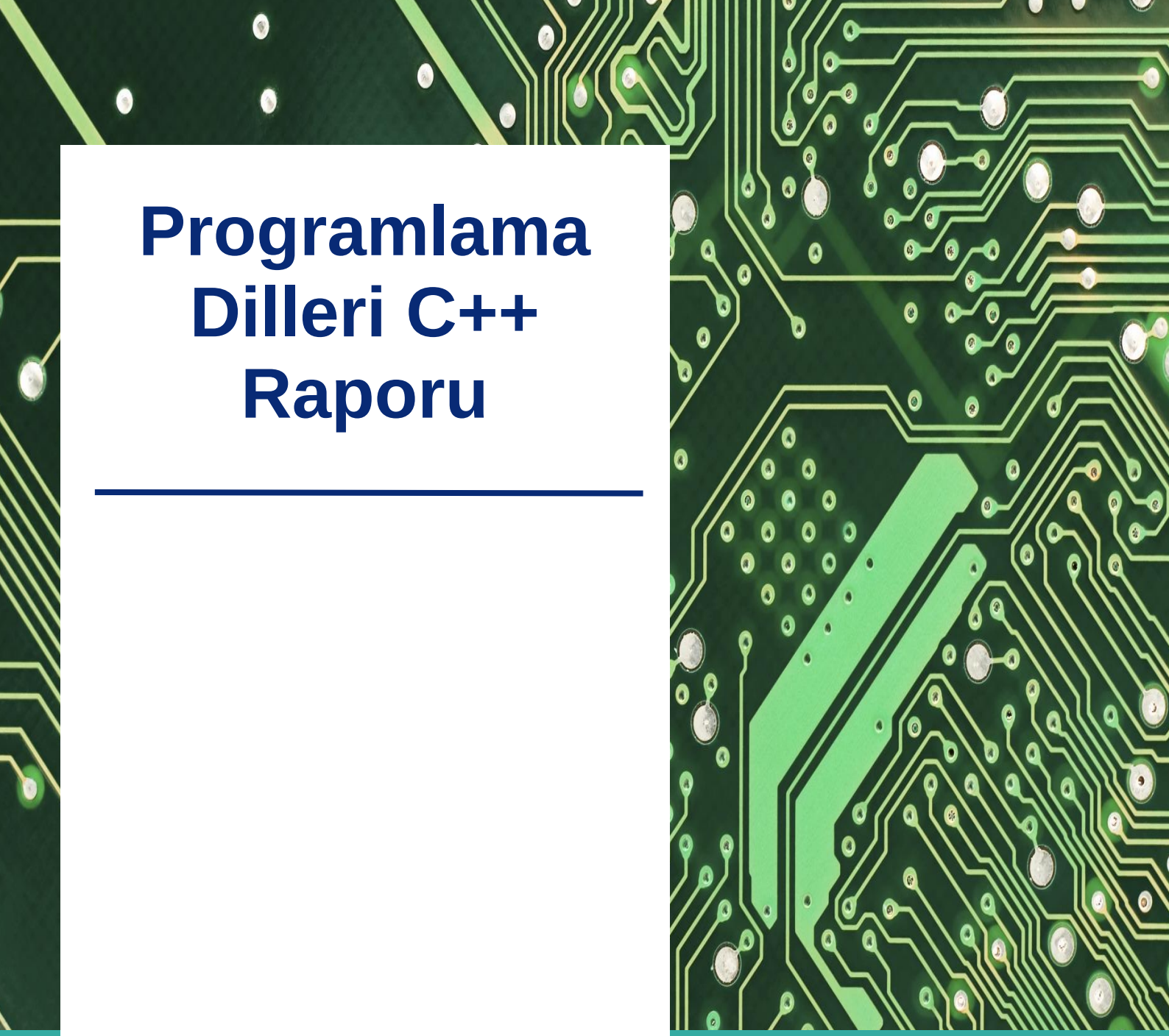

Programlama Dilleri C++ Raporu

C++ Tarihçesi ve Amaçları

C ++ programlama dilinin geçmişi Bjarne Stroustrup'un doktorası için çalıştığı 1979 yıllarına kadar uzanıyor. Stroustrup'un birlikte çalışma fırsatına sahip olduğu dillerden biri “Simula” adında bir dildi. Bir varyant olan Simula 67, nesne yönelimli programlama paradigmasını destekleyen ilk dil olarak kabul edilmiştir ve Stroustrup, bu paradigmanın yazılım geliştirme için çok yararlı olduğunu öne sürmüştür. Ancak, Simula dili pratik kullanım için çok yavaştı ve kısa süre sonra, isminden de anlaşılacağı gibi C dilinin bir üst kümesi olan "C with Classes" üzerinde çalışmaya başladı. Hedefi ise, hızdan veya düşük seviyeli işlevsellikten ödün vermeden, taşınabilirliği için saygı duyulan bir dil olan, C diline nesne yönelimli programlama eklemekti. Bu dil C dilinin özelliklerine ek olarak ; sınıfları, temel kalıtımı, satır içini, varsayılan işlev argümanlarını ve güçlü yazım denetimini içeriyordu.

1982 yıllarına kadar süren geliştirme aşamasında “C with Classes” olarak adlandırılırken, 1983 yılında C++ olarak değiştirildi. Aslında C dilinin ++ işleci, Stroustrup'un dili nasıl gördüğü hakkında fikir veren bir işleçti. Bu süre içerisinde çeşitli yeni özellikler eklenmiştir ve en önemlileri ise sanat işlevleri, işlev aşırı yüklenmesi ve & sembolüyle olan referanslar, const anahtar sözcüğü ve son olarak iki eğik çizgi (BCPL dilinden alınan bir özellik olan) tek satırlı yorumlardır.

1985 yılında ise C++ programlama dil referansı yayınlanmıştı. Artı olarak, aynı yıl içerisinde C++ ticari bir ürün olarak uygulandı ama dil resmi olarak standartlaştırılmadığı için, kitap çok önemli bir referans haline geldi. Bu dil 1989 yılında tekrar güncellendi ve 1990 yılında, açıklamalı C++ referans kılavuzu yayınlandı. Aynı yıl, Borland'ın Turbo C++ derleyicisi ticari bir ürün olarak piyasaya sürüldü.Turbo C++ , C++ diline gelişimi üzerinde önemli bir etkiye sahip olacak çok sayıda ek kitaplık ekledi.Turbo C++ programlama dilinin son ve kesin olan sürümü 2006 yılından olmasına rağmen, derleyici günümüzde hala yaygın olarak kullanılmaktadır.1998’ de C++ standartlar komitesi C++ ISO / IEC 14882 için ilk uluslararası standardı yayınladı ve gayri resmi olarak C++ 98 adıyla anıldı. Referans kılavuzunun ise standart gelişimi üzerinde büyük etkisi olduğu söylenmiştir. Artı olarak, kavramsal gelişimine 1979'da başlayan Standart şablon kütüphanesi de dahil edilmiştir.2003 yılında, komite, 1998 standardıyla bildirilen birçok soruna cevap vermiş ve buna göre dili revize etmiştir ve C++ 03 olarak adlandırmıştır.

2005 yılında da, eklenmesi planlanan çeşitli özelliklerin detaylı teknik raporu TR1 adıyla yayınlanmıştır. Yeni bir standart, gayri resmi olarak, C++ 0x olarak adlanırılıp, ilk on yılın bitiminden önce yayınlanması beklenirken, ironik bir şekilde, yeni standart 2011 ortasına kadar yayınlanmayacaktı.

O süre zarfında ise çeşitli teknik raporlar yayınlanıp, bazı derleyiciler tarafından da, özellikler için, deneysel destekler ekleniyordu.2011'in ortalarında, C++ 11 olarak adlandırılarak, C++ tamamlanmıştı. Boost kütüphane projesi yeni standart üzerinde önemli bir etki yaratarak, bazı modülleri doğrudan ilgili olan Boost kütüphanelerinde türetilmesini sağladı.

C++ Hedef Kitlesi ve Kullanım Alanları

C++ Windows Mobile için uygulama geliştirme, Masaüstü uygulama geliştirme, Yazılım geliştirmeleri, Video oyun geliştirme, İşletim sistemi yazılımı, Arama motoru geliştirme ile ilgilenen kişilere hitap eder durumdadır. C++ dili, C dilinin kullanıldığı her alanda kullanılabilir Sözcük işlemciler bu dil ile yazılabilir. Haberleşme programları, veri tabanları, dosya ve dizi işlem düzenleyiciler, kullanıcı ara yüzleri, elektronik çalışma tabloları bu dil le ile yazılabilir. Dilin kullanım alanları bunlarla sınırlı değildir. Programcı hayal gücünü kullanarak her işi yapabilir. C++ dilinin orta seviyeli bir dil olmasından dolayı diğer yüksek seviyeli programlama dillerinden gerekli optimizasyon yapıldığında daha performanslı olduğu söylenebilir. Ayrıca NYP desteği sayesinde modern programlamaya imkan vermesinden dolayı Sürücü yazılımları, Oyun, Görüntü işleme, İşletim sistemleri gibi hızın önemli olduğu yerlerde kullanılır. OpenCV görüntü işleme kütüphanesi ilk olarak C ile geliştirilmiş daha sonra C++ ile geliştirilmeye devam etmektedir. Unity ve bir çok oyun motoru C++ ile geliştirilmiştir. TensorFlow, Apache MXNet gibi Yapay Zeka araçları da C++ ile geliştirilmiştir. Chrome tarayıcısı ve Chrome, Node.js V8 JavaScript moturu C++ ile geliştirilmiştir. Benzer şekilde bir çok programlama diline ait derleyici yine C++ ile geliştirilmiştir.

C++ Desteklediği Paradigmalar

Programlama paradigması, bir programlama dilinin desteklediği bir metodoloji veya programlama yoludur. C++ şu paradigmaları destekler;

Bildirimli(Declarative)

Ne yapılacağını içeren, nasıl yapılacağının mantığını pek de içermeyen paradigmadır. Dekleratif programlamada bir durum tanımlanır. Bu tanıma göre yorumlayıcı ya da derleyici bir çözüm üretir. Böyle bir paradigma, kişinin kendi kodunu yazmaktan kaynaklanan istenmeyen yan etkileri önlemek için kullanılabilir. Örnek verecek olursak bir dizi oluşturuyoruz ve baştaki elemanı sona almak istiyoruz. Bu yaklaşımda programa diziye sona al komutunu vererek nasıl yaptığı ile ilgilenmiyoruz.

```
int values[4] = { 8, 23, 2, 4 };
RotateIndices(values);
```

Fonksiyonel(Functional)

Fonksiyonel programlama, matematiksel denklemler ve fonksiyonlar açısından problemleri ifade etmeye çalışan bir deklaratif programlamanın alt kümesidir. Fonksiyonel programlama, yalnızca fonksiyonların kullanılmasıyla yazılmış programlardır. Fonksiyonel programların tipik özellikleri: Atama deyimi bulunmaz. Değişkenlerin değeri bir kere verildi mi, bir daha değişmez. Yan etkiler yoktur. Bir fonksiyonu çağırmak kendi sonucunu hesaplamaktan başka bir etki üretmez. C++ ‘ta şu şekilde bir tanımlama yaparak bu paradigmayı desteklediğini görebiliriz.

```
void quickSort(vector<int> & a,int first,int last) {……}
```

Jeneric(Generic)

Jenerik programlama, algoritma kullanıldığında belirtilecek türler açısından iskelet algoritmaları yazmaya odaklanır, böylece katı güçlü yazma kurallarından kaçınmak isteyen programcılara biraz esneklik sağlar. İyi uygulandığında çok güçlü bir paradigma olabilir. Jenerikler C++ ‘ta Template yapısıyla uygulanabilir. Template basit ve güçlü bir araçtır. Ana fikir veri türünü bir parametre olarak almak ve böylelikle farklı veri türleri için aynı kodun tekrarlanmasını engellemek. Tanımlamasına bakacak olursak;

```
Template <typename T>
    T Max( T x, T y){
            Return (x>y) ? x: y;
  }
```

Burada çağrımı hangi veri türüyle yaparsak yapalım T türü bu veri türünü yakalayacak ve işlemi o veri türüne göre gerçekleştirecek.

Zorunlu(Imperative)

Zorunlu diller, programcıların görevi sadece belirtmeden bilgisayara talimatlar içeren talimatlar listesi vermesine olanak tanır. Deklaratif programlamanın tersi olduğu düşünülebilir. Yani burada programın yapmasını istediğimiz işi nasıl yapacağını belirtmek zorundayız. Örneğin C++ 'ta bir dizideki ilk elamanı sona almak için bu işlemi nasıl yapması gerektiğini şu satırlarla belirtebiliriz;

```
int values[4] = { 8, 23, 2, 4 };
int temp = values[0];
for (int i = 0; i < 3; ++i)
values[i] = values[i + 1];
values[3] = temp;
```

Görüldüğü üzere adım adım kaydırma işlemi yaptırdık ve ilk elemanı sona eklettik.

Yapısal(Structured)

Yapılandırılmış programlama dilleri, bir dile, ifadelerin yürütüldüğü sıra üzerinde sezgisel kontrol gibi bir tür kayda değer bir yapı sağlamayı amaçlamaktadır (X daha sonra Y yaparsa, Z yaparsa, Y Z iken X yapın). Bu tür diller genellikle C ve C ++ 'da go to ifadesi tarafından sağlanan "sıçramalar" ı kullanmaz. C++ 'ta yapı kavramında Döngüler, Koşullar, Fonksiyonlar, Prosedürler gibi kavramları sunduğu için Yapısal Paradigmayı destekler durumundadır.

Yordamsal(Procedural)

C++ programlama dilinin desteklediği bu yöntemde öncelikle gerçeklenmek istenen sistemin yapması gereken iş belirlenir. Büyük boyutlu ve karmaşık işler, daha küçük ve basit işlevlere (fonksiyon)bölünerek gerçeklenirler. Örnek olarak;

```
int f(int a){return a*a;}
int main(){
f(5);
 return 0;
}
```

Program main fonksiyonunda f(5); ifadesine geldiğinde bulunduğu yerden f() fonksiyonuna geçerek buradaki işlemi gerçekleştirip main fonksiyonundaki ifadeye bu işlemin sonucu(integer dönüş tipi olduğu için void türünde olsaydı program değer döndürmeden dönecekti) döndürür ve devam eder.

Nesneye-Yönelik(Object-Oriented)

Nesneye Yönelik programlama (genellikle OOP olarak kısaltılır), gerçek dünyada nesneleri modellemek için kullanılan programları "nesneler" terimiyle ifade eden yapılandırılmış bir programlama alt kümesidir. Böyle bir paradigma, kodun hayranlık uyandırıcı bir şekilde yeniden kullanılmasına izin verir ve anlaşılması kolaydır. Örnek olarak C++ ‘ta bir işçi sınıfı yaratmak istersek şu satırlarla gerçekleyebiliriz;

```
Class employee {
Public:
 employee();
~employee();
int salary();
void get salary(int salary){ this->salary=salary; }
.
.
.
 }
```

Bu sınıfa işçi ile ilgili metodları tanımlayarak işçinin yaptığı işlemleri dijital ortamlarda gerçekleştirerek saklayabiliriz.

C++ Bellek Yönetimi

C++ dinamik bellek kullanımı yönetimine izin veren bir dildir. Bu dilde, dizi tanımlaması gibi yapılan bazı işlemler belleğin otomatik olarak tahsis edilmesini sağlarken, bazı işlemler bu olanağı sağlamaz. Bu durumda, programın çalışması esnasında gereksinim duyduğumuzda bellek tahsis etmek zorundayız. Bu işlemi C++' ta dinamik bellek kullanımı yöntemi adı ile gerçekleştirebiliriz. C++ ile bu işlemler new ve delete anahtar kelimeleri ile yapılır. Bu anahtar kelimelerin dönüş değerleri yoktur. Dinamik olarak bellekten yer ayırma yapmak istediğimizde new anahtar kelimesini kullanarak işletim sistemine yer ayırma yapmak istediğimizi belirtiyoruz. Eğer ram'de yer varsa işletim sistemi bize ram'den adres döndürür. C++ Dinamik bellek kullanımını desteklese de Garbage Collection' gibi bir hizmeti yoktur. Yani Dinamik olarak alınan her alanı delete anahtar kelimesi ile işletim sistemine geri vermek Programcının görevidir. Kullanımı için bir örnek verecek olursak;

```
#include <iostream>
using namespace        std;
int main(){
        int *x=new int(5); // Dinamik Bellek  Çağrısı
        delete x; // Alanın geri verilmesi
return 0;
}
```

Yukarıdaki kodda new komutu ile yer ayırma işlemini gerçekleştiriyoruz ve ardından bu alanı delete komutu ile işletim sistemine bu alandaki işimizin bittiğini belirtiyoruz.

C++ Değişken Kapsamları

Bir sınıf, fonksiyon veya değişken gibi bir program öğesi bildirdiğinizde, adı yalnızca belirlendiği alanda "görülebilir". Bir adın görünür olduğu bağlama kapsamı denir . Örneğin, bir fonksiyon içinde bir x değişkeni bildirirseniz, yalnızca bu fonksiyon gövdesi içinde görünür. Bu yerel(local) kapsamdır . Ayrıca, C++ değişkenler farklı kapsamlarda oldukları sürece Programınızda aynı ada sahip başka değişkenler yaratmanıza olanak sağlar, Bu şekilde Tek Tanımlama Kuralını ihlal etmezler ve herhangi bir hata ortaya çıkarmazlar.

C++ için altı çeşit kapsam vardır:

Genel kapsam (Global Scope)

Genel ad, herhangi bir sınıf, işlev veya ad alanının dışında bildirilen addır. Genel adların kapsamı, bildirim noktasından bildirildikleri dosyanın sonuna kadar uzanır. Örnek olarak;

```
#include <iostream>
int a; // Global değişken
int main(){
.
.
.
 return 0;}
```

a değişkeni burada global değişkendir ve tüm alanlarda tanınır durumdadır.

Ad alanı kapsamı (Namespace Scope)

Herhangi bir sınıf veya fonksiyon bloğunun dışında bir ad alanı içinde bildirilen bir ad, bildirim noktasından ad alanının sonuna kadar görünür. Bir ad alanı farklı dosyalar arasında birden fazla blokta tanımlanabilir.Örnek olarak;

```
namespace ContosoData //ad alanı
{class ObjectManager // sınıf
{public:void DoSomething() {}// public sınıf fonksiyonu
};
void Func(ObjectManager) {}// ad alanı fonksiyonu
}
```

Yerel kapsam(Local Scope)

Parametre adları da dahil olmak üzere bir fonksiyon veya lambda içinde bildirilen bir adın yerel kapsamı vardır. Genellikle "locals" olarak adlandırılırlar. Sadece bildirim noktalarından işlevin veya lambda gövdesinin sonuna kadar görülebilirler. Örnek olarak;

```
void f(){
int a; // f fonksiyonunda tanımlı yerel değişken
}
int f1(){
a=5;// tanımsız
}
```

Hatalı bir ifadedir a değişkeni sadece f() fonksiyonunda tanımlıdır. f1() fonksiyonunda görünür değildir.

Sınıf kapsamı(Class Scope)

Sınıf üyelerinin isimleri, deklarasyon noktasına bakılmaksızın sınıf tanımı boyunca uzanan sınıf kapsamına sahiptir. Sınınf üyesi erişilebilirliği genel(Public) , özel(Private) ve korunan (Protected) anahtar kelimeler tarafından daha da kontrol edilir durumdadır. Herkese açık veya korunan üyelere yalnızca üye seçimi işleçleri (. Veya ->) veya işaretçi-üye işleçleri (. * Veya -> *) kullanılarak erişilebilir . Örnek olarak;

```
Class a{
public:
int b; // public sınıf değişkeni
};
İnt main(){
a k; // nesne oluşturma
k.b=5; // nesnenin özelliğine değer atama
return 0;}
```

a sınıfında tanımladığımız public olan b değişkenine main fonksiyonunda yarattığımız nesne üzerinden erişebildik ve atama işlemi yaptık.

İfade kapsamı(Statement Scope)

for , if , while veya switch ifadesinde bildirilen adlar , ifade bloğunun sonuna kadar görünür. Örnek olarak;

```
for (int i=0; i<5;i++){
}
std::cout<<i<<std::endl; // i değişkeni tanımsız
```

Hatalıdır çünkü i değişkeni for ifadesinin kapsamında yaratıldı ve döngü bittiğinde yok edildi.

Fonksiyon kapsamı(Function Scope)

Bir fonksiyon içinde bildirilen bir etiket (ve yalnızca bir etiket), o işlevin her yerinde, tüm iç içe bloklarda, kendi bildiriminden önce ve sonra kapsam dahilindedir.

```
void f() { {
goto label; // label tanımlı
label:; }
 goto label; // label tanımlı
}
 void g() { goto label; //label tanımsız }
```

label f() fonksiyonu içinde tanımlı olduğu için önce veya sonra ulaşılır durumdayken g() fonksiyonu kapsam dışında olduğu için label tanımsızdır.

C++ Tip Sistemi

Tip Sistemi(Type System)

Bir tür sistemi , bir dilin farklı değişken türlerinin uyması gereken kuralları ifade eder. Bazı dillerin (çoğu montaj dili dahil) türü yoktur ve bu nedenle bu bölüm onlar için geçerli değildir. Ancak, çoğu dilin (C ++ dahil) türleri olduğundan, bu bilgiler önemlidir. C++' da tip kavramı çok önemlidir. Her değişken, fonksiyon vs derlenebilmesi için bir türe sahip olmalıdır. Yani her nesnenin bir türü vardır ve bu tür asla değişmez. Kodunuz'da bir değişken bildirdiğinizde, türünü açıkça belirtmeniz veya türü auto anahtar kelimesi ile instructor' a bildirmeniz gereklidir. Kodunuzda bir fonksiyon bildirdiğinizde, her bağımsız değişkenin türünü ve dönüş değerini belirtmeniz veya fonksiyon bir değer döndürmeyecekse void olduğunu belirtmeniz gerekmektedir.Bu kuralları dikkate alarak C++ güçlü(strong) bir dildir diyebiliriz.

Tip Kontrolü(Type Checking)

C++ her değişkenin tipinin önceden belirtilmesini ister. Yani string bir değer tanımlıyorken ifadenin başında string, tamsayı tanımlıyorken int, double, float gibi tipleri yazıyoruz. Bu nedenle değişken tipleri program henüz çalışmıyorken bile bu tiplerin neler olduğunu biliyor. Bu da program henüz çalışmıyorken bile bir hata yapmışsanız sizi uyarır ve hatayı düzeltmenizi bekler. Bu yüzden C++ Durağan Tip Kontrolüne(Static Type Checking) sahip bir dildir. Bunun amacı donanımı optimize etmek olduğu için bu şekilde çalışmak zorundadır.