



# CPP Module 01

Bir class veya struct içerisinde private olarak oluşturulan bir öğeye o class veya struct dışından erişmek mümkün değildir. Eğer farklı bir alandan private veriye ulaşmak veya değiştirmek istiyorsak get ve set metodlarını yazmalıyız.

C içerisinde yer alan dinamik bellek yönetimi işlemleri C++ ile gelen NYP desteğini tam olarak sağlayamadığı için C++ ile bu işlemler **new** ve **delete** anahtar kelimeleri ile yapılır.

C++ içerisinde yer alan **new** ve **delete** operatörleri dinamik bellek yönetimi için kullanılır.

## C++ new ve delete kullanımı

C programlama dili ile bellek yönetimi sırasında kullanılan **malloc**, **calloc**, **realloc** ve **free** fonksiyonları C++ programlarında da kullanılabilir.

Bu fonksiyonların dönüş değeri **void\*** türünden olduğundan tür dönüşümüne ihtiyaç duyar.

```
#include <iostream>

int main()
```

```

{
    int *p;
    int i = 0;

    p = new int[5];

    delete[] p;

    return 0;
}

```

New → Blank class'ı varsa

Blanks \*SomeBlanks = new Blanks;

Referans type

```

#include <iostream>using namespace std;

int main(void)
{
    int id1, id2;
    int &rid = id1; // Referans tanımı

    id1 = 36;
    id2 = 57;

    cout << id1 << " " << rid << endl;

    rid = id2; // id2 değişken değerini id1 değişkenine aktarır. id2 değişkeni için bir referans oluşturmaz.

    cout << id1 << " " << id2 << " " << rid;

    return 0;
}

```

Yukarıdaki programı derleyip çalıştırıldığımızda, aşağıdaki ifadeleri ekrana yazar:

```

36 36
57 57 57

```

Program, önce iki adet int değişken oluşturur. id1 değişkeni için rid adlı bir referans tanımlar. Değişken adı ve referans kullanarak id1 değişken değerini iki kez ekrana yazar. Referans değişkenine id2 değişkenini atadığında, id2 değişkeni için bir referans oluşturmaz, sadece id2 değişken değerini id1 değişkenine atamış olur. Böylece her iki int değişkeni ve referans değişkeni aynı değeri gösterir.

& → C++ 'da referans operatördür.

Fonksiyon şöyle çağrırlır

```
int &count;
```

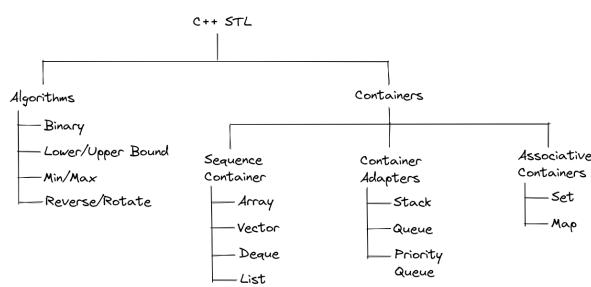
Yukarıdak ifade "Count bir tam sayıya işaret eder".

```
int Count = 1;
```

```
int &cRef = Count; //Count için cRef takma adını belirle.
```

```
++cRef; //Count değişkeninin değerini cRef takma adını kullanarak arttır.
```

STL (Standart template library) CPP08'e kadar yasak



#### Nesneye Dayalı Programlama Kavramları

- Fonksiyonel programlamada probleme  
"Verilen problemin çözümü için **algoritmayı** hangi **fonsiyonlara** parçalamalıyım" ?  
sorusu ile yaklaşılır.
- Nesneye dayalı programlamada ise  
"Verilen problemin çözümü için **veriyi** hangi **nesnelere** parçalamalıyım" ?  
sorusu ile yaklaşılır. Nesneye dayalı programlamada temel tasarım elemanı "nesneler"dir.

## C++'in C'ye Getirdiği Gelişmiş Özellikler

### Dinamik Bellek Kullanımı

C'de dinamik bellek kullanımı standart kütüphaneler kullanılarak gerçekleştirilmektedir:

```
int *p ; p = (int *) malloc(N*sizeof(int)) ; free(p) ;
```

### C++'da iki yeni operatör : new ve delete

```
int *p ;  
p = new int[N] ;  
delete []p ;
```

```
int *p,*q ;  
p = new int[9] ;  
q = new int(9) ;
```



## C++'in C'ye Getirdiği Gelişmiş Özellikler

devam

### Function Templates

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

## Nesne Üyelerine Erişimin Denetlenmesi

### Public

Public üyeler programdaki herhangi bir fonksiyon tarafından erişilebilir.

### Private

Herhangi bir tanımlama yok ise varsayılan tanımlama private'dır.  
Private üyeleri sadece o sınıfta ait üyeleri veya o sınıfın friend üyeleri tarafından erişilebilir.

### Protected (Public ile Private arasında)

Private üyeleri sadece o sınıfta ait üyeleri, o sınıftan türetilmiş sınıfların üyeleri veya o sınıfın friend üyeleri tarafından erişilebilir.

## Kopyalayıcı Kurucular

► Kopyalayıcı kurucu fonksiyonlar, o sınıftan bir nesnenin üyeleri yeni bir nesneye kopyalamakta kullanılır. Bu nedenle parametrelerden biri aynı sınıftan bir nesnedir.

► Eğer programcı tarafından tanımlanmamış ise derleyici bir tane yaratır. Yaratılan kurucu fonksiyon nesnenin birebir kopyasını oluşturur.

```
class string{  
    int size;  
    char *contents;  
public:  
    string();           // default constructor  
    string(string &); // copy constructor  
    void set (int, char *); // An ordinary member function  
    void print();  
};
```

```
string::string()      // Kurucu fonksiyon  
{  
    size = 0; contents = new char[1];  
    strcpy(contents, "");  
}  
string::string(string &in_object) // Kopyalayıcı kurucu fonksiyon  
{  
    size = in_object.size;  
    contents = new char[strlen(in_object.contents)];  
    strcpy(contents, in_object.contents);  
}  
void string::set(int in_size, char *in_data)  
{  
    size = in_size;  
    delete contents; contents = new char[strlen(in_data)];  
    strcpy(contents, in_data);  
}  
void string::print()  
{  
    cout << contents << " " << size << endl;  
}
```

# C++'ın C'ye Getirdiği Gelişmiş Özellikler

devam

## ■ Fonksiyon Yükleme (=Function Overloading)

```
double average(const double a[],int size) ;  
double average(const int a[],int size) ;  
double average(const int a[], const double b[],int size) ;
```

```
! double average(const int a[],int size) {  
    double sum = 0.0 ;  
    for(int i=0;i<size;i++) sum += a[i] ;  
    return ((double)sum/size) ;  
}
```

C++ ve NESNEYE DAYALI PROGRAMLAMA

81

```
#include <fstream>  
#include <iostream>  
#include <string>  
using namespace std;  
  
int main() {  
    // Specify the file to be read  
    string originalFile = "original.txt";  
    // Specify the string to be replaced  
    string oldString = "old";  
    // Specify the replacement string  
    string newString = "new";  
    // Open the file to be read  
    ifstream inFile(originalFile);  
    // Check if the file was successfully opened  
    if (inFile.is_open()) {  
        // Read the contents of the file  
        string content((istreambuf_iterator<char>(inFile)),  
                      istreambuf_iterator<char>());
```

```

// Close the file
inFile.close();
// Declare variable to store the modified content
string modifiedContent = "";
// Iterate through the contents of the file
for (int i = 0; i < content.length(); i++) {
    // Check if the current substring matches the old string
    if (content.substr(i, oldString.length()) == oldString) {
        // Replace the old string with the new string
        modifiedContent += newString;
        // Move the index to the end of the old string
        i += oldString.length() - 1;
    } else {
        // Add the current character to the modified content
        modifiedContent += content[i];
    }
}
// Specify the file to be written
string modifiedFile = "modified.txt";
// Open the file to be written
ofstream outFile(modifiedFile);
// Check if the file was successfully opened
if (outFile.is_open()) {
    // Write the modified content to the file
    outFile << modifiedContent;
    // Close the file
    outFile.close();
    cout << "Content modified and written to " << modifiedFile << endl;
} else {
    cout << "Unable to open file " << modifiedFile << endl;
}
} else {
    cout << "Unable to open file " << originalFile << endl;
}
return 0;
}

```