DİZGİLER

Dizgi Tanımlama

Dizgiler (string) metinler olup bellekte son karakteri sıfır olan karakter dizileri olarak tutulurlar. Dizgiler, boş bir karakter `'\0'` ile sonlanan karakter dizisi olarak saklanır. Metinleri tutan değişkenler uzantısı olduğu C dilinin belirlediği şekilde tanımlanabilir;

```
    char metin[]="Merhaba";  /* değişkenine metin değişmezi(string literal) ile
                                ilk değer verildi */
    char metin2[]={'M','e','r','h','a','b','a','\0'};
    char* metinGostericisi="Merhaba";
```

Aşağıda C dilinden geldiği şekliyle metinlerin tanımlanıp kullandığı bir örnek program verilmiştir;

```
#include <iostream>
using namespace std;
int main() {
    char metin[] = "Merhaba";
    cout << metin << endl;
}
```

C dilinde karakter dizileri olarak kullanılan dizgiler (string) yerine C++ dilinde konteyner şablonlarından biri olan `std::string` kullanılır. C++ dilinde dizgiler, karakter dizilerini temsil eden nesnelerdir. Standart dizgi sınıfı, metin ve diğer karakter dizgileriyle çalışır ve C dilindeki açık karakter dizileri yerine basit, güvenli ve çok yönlü bir alternatiftir. C++ dizgi sınıfı, `std` isim uzayının bir parçasıdır ve 1998'de standartlaştırılmıştır.

```
    string metin=("Merhaba");
    string metin2="Merhaba";
    string metin3; metin3="Merhaba";
```

Yukarıda verilen program `std::string` ile aşağıdaki gibi yazılabilir;

```
#include <iostream>
int main() {
    std::string str("Merhaba");
    std::cout << str;
}
```

std::string

Dizgiler belirlenen bir ayraç metni ile belirteç (token) denilen küçük parçalara `strtok()` fonksiyonu ile ayrılabilir;

```
std::string str{ "Pijamalı hasta yağız şoföre çabucak güvendi." };

vector<std::string> tokens;
for (auto i = strtok(&str[0], " "); i != NULL; i = strtok(NULL, " "))
tokens.push_back(i);
```

Bir `std::string` verilerine `const char*` olarak erişim sağlamak için `c_str()` üye yöntemi kullanılır;

```
//Göstericiler str nesnesinin altında yatan karakter dizisini gösterir;
std::string str("This is a string.");
const char* cstr = str.c_str(); // cstr göstericisi bu metni gösterir: "This is a string.\0"
const char* data = str.data();  // data göstericisi bu metni gösterir: "This is a string.\0"

//Göstericilerin str nesnesinden ömür boyu bağını çözmek için kopyasını oluşturun
std::string str("This is a string.");
std::unique_ptr<char []> cstr = std::make_unique<char[]>(str.size() + 1);
```

```
// Yukarıdaki satırın eşdeğeri: char* cstr_unsafe = new char[str.size() + 1];
std::copy(str.data(), str.data() + str.size(), cstr);
cstr[str.size()] = '\0'; // NULL karakter sonuna eklenir.
// delete[] cstr_unsafe;
std::cout << cstr.get();
```

C++17 ile birlikte `std:string` altında yatan `const char` üzerinden işlem yapmak yerine `std::string_view` üzerinden işlemler yapılır. Değiştirilemeyen dizgi verileri gerektiren işlevler için üstünlük sağlar;

```
//1)Alt metin elde etmek için kopya oluşturarak;
std::string str = "Çoook uzun biiiir metiiiiiin";
//'string::substr' returns a new string (expensive if the string is long)
std::cout << str.substr(15, 10) << '\n';

//2)Hiçbir kopya oluşturmadan;
std::string_view view = str;
// string_view::substr returns a new string_view
std::cout << view.substr(15, 10) << '\n';
```

`std::string`, `char` tipindeki elemanlarla oluşturulmuştur. `std::wstring` ise uluslararası karakterleri temsil eden `wchar_t` tipindeki elemanlarla oluşturulmuştur. Bunlar arasında çeviri yapılabilir;

```
#include <iostream>
#include <string>
#include <codecvt>
#include <locale>
int main() {
    std::locale::global(std::locale(""));
    std::wcout.imbue(std::locale());
    setlocale(LC_ALL, "Turkish");
     const wchar_t message_turkish[] = L"ıİüÜğĞiİşŞçÇöÖ"; //Türkçe Karakterler
    //std::wcout << message_turkish << std::endl;

    std::string input_str = "İlhan-this is a -string-";
    std::wstring input_wstr = L"İlhan-this is a -wide- string";
    // conversion
    std::wstring str_turned_to_wstr =
    std::wstring_convert<std::codecvt_utf8<wchar_t>>().from_bytes(input_str);
    std::wcout << str_turned_to_wstr << std::endl;

    std::string wstr_turned_to_str =
    std::wstring_convert<std::codecvt_utf8<wchar_t>>().to_bytes(input_wstr);
    std::cout << wstr_turned_to_str << std::endl;

    wstr_turned_to_str =
    std::wstring_convert<std::codecvt_utf8<wchar_t>>().to_bytes(message_turkish);
    std::cout << wstr_turned_to_str << std::endl;
}
```

İki `std::string`, `==`, `!=`, `<`, `<=`, `>` ve `>=` işleçleri kullanarak sözlükteki olarak karşılaştırılabilir;

```
#include <iostream>
#include <string>
#include <locale>
int main() {
    std::locale::global(std::locale(""));
    std::wcout.imbue(std::locale());
    setlocale(LC_ALL, "Turkish");
    std::string sozcuk1 = "İlhan";
    std::string sozcuk2 = "İlhan";
    if (sozcuk2==sozcuk1)
        std::cout << sozcuk1 << "=" << sozcuk2 << std::endl;
    std::wstring wsozcuk1 = L"İlhan";
```

```
    std::wstring wsozcuk2 = L"İlhan";
    if (wsozcuk1==wsozcuk2)
        std::wcout << wsozcuk1 << "=" << wsozcuk2 << std::endl;
}
```

Dizgenin bir bölümü bir başka metin ile `replace()` üye yöntemi ile değiştirilebilir;

```
std::string str = "Hello foo, bar and world!";
std::string alternate = "Hello foobar";
//1)
str.replace(6, 3, "bar"); //"Hello bar, bar and world!"
//2)
str.replace(str.begin() + 6, str.end(), "nobody!"); //"Hello nobody!"
//3)
str.replace(19, 5, alternate, 6, 6); //"Hello foo, bar and foobar!"
Version ≥ C++14
//4)
str.replace(19, 5, alternate, 6); //"Hello foo, bar and foobar!"
//5)
str.replace(str.begin(), str.begin() + 5, str.begin() + 6, str.begin() + 9);
//"foo foo, bar and world!"
//6)
str.replace(0, 5, 3, 'z'); //"zzz foo, bar and world!"
//7)
str.replace(str.begin() + 6, str.begin() + 9, 3, 'x'); //"Hello xxx, bar and world!"
Version ≥ C++11
//8)
str.replace(str.begin(), str.begin() + 5, { 'x', 'y', 'z' }); //"xyz foo, bar and world!"
```

Dizgenin bir kısmı `substr()` üye yöntemi ile alınabilir;

```
std::string str = "Hello foo, bar and world!";
std::string newstr = str.substr(11); // "bar and world!"
```

Dizgenin altında yatan karakter dizisine `[]` işleci ile `at()`, `front()` ve `back()` üye fonksiyonları ile erişilebilir;

```
std::string str("Hello world!");
char c1 = str[6]; // 'w'
char c2 = str.at(7); // 'o'
char c3 = str.front(); // 'H'
char c4 = str.back(); // '!'
```

Dizginin altında yatan karakter dizisi üzerinde yineleme yapılabilir;

```
std::string str = "Hello World!";
//1) Foreach
for (auto c : str)
    std::cout << c;
//2) Geleneksel
for (std::size_t i = 0; i < str.length(); ++i)
    std::cout << str[i];
```

Metinleri diğer veri tiplerine de dönüştürebiliriz;

```
std::string ten = "10";
int num1 = std::stoi(ten);
long num2 = std::stol(ten);
long long num3 = std::stoll(ten);
float num4 = std::stof(ten);
double num5 = std::stod(ten);
long double num6 = std::stold(ten);
```

Metinleri dönüştürürken sayı tabanları da kullanılabilir;

```
std::string ten = "10";
std::string ten_octal = "12";
std::string ten_hex = "0xA";
int num1 = std::stoi(ten, 0, 2); // Returns 2
int num2 = std::stoi(ten_octal, 0, 8); // Returns 10
long num3 = std::stol(ten_hex, 0, 16); // Returns 10
long num4 = std::stol(ten_hex); // Returns 0
long num5 = std::stol(ten_hex, 0, 0); // Returns 10 as it detects the leading 0x
```

Metinler `+` ve `+=` işleçleri birleştirilebilir;

```
std::string hello = "Hello";
std::string world = "world";
std::string helloworld = hello + world; // "Helloworld"
hello += world; // "Helloworld"
```

Metinler birinin sonuna diğeri gelecek şekilde `append()` üye fonksiyonu ile birleştirilebilir;

```
std::string app = "test and ";
app.append("test"); // "test and test"
```

Bir karakteri veya başka bir dizeyi bulmak için `std::string::find()` kullanabilirsiniz. İlk eşleşmenin ilk karakterinin konumunu döndürür. Eşleşme bulunamazsa, `std::string::npos` döndürür. `find_first_of()` ile karakterlerin ilk oluşumu bulunur, `find_first_not_of()` ile karakterlerin ilk yokluğunu bulunur, `find_last_of()` ile karakterlerin son oluşumu bulunur, `find_last_not_of()` ile karakterlerin son yokluğu bulunur;

```
std::string str = "Hello world!";
auto it = str.find("world");
if (it != std::string::npos)
    std::cout << "Found at position: " << it << '\n';
else
    std::cout << "Not found!\n";
```

Klavyeden Metin Okuma

Bir girdi nesnesinden dizgi okumak için en yaygın yol, C++ dininde akıştan veri çıkarma işleci (stream extraction operator) olan `>>` ile `std::cin` nesnesini kullanmaktır.

```
#include <iostream>
using namespace std;
int main() {
    string metin;
    cout << "Bir metin Giriniz: ";
    cin >> metin;
    cout<< "Girilen Metin: " << metin << endl;
}
/* Program Çalıştığında:
Bir metin Giriniz: Ilhan Ozkan
Girilen Metin: Ilhan

...Program finished with exit code 0
*/
```

Program incelendiğinde girilen ilk sözcük metin değişkenine atanmıştır. Eğer girilen tüm satırı bu metne aktarmak istersek `<string>` başlık dosyasındaki `getline()` metodunu kullanmalıyız;

```
#include <iostream>
using namespace std;
int main() {
    string metin;
    cout << "Bir metin Giriniz: ";
```

```
    getline(cin, metin);
    cout<< "Girilen Metin: " << metin << endl;
}
/* Program Çalıştığında:
Bir metin Giriniz: Ilhan Ozkan
Girilen Metin: Ilhan Ozkan

...Program finished with exit code 0
*/
```

Uluslararası karakterlerle çalışılan aşağıdaki örnek verilebilir;

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    const wchar_t message_turkish[] = L"ıIüÜğĞiİşŞçÇöÖ"; //Türkçe "Diğer Karakterler
    std::locale::global(std::locale(""));
    std::wcout.imbue(std::locale());
    setlocale(LC_ALL, "Turkish");
    wcout << message_turkish << endl;

    wchar_t message[100];
    wcout << "Tükçe Bir Metin Giriniz:";
    wcin.getline(message,100);
    wcout << "Girilen metin:" << message <<endl;

    wstring message2;
    wcout << "Tükçe Bir Başka Giriniz:";
    getline(wcin, message2);
    wcout << "Girilen metin:" << message2;
}
```

Parametre Olarak Dizgiler

Bir diziyi bir fonksiyona geçirdiğimiz gibi, C++ dilinde de dizgiler de karakter dizileri olarak fonksiyonlara geçirilebilir.

```
#include <iostream>
using namespace std;
void metniKonsolaYaz(string pMetin) {
    cout << "Parametre olarak geçirilen metin: " << pMetin << endl;
    return;
}
int main() {
    string metin = "Merhaba Ilhan Ozkan.";
    metniKonsolaYaz(metin);
}
/* Program Çalıştığında:
Parametre olarak geçirilen metin: Merhaba Ilhan Ozkan.

...Program finished with exit code 0
*/
```

Karakter Göstericileri

C++ dilinde göstericiler, adreslerin sembolik gösterimleridir. Göstericileri kullanarak dizginin ilk karakterini, aslında karakter dizisi olarak tutulan metnin ilk karakterini başlangıç adresini elde edebiliriz. Aşağıda gösterildiği gibi, verilen dizgiye göstericiler aracılığıyla erişilebilir ve yazdırılabiliriz;

```
#include <iostream>
using namespace std;
int main() {

    string s = "Merhaba Ilhan";
    char* p = &s[0];
    while (*p != '\0') {
        cout << *p;
        p++;
    }
    cout << endl;
}
/* Program çalıştığında;
Merhaba Ilhan

...Program finished with exit code 0
*/
```

Dizgiler ve Karakter Dizisi

Bir dizgi ile bir karakter dizisi arasındaki temel fark, dizgilerin `mutable` olmayan değiştirilemez nesne (immutable object), karakter dizilerinin ise boyutunun değiştirilemez olmasıdır. Aralarındaki farklar aşağıdaki tabloda verilmiştir;

Dizgi	Karakter Dizisi
Dizgiler, dizgi akışları (stream) olarak temsil edilebilen nesneleri tanımlar.	`'\0'` olarak kodlanan `NULL` karakteri, karakterlerden oluşan bir karakter dizisini sonlandırır.
Dizgilerde dizi bozulması meydana gelmez çünkü dizeler nesneler olarak temsil edilir.	Bir dizinin tip ve boyut kaybına dizinin bozulması (decay of array) denir. Bu durum genellikle diziyi değer veya gösterici ile fonksiyona geçirdiğimizde ortaya çıkar. Karakter dizisinde böyle bir durum ortaya çıkabilir.
Bir dizgi sınıfı, dizgileri işlemek için çok sayıda fonksiyon sağlar.	Karakter dizileri, dizeleri işlemek için C++ dilinde yerleşik işlevler sunmaz.
Dizgilere bellek dinamik olarak tahsis edilir.	Karakter dizisine boyutu kadar veri bellekte (data segment) veya yığın bellekte (stack segment) yer statik olarak tahsis edilir.

Tablo 20. Dizgi ile Karakter Dizisi Karşılaştırması

Dizgi Fonksiyonları

C++ dilinde dizgileri kopyalamak ve birleştirmek için `strcpy()` ve `strcat()` işlevleri gibi dize işleme için kullanılan bazı yerleşik işlevler sağlar. Bunlardan bazıları şunlardır

Fonksiyon	Açıklama
`length()`	Metnin karakter uzunluğunu döndürür.
`swap()`	İki metni yer değiştirmek için kullanılır.
`size()`	Metnin boyutunu bulmak için kullanılır. Metne bellekte kaç karakter yer ayrılmış onu döner.
`resize()`	Metnin uzunluğunu belirtilen karakter sayısına kadar yeniden boyutlandırmak için kullanılır.
`find()`	Metnin içinde parametre olarak geçirilen bir başka metni arar.
`push_back()`	Parametre olarak geçirilen karakteri metnin sonuna eklemek kullanılır.
`pop_back()`	Metinden son karakteri çıkarmak için kullanılır.
`clear()`	Metnin tüm karakterlerini silmek için kullanılır.
`strncmp()`	Metin ile parametre olarak geçirilen bir başka metnin en fazla ilk num kadar karakterini karşılaştırır.
`strncpy()`	strcpy() fonksiyonuna benzerdir, ancak en fazla n bayt kopyalanır.
`strrchr()`	Metindeki bir karakterin son bulunduğu yeri bulur.

`strcat()`	Kaynak metnin bir kopyasını hedef metnin sonuna ekler.
`find()`	Bir metnin içerisinde belirli bir alt metni aramak için kullanılır ve alt metnin ilk karakterinin konumunu döndürür.
`replace()`	first-last aralığındaki eski değere eşit olan her bir elemanı yeni değerle değiştirmek için kullanılır
`substr()`	Verilen bir metinden bir alt metin oluşturmak için kullanılır.
`compare()`	İki metni karşılaştırmak ve sonucu tam sayı biçiminde döndürmek için kullanılır.
`erase()`	Bu fonksiyon bir metnin belli bir kısmını silmek için kullanılır.
`rfind()`	Metnin son bulunduğu konumu bulmak için kullanılır.
`capacity()`	Bu fonksiyon, derleyici tarafından dizgiye tahsis edilen kapasiteyi döndürür.
`shrink_to_fit()`	Bu fonksiyon dizginin kapasitesini en az (minimum) olacak şekilde azaltır.

Tablo 21. Çok Kullanılan Dizgi Fonksiyonları

Bunların yanında metni tutan dizgi değişkeni kullanılarak metin karakterleri üzerinde işlem yapmak için yineleme (itertation) yöntemlerini kullanabiliriz. Yineleme nesneleri diziler gibi birden çok aynı nesneyi tutan veri yapılarında elemanlar üzerinde hareket etmeyi sağlayana nesnelerdir.

Fonksiyon	Açıklama
`begin()`	Bu fonksiyon, dizginin başlangıcına işaret eden bir yineleyici (`iterator`) nesne döndürür.
`end()`	Bu fonksiyon, dizginin sonunu işaret eden bir yineleyici nesne döndürür.
`rbegin()`	Bu fonksiyon, dizginin başlangıcına işaret eden ters yineleyici (`reverse_iterator`) nesne döndürür.
`rend ()`	Bu fonksiyon, dizginin sonunu işaret eden ters yineleyici nesne döndürür.
`cbegin ()`	Bu fonksiyon, dizginin başlangıcına işaret eden bir sabit yineleyici (`const_iterator`) döndürür.
`cend()`	Bu fonksiyon dizginin sonunu işaret eden bir `const_iterator` döndürür.
`crbegin()`	Bu fonksiyon dizginin sonunu işaret eden bir sabit bir ters yineleyici (`const_reverse_iterator`) döndürür.
`crend()`	Bu fonksiyon, dizginin başlangıcına işaret eden bir yineleyici (`const_reverse_iterator`) döndürür.

Tablo 22. Dizgi Metni Yineleme Fonksiyonları

Yineleme fonksiyonlarına ilişkin örnek aşağıda verilmiştir;

```
#include <iostream>
using namespace std;

int main() {
    string::iterator itr; //string yineleme nesnesi
    string::reverse_iterator rit; //string ters yineleme nesnesi

    string metin = "Merhana Ilhan.";

    itr = metin.begin();
    cout << "Yineleyicinin işaret ettiği karakter: " << *itr<< endl;
    itr = metin.end() - 1;
    cout << "Yineleyicinin işaret ettiği son metin karakteri: " << *itr << endl;

    rit = metin.rbegin();
    cout << "Ters yineleyicinin işaret ettiği karakter:: " << *rit << endl;
    rit = metin.rend() - 1;
    cout << "Ters yineleyicinin işaret ettiği on metin karakteri:: " << *rit << endl;
}
/*Program çalıştığında:
Yineleyicinin işaret ettiği karakter: M
Yineleyicinin işaret ettiği son metin karakteri: .
```

```
Ters yineleyicinin işaret ettiği karakter:: .
Ters yineleyicinin işaret ettiği on metin karakteri:: M

...Program finished with exit code 0
*/
```