StockMarket-Simulator

> Projenin güncel durumunu buradan inceleyebilirsiniz: StockMarket-Simulator

Bu proje, bir yatırımcının başlangıçta $5,000 nakit ile başladığı bir oyun olarak tasarlanmıştır. Oyunun amacı, 20 yıl sonunda $10 milyon servete ulaşmaktır. Oyunda her turda 1 çeyrek yani 3 ay geçer.

Oyunun başlangıcında, oyuncunun yaşı ve adı sorulur ve 20 adet hisse senedi oluşturulur. Hisse senetleri rasgele semboller ve fiyatlar ile oluşturulur.

Oyun sırasında, oyuncu aşağıdaki seçeneklerden birini seçebilir:

1. Hisse senetlerini listelemek
2. Hisse seneti satın almak
3. Hisse seneti satmak

> 💡 Oyun sırasında, her turda hisse senetlerinin fiyatları ve nakit bakiyesi güncellenir. Oyun, 20 yıl geçmesi ile sona erer.

Projede işlenecek asgari konular

Proje gereksinimlerinde belirtilen asgari konular şunlardır:

1. Döngü kontrol ifadeleri (break, continue ve goto)
2. İç içe döngüler
3. Karar yapıları / ifadeleri (Switch-case yapısı mutlaka işlenmeli)
4. Diziler (Karakter dizileri mutlaka işlenmeli)
5. Fonksiyonlar

1. Döngü Kontrol İfadeleri (break, continue ve goto)

Break

Yazdığım kodda, kullanıcının yaptığı seçimlere göre fonksiyonlar çağrılmaktadır. Bu seçimler switch-case yapısı kullanılarak yapılmaktadır. Switch-case yapısı içerisinde de `break` keywordu kullanılmaktadır. Örnek olarak aşağıdaki kod parçasına bakabiliriz:

```
case 1:
  listStocks(stocks);
  break;
case 2:
  .
  .
  .
  break;
```

Burada kullanıcıdan aldığımız girdinin 1 olması durumunda `listStocks` fonksiyonu çağırılıyor ve ardın `break` komutu kullanılarak `case` 2'ye giren durumda gerçekleşmesini istediğimiz işlemlerin yapılması engelleniyor.

Continue

Bu programda `continue` anahtar kelimesi yalnızca bir yerde kullanılmıştır. Kullanıldığı kod bloğu aşağıdadır:

```
if (check == 1)
  {
    for (int i = 0; i < 20; i++)
    {
       stocks[i].price = updatePrice(stocks[i].price);
    }
    printf("\n-------------------------------\n");
    quarter += 1;
    continue;
  }
```

Burada kullanıcıdan alınan ifade eğer 1'e eşit ise döngüye baştan devam edilmesi sağlanmaktadır.

Goto

`Continue` ifadesinin kullanıldığı kod bloğunun devamında `goto` ifadesine de yer verilmiştir.

```
else
  goto here;
```

Burada kullanıcı 1 dışında bir değer kullanması durumunda here isimli label'a gidilmesi ve programa oradan devam edilmesi sağlanmaktadır. here etiketi ise şu şekilde belirtirmiştir.

```
here:
  .
  . //Codes
  .
```

İç İçe Döngüler

Proje genel hattıyla bir `while` döngüsü içerisinde yer almaktadır. Bu `while` döngüsünün içerisinde de çeşitli amaçlarla farklı döngüler bulunmaktadır. Bu bölümde ise iç içe `for` kullanılan bir fonksiyonu örnek vermek istiyorum.

```
void calculateTotalStocksValue(Stock *stocks)
{
    double totalValue = 0;
    for (int i = 0; i < 20; i++)
    {
        for (int j = 0; j < stocks[i].quantity; j++)
        {
            totalValue += stocks[i].price;
        }
    }
    printf("Total value of stocks: $%.2lf\n", totalValue);
}
```

Bu fonksiyon, `stocks` dizisindeki tüm hisselerin toplam değerini hesaplar. İlk döngü, her bir hisse için geçerlidir ve ikinci döngü, her bir hisse için mevcut miktar kadar hisse değerinin toplamını hesaplar. Bu fonksiyon sonunda, toplam hisse değeri ekrana yazdırılır. Örnek olarak, eğer stocks[0] için symbol "AAPL" price=150, quantity=5 ise, totalValue = 150*5 = 750 olur.

3. Karar Yapıları / İfadeleri

Projede hisselerin listelenmesi, satın alma ve satma işlemleri `switch-case` yapısı ile sağlanmaktadır.

```
switch (choice)
  {
  case 1:
    listStocks(stocks);
      break;
  case 2:
    listStocks(stocks);
    printf("\n\nEnter the symbol of the stock: ");
    scanf("%s", inputStock);
    inputStock[5] = '\0';
    for (int i = 0; i < 20; i++)
    {
      if (strcmp(inputStock, stocks[i].symbol) == 0)
      {
        int num = 0;
        printf("\tHow many stock do you want to buy: ");
        scanf("%d", &num);
        if (cash >= num * stocks[i].price)
        {
          stocks[i].quantity += num;
          cash -= num * stocks[i].price;
      }
        else
          printf("\nYou don't have enough money for this. Sell some stocks first!\n");
      }
    }
    break;
  case 3:
    listStocks(stocks);
    printf("\n\nEnter the symbol of the stock: ");
    scanf("%s", inputStock);
    inputStock[5] = '\0';
    for (int i = 0; i < 20; i++)
    {
        if (strcmp(inputStock, stocks[i].symbol) == 0)
        {
            int num = 0;
            printf("\tHow many stock do you want to sell: ");
            scanf("%d", &num);
            stocks[i].quantity -= num;
            cash += num * stocks[i].price;
        }
    }

    break;
  default:
    printf("Invalid choice. Please try again.\n");
    break;
  }
```

Kullanıcıdan alınan choice isimli değişkenin değerine göre yapılacak işlem belirleniyor.

Uygulamada kullanılan karar yapılarına bir örnek de `if-else` ile vermek gerekirse:

```
if (check == 1)
{
  for (int i = 0; i < 20; i++)
  {
      stocks[i].price = updatePrice(stocks[i].price);
  }
  printf("\n--------------------------------\n");
  quarter += 1;
  continue;
}
else
    goto here;
```

Burada ise kullanıcıdan alınan ve check değişkeninde tutulan integer tipindeki değer 1 ise if ifadesinin ardındaki kod bloğu çalışır. Stocks dizisinin tüm elemanları dönülüp, price değerleri updatePrice fonksiyonu ile yeniden belirlenir. Ardın quarter 1 artırılarak döngüye en başından devam edilir. Tur atlanmış olunur.

Eğer kullanıcı 1 dışında bir değer girerse daha önce belirttiğimiz here etiketinden, tur atlamadan devam eder.

4. Diziler

Programda başlıca iki tip dizi tutulmuştur. Bunlar karakter dizileri ve Stock yapısındaki diziler.

Stock

Stock yapısı aşağıda verilmiştir.

```
typedef struct stock
{
    char *symbol;
    double price;
    int quantity;
} Stock;
```

Stock yapısını incelediğimizde içerisinde bir de symbol isimli karakter dizisi olduğunu görmekteyiz.

Stock yapısı ile oluşturulan diziye bakalım.

```
Stock stocks[20];
```

Stock tipinde 20 elemanlı bir dizi oluşturulmuş oldu.

Bu diziye uygulanan manipülasyonlara bir örnek vermek olarak `initializeStockMarket` fonksiyonu verilebilir.

```
void initializeStockMarket(Stock *stocks)
{
    for (int i = 0; i < 20; i++)
    {
        stocks[i].symbol = (char *)malloc(5 * sizeof(char));
        randomSymbolCreator(stocks[i].symbol);
        stocks[i].symbol[4] = '\0';
        stocks[i].price = randomPriceGenerator();
        stocks[i].quantity = 0;
```

```
    }
}
```

Karakter Dizileri

Karakter dizisi olarak hisselerin sembolleri ve oyuncunun adı gibi değerler tutulmuştur.

Hisse sembolü yukarıda belirtilmişti. Şimdi bizzat o karakter dizisini manipüle eden fonksiyonu verelim.

```
char *randomSymbolCreator(char *symbol)
{
    for (int i = 0; i < 4; i++)
    {
        char randomletter = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"[rand() % 26];
        symbol[i] = randomletter;
    }
}
```

Bir de kullanıcının isminin alınırken kullanılan kodu inceleyelim.

```
char name[20];
printf("What's your name: ");
scanf("%s", name);
```

Burada 20 karakter boyutunda name isimli bir karakter dizisi oluşturulmuş. Ardından `scanf` fonksiyonu ile string ifade alınarak bu diziye atanmıştır.

Kullanılan karakter dizilerine bir örnek ise kullanıcıdan hisse sembolü alınırken kullanılan `inputStock` karakter dizisidir.

```
char *inputStock = (char *)malloc(5 * sizeof(char));
.
.
.
printf("\n\nEnter the symbol of the stock: ");
scanf("%s", inputStock);
inputStock[5] = '\0';
```

Bu karakter dizisi hafızada 5 karakterlik yer kaplamaktadır. Yine `scanf` fonksiyonu ile kullanıcının atama yapması sağlanmaktadır.

5. Fonksiyonlar

Main fonksiyon, programın çalışmasını başlatan ve yürütülen kod bloğudur. Bu projede, oyunun başlaması ile ilgili bilgilerin kullanıcıya verilmesi, hisselerin rastgele oluşturulduğu ve oyunun yıllar boyunca sürekli olarak güncellendiği için Main fonksiyon önemlidir. Ayrıca, oyunun sonunda kazanma veya kaybetme durumlarının kullanıcıya bildirilmesi de Main fonksiyonda gerçekleştirilir.

Main fonksiyon içinde, oyunun başlaması ile ilgili bilgilerin kullanıcıya verilmesi için `initializeApp` fonksiyonu çağrılır. Ayrıca, hisselerin rastgele oluşturulduğu `initializeStockMarket` fonksiyonu çağrılır.

Main fonksiyon içinde, oyunun yıllar boyunca sürekli olarak güncellendiği için bir while döngüsü kullanılır. Bu döngü içinde, oyunun zamanı ve oyuncunun para ve hisse durumlarının güncellendiği `calcNetWorth` fonksiyonu çağrılır. Ayrıca, oyunun sonunda kazanma veya kaybetme durumlarının kullanıcıya bildirilmesi için if-else yapısı kullanılır.

Main fonksiyon içinde, oyuncunun seçimlerini yapması için `choosesPrinter` fonksiyonu çağrılır ve kullanıcının seçimine göre switch-case yapısı kullanılarak fonksiyonlar çağrılır.

- `initializeApp()` fonksiyonu: Bu fonksiyon, oyunun ne olduğunu, oyuncunun 20 yıl sonunda $5k 'dan $10M servete ulaşması gerektiğini, oyunda her turda bir çeyrek yani 3 ay geçileceğini ve oyuna başlarken hazır mısın başlıyoruz tarzı yazıları ekrana yazdırmak için kullanılır.

```
void initializeApp()
{
  printf("Welcome to the Stock Market Game!\n");
  sleep(1);
  printf("In this game, you will start with $5,000 and have 20 years to reach a net worth of $10,000,000.\n");
  sleep(3);
  printf("You will be simulating a quarter at a time, or 3 months.\n");
  sleep(2);
  printf("Are you ready to begin? Starting in 3...\n");
  sleep(1);
  printf("2...\n");
  sleep(1);
  printf("1...\n");
  sleep(1);
  printf("Let's go!\n");
}
```

- `initializeStockMarket(Stock *stocks)` fonksiyonu: Bu fonksiyon, hisse senetleri piyasasının başlangıç durumunu oluşturmak için kullanılır. 20 adet hisse senedi oluşturulur ve rastgele semboller, fiyatlar ve miktarlar atanır.

```
void initializeStockMarket(Stock *stocks)
{
    for (int i = 0; i < 20; i++)
    {
        stocks[i].symbol = (char *)malloc(5 * sizeof(char));
        randomSymbolCreator(stocks[i].symbol);
        stocks[i].symbol[4] = '\0';
        stocks[i].price = randomPriceGenerator();
        stocks[i].quantity = 0;
    }
}
```

- `randomSymbolCreator(char *symbol)` fonksiyonu: Bu fonksiyon, rastgele semboller oluşturmak için kullanılır. 4 adet rastgele harf döndürür.

```
char *randomSymbolCreator(char *symbol)
{
    for (int i = 0; i < 4; i++)
    {
        char randomletter = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"[rand() % 26];
        symbol[i] = randomletter;
    }
}
```

- `randomPriceGenerator()` fonksiyonu: Bu fonksiyon, rastgele bir fiyat oluşturmak için kullanılır. 0 ile 100 arasında bir fiyat döndürür.

```
double randomPriceGenerator()
{
    return (double)rand() / RAND_MAX * 100;
}
```

- `freeMemory(Stock *stocks, int size)` fonksiyonu: Bu fonksiyon, bellekte kullanılan hisse senedi yapılarının bellekten serbest bırakılması için kullanılır.

```
void freeMemory(Stock *stocks, int size)
{
    for (int i = 0; i < size; i++)
    {
        free(stocks[i].symbol);
        stocks[i].symbol = NULL;
    }
}
```

- `updatePrice(double price)` fonksiyonu: Bu fonksiyon, her turda hisse senetlerinin fiyatlarının güncellenmesi için kullanılır.

```
double updatePrice(double price)
{
  double randNum = (double)rand() / RAND_MAX; // 0 ile 1 arasında rastgele sayı
  randNum = randNum * 1.5 - 0.5;              // -0.5 ile 1 arasında sayı elde etmek için
  return (double)price * (randNum + 1);
}
```

- `listStocks(Stock *stocks)` : Bu fonksiyon, oyunda mevcut olan tüm hisselerin listesini ekrana yazdırır. Bu fonksiyon, Stock tipinde bir dizi alır ve içindeki tüm elemanların sembol, fiyat ve miktar bilgilerini ekrana yazdırır. Bu fonksiyon her seferinde oyun içinde kullanılır, oyuncunun mevcut hisse durumunu kontrol etmesi için. Bu fonksiyon herhangi bir değer döndürmez.

```
void listStocks(Stock *stocks)
{
  for (int i = 0; i < 20; i++)
  {
    printf("\n%s | %.2lf | %d", stocks[i].symbol, stocks[i].price, stocks[i].quantity);
  }
  printf("\n");
}
```

- `calcNetWorth(Stock *stocks, double netWorth, double cash)` : Bu fonksiyon, oyuncunun mevcut net değerini hesaplar. Bu fonksiyon, Stock tipinde bir dizi, oyuncunun mevcut net değerini ve mevcut nakit miktarını alır. Fonksiyon içinde, oyuncunun mevcut hisse senedi miktarının fiyatı ile çarparak toplam değer hesaplanır ve oyuncunun mevcut nakit miktarı da bu değer ile toplandıktan sonra geri döndürülür. Bu fonksiyon her seferinde oyun içinde kullanılır, oyuncunun net değerini kontrol etmek için. Bu fonksiyon double tipinde bir değer döndürür.

```
double calcNetWorth(Stock *stocks, double netWorth, double cash)
{
    netWorth = cash;
    for (int i = 0; i < 20; i++)
    {
        if (stocks[i].quantity != 0)
```

```
        {
            netWorth += stocks[i].price * stocks[i].quantity;
        }
    }
    return netWorth;
}
```

- `choosesPrinter()` : fonksiyonu, oyuncunun oyunda yapabileceği seçenekleri ekrana yazdırması için kullanılır. Bu fonksiyon, oyun içinde her turda çağrılır ve oyuncunun oyunda ne yapabileceğini gösterir. Bu fonksiyonun parametreleri yoktur ve bir dönen değeri de yoktur. Bu fonksiyon, switch-case yapısı kullanılarak yazılmıştır ve oyuncunun oyunda yapabileceği seçenekleri ekrana yazdırmak için kullanılır. Örneğin, oyuncunun hisseleri görüntüleme, hisse satın alma veya hisse satma gibi seçenekleri ekrana yazdırır.

```
void choosesPrinter()
{
    printf("******************");
    printf("\nWhat would you like to do?\n");
    printf("1. View stocks\n");
    printf("2. Buy stocks\n");
    printf("3. Sell stocks\n");
    printf("******************\n\n");
}
```

- `calculateTotalStocksValue` fonksiyonu: Bu fonksiyon, `stocks` dizisindeki tüm hisselerin toplam değerini hesaplar. İlk döngü, her bir hisse için geçerlidir ve ikinci döngü, her bir hisse için mevcut miktar kadar hisse değerinin toplamını hesaplar. Bu fonksiyon sonunda, toplam hisse değeri ekrana yazdırılır. Örnek olarak, eğer stocks[0] için symbol "AAPL" price=150, quantity=5 ise, totalValue = 150*5 = 750 olur.

```
void calculateTotalStocksValue(Stock *stocks)
{
    double totalValue = 0;
    for (int i = 0; i < 20; i++)
    {
        for (int j = 0; j < stocks[i].quantity; j++)
        {
            totalValue += stocks[i].price;
        }
    }
    printf("Total value of stocks: $%.2lf\n", totalValue);
}
```