ITU SCT C++ Assignment Report

Mislina Feyza Özkan

December 2025

1 1. C++ Fundamentals: Data Types, Memory Model, and Basic Data Structures

```cpp
#include <iostream>
using namespace std;
double square(double x)
{
return x*x;
}
void print_square(double x)
{
cout << "the square of " << x << " is " << square(x) << "\n";
}
int main()
// hata kodu integer oldugu icin
{
print_square(5);
return 0;
}
```

Figure 1: square

Others > sizeof.cpp > ...

```cpp
#include <iostream>
#include <limits>
#include <iomanip>
#include <cstdint>

int main () {
    using namespace std;

    int file_size = 10;
    int counter = 0;
    double price = 8.50;
    std::cout <<file_size<< std::endl;
    std::cout <<counter << std::endl;
    std::cout <<sizeof(file_size)<< " elements\n";
    return 0;
}
```

Figure 2: sizeof

2

Dynamic Memory Allocation
new: bellekte değişkenler için alan ayırır.
delete: new ile ayrılan bellek silinir.
new[]: alanı dizi için ayırır.
delete[]: diziyi serbest bırakır.
static allocation: çalışan programın çalıştığı sürece sabit kalan bellek.
dynamic allocation: program çalışırken ayrılır ve istendiğinde serbest bırakılabilir.

3

Variables and Storage Duration
C++ dilinde değişkenler, nasıl tanımlandıklarına bağlı olarak farklı saklama sürelerine sahiptir. `auto` anahtar kelimesi, değişkenin türünün derleyici tarafından otomatik olarak belirlenmesini sağlar. Bu, kodun daha kısa ve okunabilir olmasına yardımcı olur.

`const` anahtar kelimesi, değeri tanımlandıktan sonra değiştirilemeyen değişkenler oluşturmak için kullanılır. Bu sayede önemli değerlerin yanlışlıkla değiştirilmesi önlenir. Benzer şekilde, `constexpr` anahtar kelimesi derleme zamanında hesaplanması gereken sabit değerler için kullanılır ve performans avantajı sağlar.

Bir diğer önemli kavram ise lvalue ve rvalue farkıdır. Lvalue, bellekte belirli bir adresi olan nesneleri ifade ederken; rvalue, geçici olan ve kalıcı bir bellek adresine sahip olmayan değerleri ifade eder. Bu farkın anlaşılması, özellikle pointer ve reference kullanımı açısından önemlidir.

4

Calculator

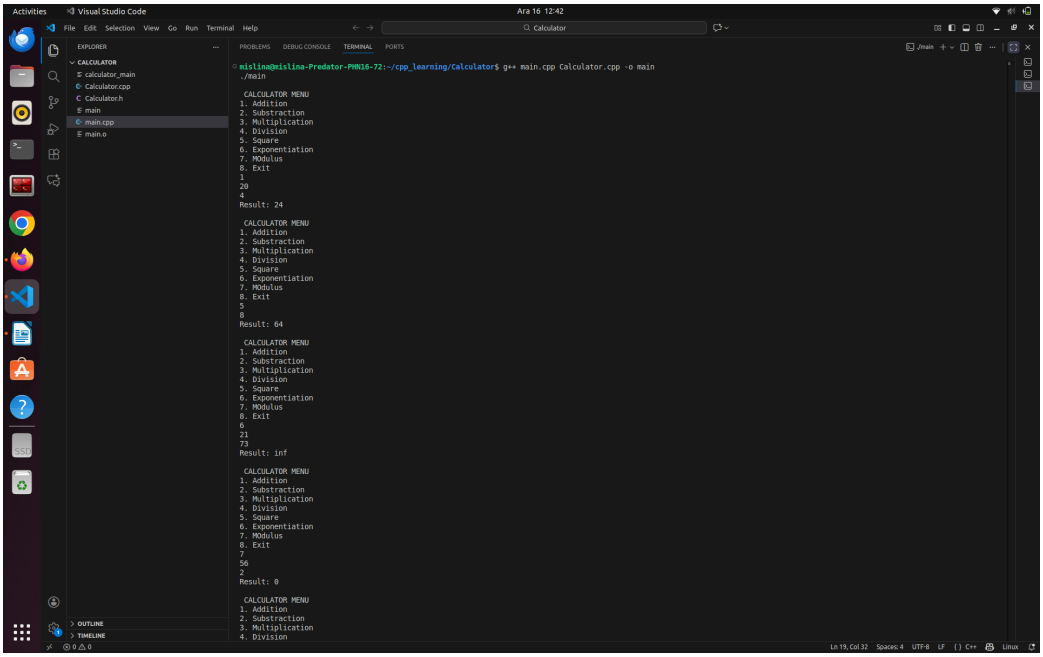

Figure 3: Calculator Outputs

5

Robot Dead Reckoning Simulator

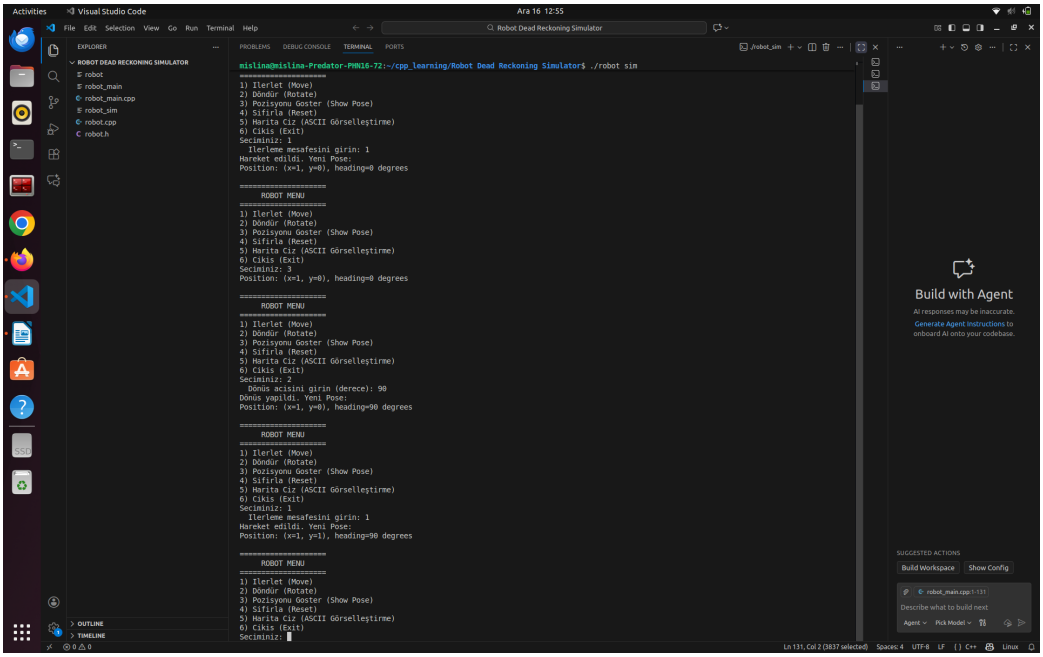

Figure 4: Robot Dead Reckoning Simulator Outputs-1

Bu simülasyonda robotun konumu, önceki konumu, hareket mesafesi ve yön açısı kullanılarak hesaplanmaktadır. Robotun yeni pozisyonu, trigonometrik hesaplamalar ile x ve y koordinatları üzerinden güncellenir.

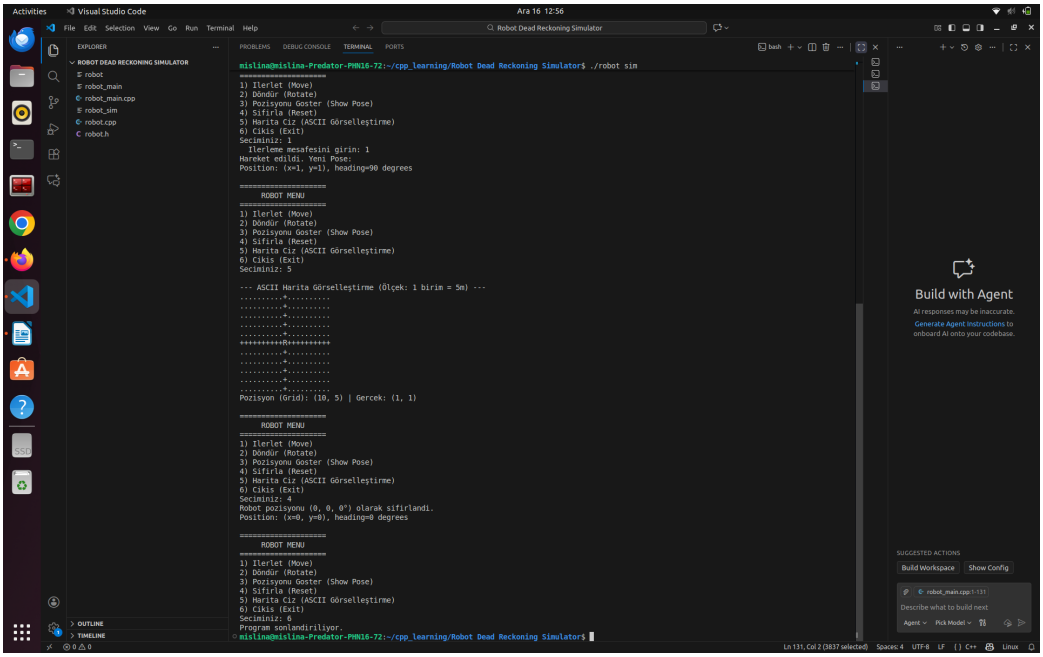

Figure 5: Robot Dead Reckoning Simulator Outputs-2

6

Memory Leak
Ayrılan belleğin delete veya delete[] ile serbest bırakılmaması durumunda olur. Uzun süre çalışan programlarda bellek tüketimini arttırır ve performansı düşürür.

Dinamik Belleğin Gerçek Hayattaki Kullanımı
Robotik: Sensör verilerini depolamak ve işlemek için kullanılır.
Oyun motorları: Karakterler, objeler ve haritalar için dinamik bellek tahsisi gerekir.
İşletim sistemleri: Dosya yönetimi, işlem yönetimi ve önbellekleme için kullanılır.
Dinamik bellek, programların esnek ve verimli çalışmasını sağlar, ancak doğru yönetilmezsembellek sızıntısı ve performans sorunlarına yol açabilir.

7

Introduction to Pointers
Pointer, bir değişkenin değerini değil, bellekteki adresini tutan özel bir değişkendir. Bu sayede bellek üzerinde doğrudan işlem yapılabilir ve dinamik bellek yönetimi mümkün hale gelir.

Bir pointer'ın gösterdiği adresteki değere erişmeye dereferencing denir. Pointer arithmetic ise pointer'ların bellekteki ardışık adresler arasında hareket edebilmesini sağlar ve genellikle dizilerle çalışırken kullanılır.

`nullptr`, herhangi bir geçerli bellek adresini göstermeyen pointer'ları temsil eder. `NULL` yerine `nullptr` kullanılması, modern C++'ta daha güvenli ve okunabilir kod yazılmasını sağlar.

Modern C++ ile birlikte gelen akıllı pointer'lar (`std::unique_ptr`, `std::shared_ptr` ve `std::weak_ptr`), dinamik belleğin otomatik olarak yönetilmesini sağlar. Bu yapılar, bellek sızıntılarını önlemeye yardımcı olur ve sahiplik (ownership) kavramını daha net hale getirir.

8

Point Structure
Istenilen veriyi sadece ihtiyac oldugunda cagirmayi saglar, boylece onlarin gerek olmadiginda uzun vadeli yer kaplamasina engel olur.

9

Introduction to Basic Data Structures

1. Array
Array, aynı türden verilerin ardışık olarak saklandığı bir veri yapısıdır.
Kullanım alanı: Sabit sayıda veriyi depolamak ve hızlı erişim gerektiğinde kullanılır.
Avantajları: Elemanlara indeksle hızlı erişim sağlar. Bellek kullanımı düzenlidir.
Dezavantajları: Boyutu sabittir; ekleme veya silme işlemleri masraflı olabilir.

2. Linked List
Linked List, her elemanın bir sonraki elemanın adresini tuttuğu ve elemanların birbirine
bağlı olduğu bir veri yapısıdır.
Kullanım alanı: Dinamik veri saklama ve sık ekleme-silme işlemleri gereken durumlarda kullanılır.
Avantajları: Belleği esnek kullanır, ekleme ve silme işlemleri hızlıdır.
Dezavantajları: Rastgele erişim yavaştır; ekstra hafıza (pointer) gerektirir.

3. Stack
Stack, LIFO prensibine göre çalışan bir veri yapısıdır.
Kullanım alanı: Fonksiyon çağrıları, undo işlemleri ve geri alma sistemlerinde kullanılır.
Avantajları: Basit ve hızlıdır; işlem sırası kolay yönetilir.
Dezavantajları: Sadece en üst elemanla işlem yapılabilir; rastgele erişim yoktur.

4. Queue
Queue, FIFO prensibine göre çalışan bir veri yapısıdır.
Kullanım alanı: İşlem sırasının önemli olduğu durumlar ve görev yönetimi sıralarında kullanılır.
Avantajları: İşlem sırası korunur; ekleme ve çıkarma kolaydır.
Dezavantajları: Rastgele erişim yoktur; sadece ön ve arka uçtan işlem yapılabilir.

10

C++ STL Containers (Overview)
STL, sık kullanılan veri yapıları için hazır konteynerler sunar. Bu konteynerler programcıya
hızlı ve güvenli bir şekilde veri yönetimi sağlar.

1. std::vector
• Kullanım Alanı: Dinamik boyutlu diziler için, eleman ekleme ve silme işlemlerinin
çoğunlukla sona yapılacağı durumlarda kullanılır.
• İçsel Veri Yapısı: Dinamik array olarak çalışır.
• Özellikleri: Rastgele erişim hızlıdır, boyut gerektiğinde otomatik büyür.

2. std::array
• Kullanım Alanı: Boyutu sabit olan dizilerde kullanılır.
• İçsel Veri Yapısı: Statik array.
• Özellikleri: Elemanlara hızlı erişim sağlar, performansı yüksektir; boyutu derleme
zamanında belirlenir.

3. std::list
• Kullanım Alanı: Sık ekleme ve silme işlemleri gereken durumlar için uygundur.
• İçsel Veri Yapısı: Doubly linked list.
• Özellikleri: Rastgele erişim yavaş, ekleme ve silme hızlıdır.

4. std::stack
• Kullanım Alanı: Fonksiyon çağrıları, geri alma (undo) işlemleri gibi son giren ilk çıkar
mantığı gereken durumlarda kullanılır.
• İçsel Veri Yapısı: Vector veya deque üzerinde LIFO davranışı uygular.
• Özellikleri: Sadece en üst elemanla işlem yapılabilir.

5. std::queue

• Kullanım Alanı: İşlem sırasının önemli olduğu durumlar, görev sıralama ve yazıcı kuyruğu
gibi uygulamalarda kullanılır.
• İçsel Veri Yapısı: Deque veya list üzerinde FIFO davranışı uygular.
• Özellikleri: Eleman ekleme arka uçtan, çıkarma ön uçtan yapılır.
Özet: STL konteynerleri, programcıya veri yönetimini kolaylaştırır ve performans avantajı
sağlar. Doğru konteyner seçimi, uygulamanın gereksinimlerine ve işlem türüne bağlıdır.

11

Stack vs Heap

Stack
• Fonksiyon içinde otomatik olarak oluşan bellek alanıdır.
• Çok hızlıdır çünkü bellek ardışık olarak yönetilir.
• Boyutu sınırlıdır ve büyük veri saklamaya uygun değildir.
Heap:
• new ile programcının ayırdığı dinamik bellek alanıdır.
• Daha yavaştır çünkü işletim sistemi yönetir.
• Büyük ve esnek yapılar için uygundur ama serbest bırakma unutulursa memory leak oluşur

12

Array vs Vector

Array
• Boyutu program başında belirlenir ve değişmez
• Bellekte ardışık tutulur, bu yüzden erişim çok hızlıdır.
• Basit ama esnek olmayan bir yapı sağlar.
Vector:
• Boyutu çalışma sırasında otomatik olarak büyüyebilir.
• Eleman ekleme/çıkarma kolaydır.
• İçeride dinamik array kullanır ve kapasite yönetimi yapar.

13

Pointer vs Reference Pointer:
• Bir değişkenin adresini tutar ve * ile dereference edilir.
• Null olabilir ve istenirse başka bir adresi gösterecek şekilde değiştirilebilir.
• Doğrudan bellek yönetimi ve dinamik bellek işlemlerinde kullanılır.
Reference:
• Bir değişkene takma ad olarak davranır ve null olamaz.
• İlk bağlandığı değişkeni değiştiremez, yeniden bağlama yapılamaz.
• Kullanımı pointer'dan daha güvenli ve kolaydır.