Omar Mohamed Atia Mohamed Shehab

Professor Mohamed Eid

Computer Programming for Engineers

November 27, 2022

Step 1: Problem Identification and Statement:

- Time-division multiple access (TDMA) is a channel access method for shared-medium networks. Several users share the same frequency channel by dividing the signal into different time slots. The users transmit in rapid succession, one after the other, **each using its own time slot**. This allows multiple stations to share the same transmission medium which is the **Base Transceiver Station(BTS)**.
- Each Base Transceiver Station has different time slots for the mobile stations and each time slot has its own frequency. This BTS has a large scale network which is the cellular telecommunication system and it's called **Base Station Controller(BSC)**. The BSC contains many BTSs. Each BTS is considered the network control of both radio transmission and the interface to a mobile phone. Each Mobile Station(MS) comprises the user equipment(mobile device) and software needed for communication with a mobile network.
- The importance of the program is to calculate the distance of the MS on each move to get the optimal distance(the shortest one) and based on it the MS will connect to the nearest BTS. Each time the MS moves from a location to another, an update function will be called to check the

distance and based on that it will decide the correct option which is the nearest BTS.

Step 2: Gathering Information:

- **Architecture of the cellular network:**

We have the base station controller which is the main station for controlling the process of mobile station connections. It contains smaller stations called base transceiver stations which control the radio and mobile phones transmissions. The BTS connects mobile devices to the network. It sends and receives radio signals to mobile devices and converts them to digital signals that it passes on the network to route to other terminals in the network or to the Internet. This helps the mobile stations to connect to the nearest BTS. The figure below shows a graphical representation of how the system looks like (figure 1).

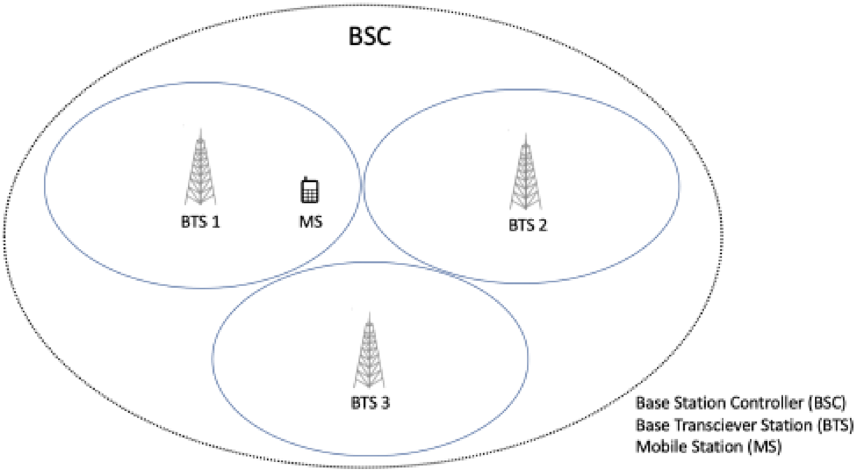


Figure 1: Architecture of the cellular network.

- **Distance calculation:**

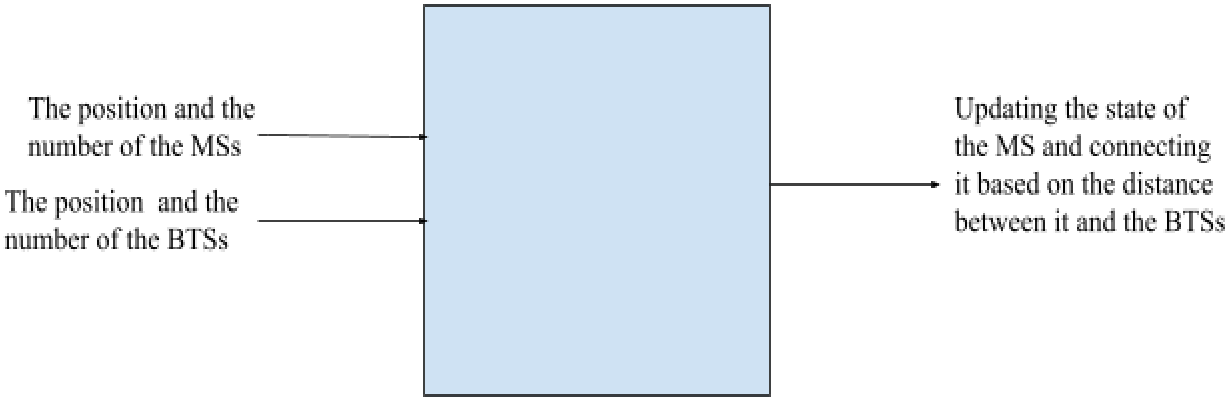
This is considered the main step of the process because everything is based on it. As if we calculated the distance of the MS to the BTS it will check if it's the nearest BTS to it or not. If it's the nearest one then it will connect to it if there are empty slots. If it's not the nearest one then it will go to the next BTS object and do the same process again till it reaches the least distance out of all objects. And the formula that we should use to calculate the distance between the MS and the BTS is:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Where x1 and x2 are the dimensions of the two objects on the x axis, y1 and y2 are the dimensions of the two objects on the y axis, and z1 and z2 the dimensions of the two objects on the z axis**.**

- **I/O Menu:**

The position of the MSs and BTSs will be provided to allow the program to use it in order to calculate the distance, then the function update the active BTS will be called in order to update the position of the MS and connect it to the suitable BTS.


The position and the number of the MSs
The position and the number of the BTSs
Updating the state of the MS and connecting it based on the distance between it and the BTSs

Step 3: Test cases and Algorithms:

A. Test cases:

- **Test case 1:**

MS Position	BTS Position	ID of the BTS containing MS
(10,11,12)	The BTS ID: 1 BTS Position: X: 10 Y: 11 Z: 12 number of free slots: 2	1 then moved to 4
(11,13,12)	The BTS ID: 2 BTS Position: X: 11 Y: 12 Z: 13 number of free slots: 7	2
(19,17,11)	The BTS ID: 3 BTS Position: X: 9 Y: 8 Z: 10 number of free slots: 0	4 then moved to 1
(4,3,5)	The BTS ID: 4 BTS Position: X: 19 Y: 18 Z: 10 number of free slots: 7	3
(5,4,3)	The BTS ID: 5 BTS Position: X: 30 Y: 30 Z: 10 number of free slots: 7	3 then moved to 5
(3,4,5)		3
(0,1,0)		3
(0,0,1)		3
(1,0,1)		3
(0,1,1)		3
(0,2,1)		3
(3,2,1)		1
(10,2,1)		1
(5,12,2)		1

(2,3,1)		1
(3,3,5)		1
(5,4,2)		1

- **Test case 2**

MS Position	BTS Position	ID of the BTS containing MS
(10,11,12)	The BTS ID: 1 BTS Position: X: 10 Y: 11 Z: 12 number of free slots: 0	1
(10,11,12)	The BTS ID: 2 BTS Position: X: 11 Y: 12 Z: 13 number of free slots: 7	1
(10,11,12)	The BTS ID: 3 BTS Position: X: 9 Y: 8 Z: 10 number of free slots: 8	1
(10,11,12)	The BTS ID: 4 BTS Position: X: 19 Y: 18 Z: 10 number of free slots: 8	1
(10,11,12)	The BTS ID: 5 BTS Position: X: 30 Y: 30 Z: 10 number of free slots: 8	1
(10,11,12)		1
(10,11,12)		1
(10,11,12)		1
(10,11,12)		2

- **Test case 3**

MS Position	BTS Position	ID of the BTS containing MS
(10,11,12)	The BTS ID: 1 BTS Position: X: 10 Y: 11 Z: 12 number of free slots: 0	1

(10,11,12)		1
(10,11,12)		1
(10,11,12)		1
(10,11,12)		1
(10,11,12)		1
(10,11,12)		1
(10,11,12)		1
(10,11,12)		The BTS 1 is full

- **Test case 4**

MS Position	BTS Position	ID of the BTS containing MS
(10.3,11.1,12.5)	The BTS ID: 1 BTS Position: X: 10 Y: 11 Z: 12 number of free slots: 7	1
(-10,0.11,12)	The BTS ID: 2 BTS Position: X: -1 Y: 2 Z: 13 number of free slots: 7	2

For the first test case, there are 17 MS and 5 BTS with different positions. then moved 3 MS to check the move, updateActiveBTS, leave functions and the results showed that each MS approaches the nearest BTS and after calling the move function it does the same thing. For the second test case, there are 9 MS with the same position and 5 BTS with different positions. This test case is made to check the behavior of the MS when it finds the nearest BTS full. And the results show that it searches for the second nearest BTS.

For the third test case, there are 9 MS with the same position and 1 BTS. This test case is made to check what the MS does when it finds a full BTS. For the fourth test case, there are 2 MS with decimal and negative numbers and 2 BTS with the same approaches to check how the program deals with different types of coordinates.

B. Algorithms:

```
Define SIZE to 10000 as a symbolic constant
Define SLOTS to 8 as symbolic constant
Class definition MobileStation
Class definition BaseStationController
Class definition BaseTranscieverStation
     Private data members:
          Declare numOfMS as int
          Declare ID_BTS as static int
          Declare GeneratedID as int
          Declare numOfBTS as int
          Declare position[3] as double
          Declare pointer numOfOccupiedSlots[8] as array of
MobileStatoin objects
     Public data members:
          Default Constructor BaseTranscieverStation
               Initialize Position[0], position[1], position[2]
     to 0
               Initialize numOfMS to 0
               Increment ID_BTS by 1
               Assign GeneratedID to ID_BTS
          Parameterized Constructor BaseTranscieverStation
     (double x, double y, double z)
               Assign position[0] to x
               Assign position[1] to y
               Assign position[2] to z
               Assign numOfMS to 0
               Increment ID_BTS by 1
               Assign GeneratedID to ID_BTS
          Declare *getPosBTS as double function
               Return position
```

```
        Declare getID_BTS as int function
            Return GeneratedID
        Declare connectMSToBTS(MobileStation *newMS) as boolean function
        Declare numberOfDevices as void function
        Declare leave_MS_BTS(MobileStation *newMS)
        Declare num_of_freeslots as int function
            Return (SLOTS - numOfMS)
        Declare print as void function
            Print “The BTS ID: “ getID_BTS() newline
            Print “BTS Position: “ newline
            Print “X: “ position[0] “ Y: “ position[1] “ Z: “ position[2] newline
            Print “number of free slots: “ num_of_freeslots() newline
        Destructor BaseTranscieverStation

Declare BaseTranscieverStation::numberOfDevices as void function
    Print “Number of connected MS: “ GeneratedID newline

Declare BaseTranscieverStation::ID_BTS and assign it to 0
Class definition MobileStation
    Private data members:
        Declare ID as static int
        Declare GeneratedID as int
        Declare pos[SIZE] as array of type double
        Declare pointer connectedBTS as BaseTranscieverStation object
        Declare pointer connectedBSC as BaseStationController object
    Public data members:
        Default Constructor MobileStation
```

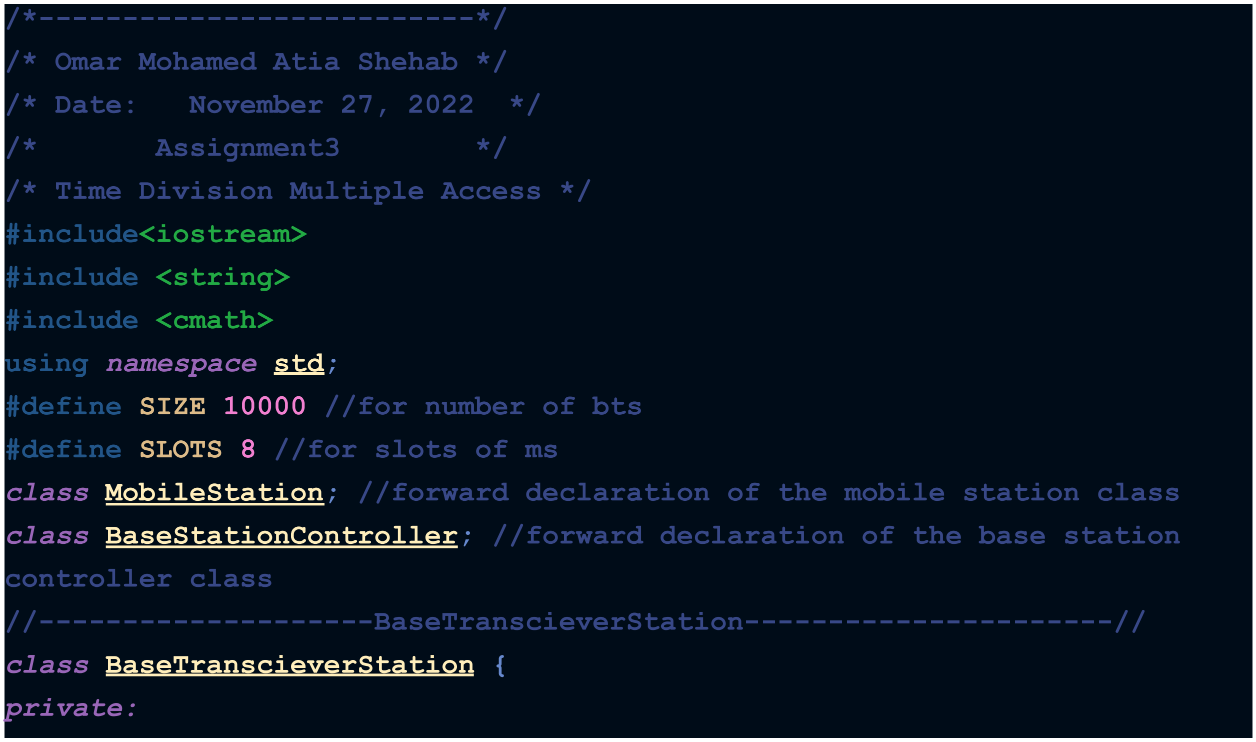
```
            Increment ID by 1
            Assign GeneratedID to ID
            Assign pos[0], pos[1], pos[2] to 0
        Parameterized Constructor MobileStation
            Increment ID by 1
            Assign GeneratedID to ID
            Assign pos[0] to x
            Assign pos[1] to y
            Assign pos[2] to z
        Declare getID as int function
            Return GeneratedID
        Declare *getPos as double function
            Return pos
        Delare updateActiveBTS(BaseStationController* BSC, int
    numOfBTS) as void function
        Declare connected_BTS(BaseTranscieverStation *BTS) as
    void function
            Assign connectedBTS to BTS
        Declare numberOfDevice as static void function
            Print “The number of MS devices is: “ ID newline
        Declare move(double x, double y, double z,
        BaseStationController* nn, int n) as void function
        Declare printPos as void function
            Print “X: “ pos[0] “ Y: “ pos[1] “ Z: “ pos[2]
        newline
        Destructor MobileStation

Assign MobileStation::ID to 0
Class definition BaseStationController
    Private data members:
        Declare pointer numOfBTS as BaseTranscieverStation
    object
```

```
        Declare numBTS as static int
    Public data members:
        Default constructor BaseStationController
            Assign numBTS to 0
            Assign numOfBTS to new BaseTranscieverStation[SIZE]
        Declare Add_BTS(BaseTranscieverStation BTS) as void function
            If numBTS less than SIZE
                numOfBTS[numBTS] = BTS
                Increment numBTS by 1
            Else
                Print “No Space for more BTSs as the max number of BTSs is ‟ SIZE newline
        Declare PrintInfo  as void function
        Declare *getAllBTS as BaseTranscieverStation
            Return numOfBTS
        Declare getNUM as static int
            Return numBTS
        Destructor BaseStationController
Assign BaseStationController::numBTS to 0
Declare BaseTranscieverStation::connectMSToBTS(MobileStation *newMS) as boolean function
    If numOfMS greater than or equal to SLOTS
        Return false
    Else
        Assign numOfOccupiedSlots[numOfMS] to newMS
        newMS->connected_BTS(this)
        Increment numOfMS by one
        Print “MS ‟ newMS->getID() ‟ to BTS with ID: ‟ GeneratedID newline
        Return true
```

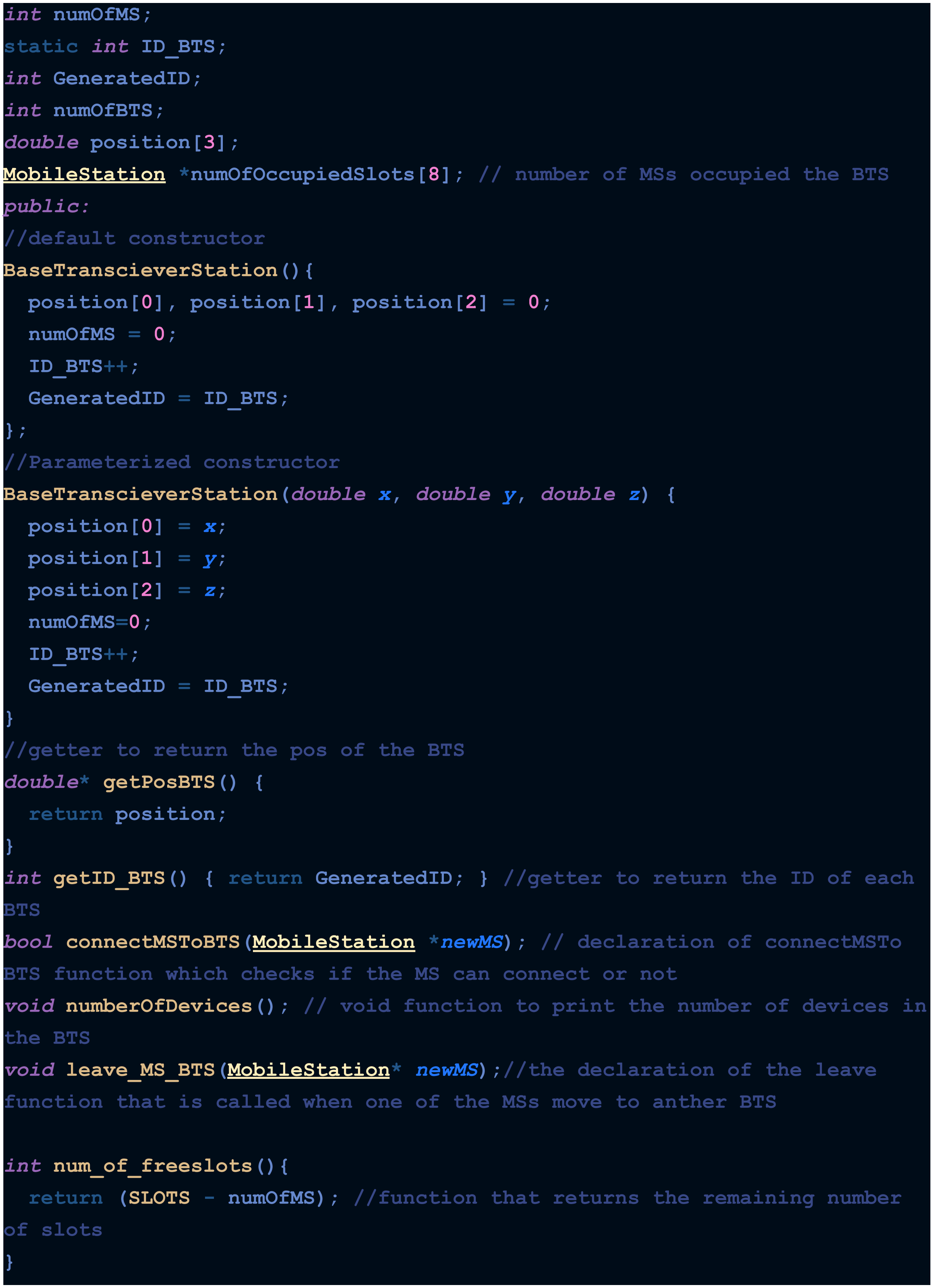
```
Declare BaseTranscieverStation::leave_MS_BTS(MobileStation *newMs) as void function
    Declare check as boolean and assign it to false
    For int i = 0 to i < (8-num_of_freeslots()) repeat
        if(newMS equals numOfOccupiedSlots[i])
            For int j=0 to j<(8-num_of_freeslots()) repeat
                Assign numOfOccupiedSlots[j] to numOfOccupiedSlots[j+1]
            Decrement numOfMS by 1
            Assign check to true
    if(not check)
        Print ‟ No MS to leave” newline
    Else
        Print “MS” newMS->getID() ‟ Left the BTS ‟ this->getID_BTs() newline
Declare MobileStation::updateActiveBTS (BaseStationController* BSC, int numofBTS) as void function
    Declare Dis as variable of type double and assign it to 0
    Declare check as variable of type int and assign it to 0
    Declare BTS as BaseTranscieverStation object and assign it to BSC->getAllBTS()
    Declare min as variable of type double and assign to 1000000000
    For int i = 0 to i < numofBTS repeat
        If BTS[i].num_of_freeslots() equal to 0
            continue
        Assign Dis square root of (pos[0]-(BTS[i].getPosBTS()[0]))^2 + (pos[1]-(BTS[i].getPosBTS()[1]))^2 + (pos[2]-(BTS[i].getPosBTS()[2]))^2
        If Dis less than min
            Assign min to Dis
```

```
                Assign check to i
     If BTS[check].connectMSToBTS(this) is false
                Print “ The BTS ” BTS[check].getID_BTS() “ is
full “ newline
Declare MobileStation::move(double x, double y, double z,
BaseStationController* newBSC, int n
     Assign pos[0] to x
     Assign pos[1] to y
     Assign pos[2] to z
     Print “MS “ getID() “ Moved to: (“ pos[0] “ , “ pos[1] “ ,
“ pos[2] newline
     connectedBTS->leave_MS_BTS(this)
     this->updateActiveBTS(newBSC, n)
Declare BaseStationController::PrintInfo as void function
     For int i =0 to i < numBTS repeat
          Call numOfBTS[i].print()
          Print newline
```

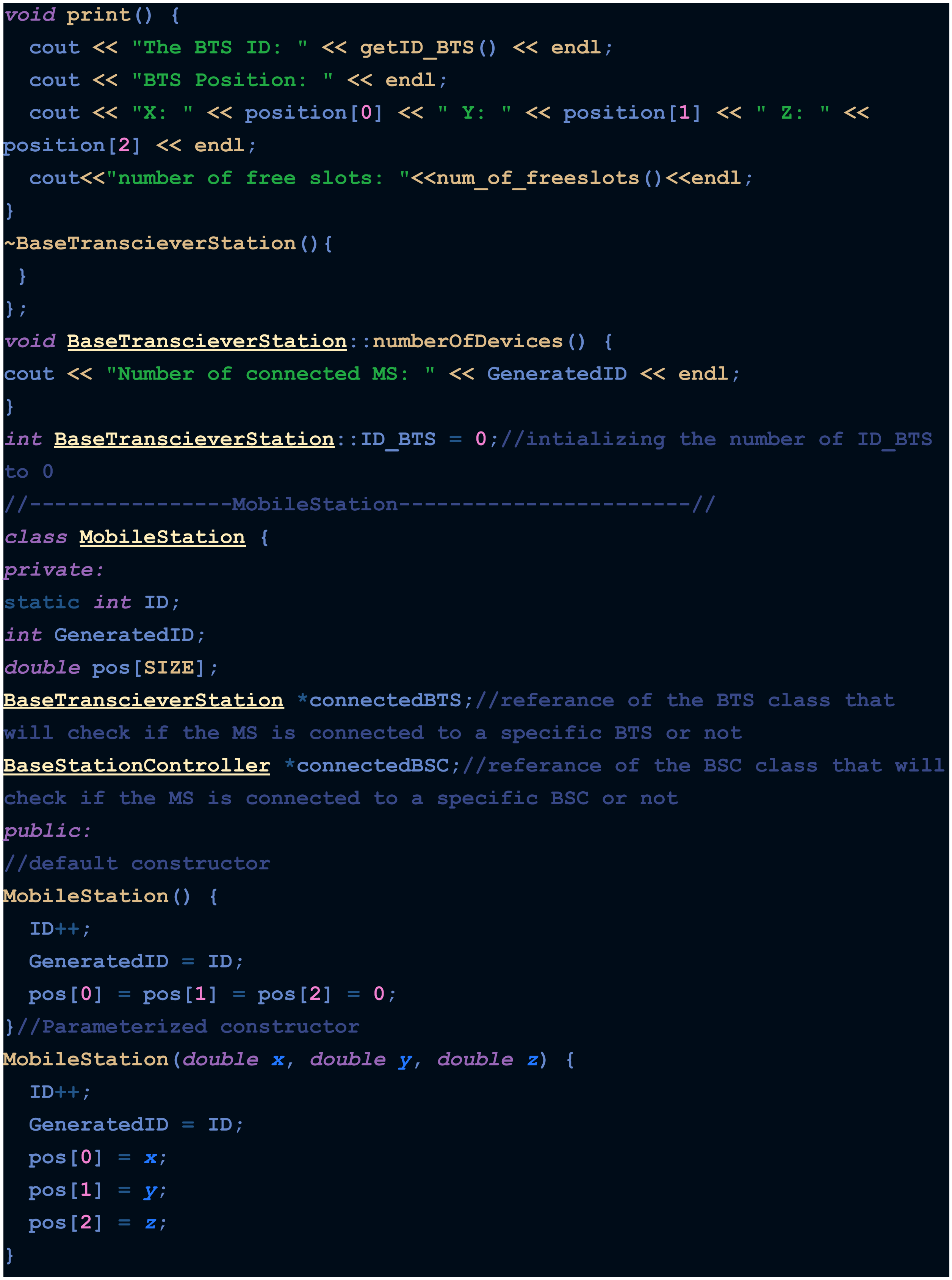
Step 4: Code



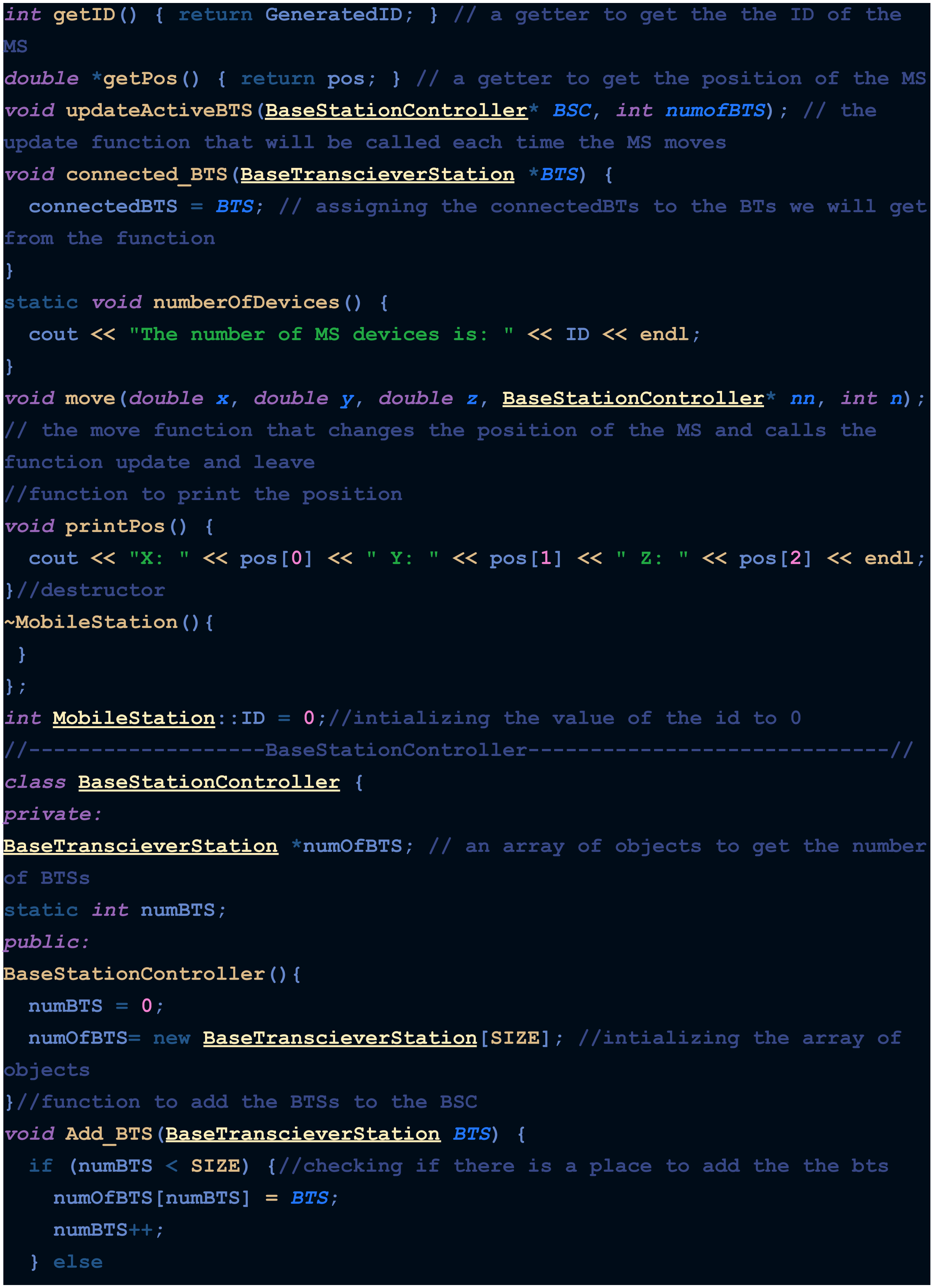
```
/*---------------------------*/
/* Omar Mohamed Atia Shehab */
/* Date:   November 27, 2022   */
/*       Assignment3        */
/* Time Division Multiple Access */
#include<iostream>
#include <string>
#include <cmath>
using namespace std;
#define SIZE 10000 //for number of bts
#define SLOTS 8 //for slots of ms
class MobileStation; //forward declaration of the mobile station class
class BaseStationController; //forward declaration of the base station
controller class
//--------------------BaseTranscieverStation----------------------//
class BaseTranscieverStation {
private:
```



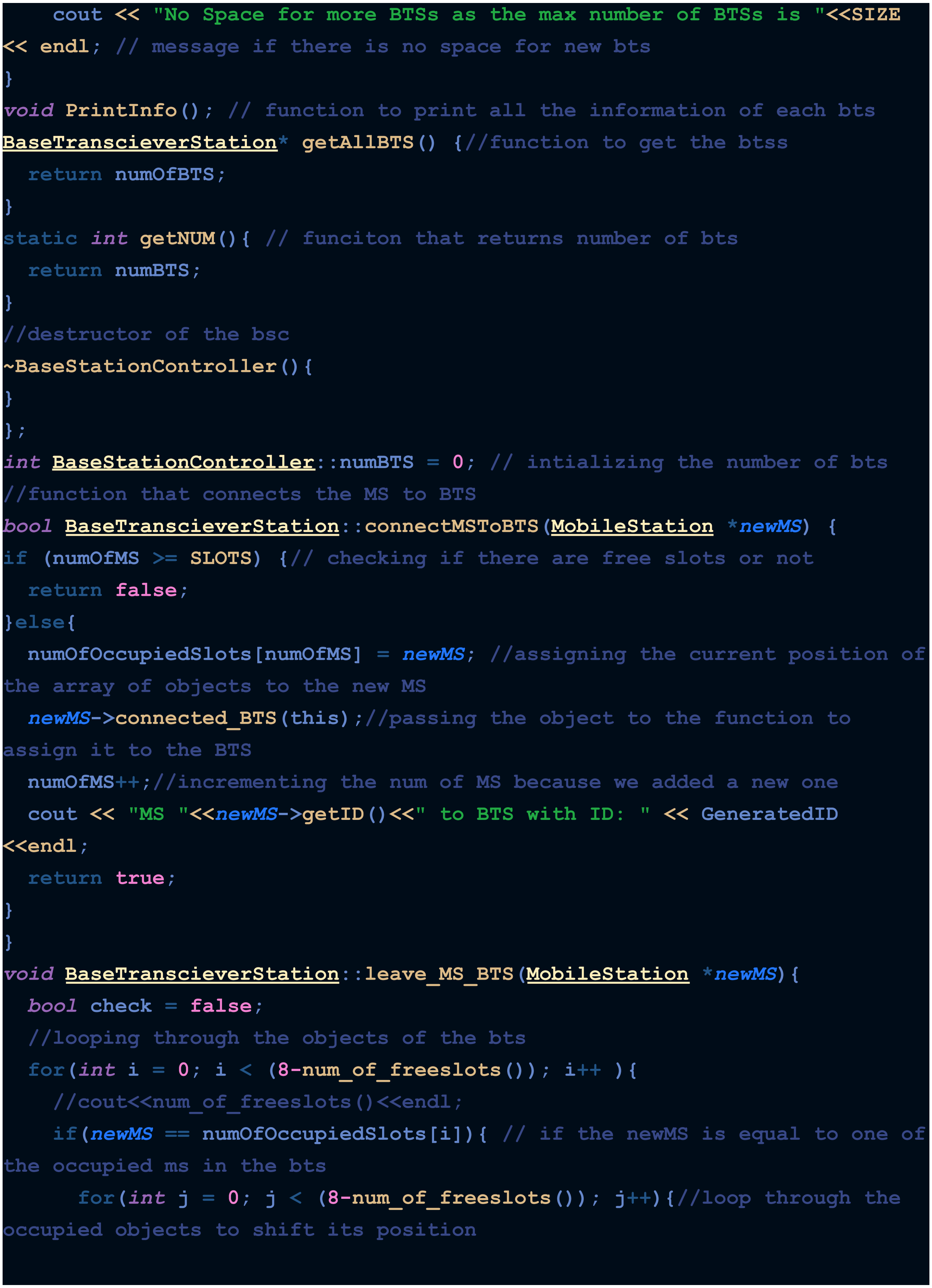
```
int numOfMS;
static int ID_BTS;
int GeneratedID;
int numOfBTS;
double position[3];
MobileStation *numOfOccupiedSlots[8]; // number of MSs occupied the BTS
public:
//default constructor
BaseTranscieverStation(){
  position[0], position[1], position[2] = 0;
  numOfMS = 0;
  ID_BTS++;
  GeneratedID = ID_BTS;
};
//Parameterized constructor
BaseTranscieverStation(double x, double y, double z) {
  position[0] = x;
  position[1] = y;
  position[2] = z;
  numOfMS=0;
  ID_BTS++;
  GeneratedID = ID_BTS;
}
//getter to return the pos of the BTS
double* getPosBTS() {
  return position;
}
int getID_BTS() { return GeneratedID; } //getter to return the ID of each
BTS
bool connectMSToBTS(MobileStation *newMS); // declaration of connectMSTo
BTS function which checks if the MS can connect or not
void numberOfDevices(); // void function to print the number of devices in
the BTS
void leave_MS_BTS(MobileStation* newMS);//the declaration of the leave
function that is called when one of the MSs move to anther BTS

int num_of_freeslots(){
  return (SLOTS - numOfMS); //function that returns the remaining number
of slots
}
```



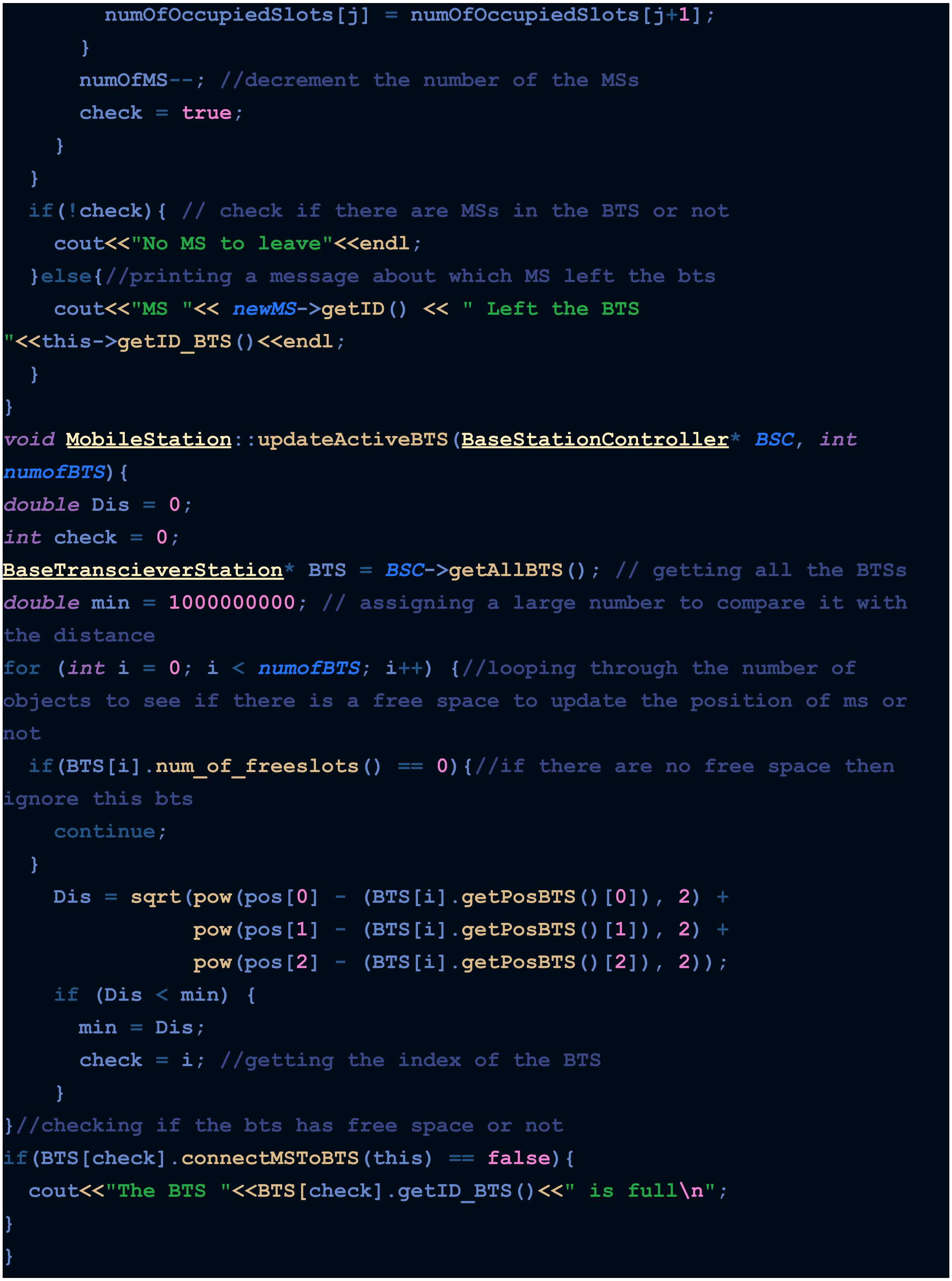
```
void print() {
  cout << "The BTS ID: " << getID_BTS() << endl;
  cout << "BTS Position: " << endl;
  cout << "X: " << position[0] << " Y: " << position[1] << " Z: " <<
position[2] << endl;
  cout<<"number of free slots: "<<num_of_freeslots()<<endl;
}
~BaseTranscieverStation(){
 }
};
void BaseTranscieverStation::numberOfDevices() {
cout << "Number of connected MS: " << GeneratedID << endl;
}
int BaseTranscieverStation::ID_BTS = 0;//intializing the number of ID_BTS
to 0
//----------------MobileStation-----------------------//
class MobileStation {
private:
static int ID;
int GeneratedID;
double pos[SIZE];
BaseTranscieverStation *connectedBTS;//referance of the BTS class that
will check if the MS is connected to a specific BTS or not
BaseStationController *connectedBSC;//referance of the BSC class that will
check if the MS is connected to a specific BSC or not
public:
//default constructor
MobileStation() {
  ID++;
  GeneratedID = ID;
  pos[0] = pos[1] = pos[2] = 0;
}//Parameterized constructor
MobileStation(double x, double y, double z) {
  ID++;
  GeneratedID = ID;
  pos[0] = x;
  pos[1] = y;
  pos[2] = z;
}
```



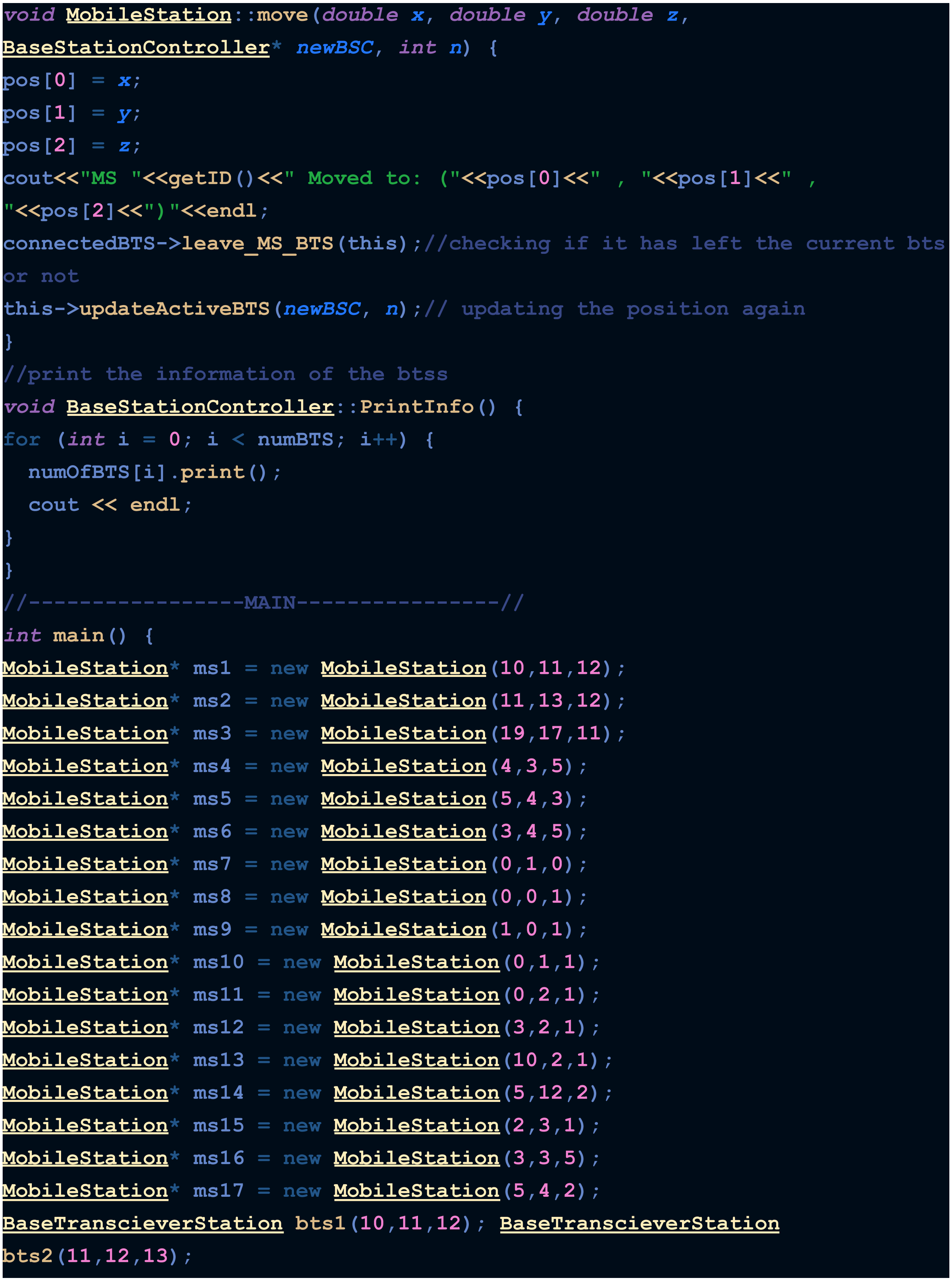
```
int getID() { return GeneratedID; } // a getter to get the the ID of the
MS
double *getPos() { return pos; } // a getter to get the position of the MS
void updateActiveBTS(BaseStationController* BSC, int numofBTS); // the
update function that will be called each time the MS moves
void connected_BTS(BaseTranscieverStation *BTS) {
  connectedBTS = BTS; // assigning the connectedBTs to the BTs we will get
from the function
}
static void numberOfDevices() {
  cout << "The number of MS devices is: " << ID << endl;
}
void move(double x, double y, double z, BaseStationController* nn, int n);
// the move function that changes the position of the MS and calls the
function update and leave
//function to print the position
void printPos() {
  cout << "X: " << pos[0] << " Y: " << pos[1] << " Z: " << pos[2] << endl;
}//destructor
~MobileStation(){
 }
};
int MobileStation::ID = 0;//intializing the value of the id to 0
//--------------------BaseStationController-----------------------------//
class BaseStationController {
private:
BaseTranscieverStation *numOfBTS; // an array of objects to get the number
of BTSs
static int numBTS;
public:
BaseStationController(){
  numBTS = 0;
  numOfBTS= new BaseTranscieverStation[SIZE]; //intializing the array of
objects
}//function to add the BTSs to the BSC
void Add_BTS(BaseTranscieverStation BTS) {
  if (numBTS < SIZE) {//checking if there is a place to add the the bts
    numOfBTS[numBTS] = BTS;
    numBTS++;
  } else
```



```
    cout << "No Space for more BTSs as the max number of BTSs is "<<SIZE
<< endl; // message if there is no space for new bts
}
void PrintInfo(); // function to print all the information of each bts
BaseTranscieverStation* getAllBTS() {//function to get the btss
  return numOfBTS;
}
static int getNUM(){ // funciton that returns number of bts
  return numBTS;
}
//destructor of the bsc
~BaseStationController(){
}
};
int BaseStationController::numBTS = 0; // intializing the number of bts
//function that connects the MS to BTS
bool BaseTranscieverStation::connectMSToBTS(MobileStation *newMS) {
if (numOfMS >= SLOTS) {// checking if there are free slots or not
  return false;
}else{
  numOfOccupiedSlots[numOfMS] = newMS; //assigning the current position of
the array of objects to the new MS
  newMS->connected_BTS(this);//passing the object to the function to
assign it to the BTS
  numOfMS++;//incrementing the num of MS because we added a new one
  cout << "MS "<<newMS->getID()<<" to BTS with ID: " << GeneratedID
<<endl;
  return true;
}
}
void BaseTranscieverStation::leave_MS_BTS(MobileStation *newMS){
  bool check = false;
  //looping through the objects of the bts
  for(int i = 0; i < (8-num_of_freeslots()); i++ ){
    //cout<<num_of_freeslots()<<endl;
    if(newMS == numOfOccupiedSlots[i]){ // if the newMS is equal to one of
the occupied ms in the bts
      for(int j = 0; j < (8-num_of_freeslots()); j++){//loop through the
occupied objects to shift its position
```



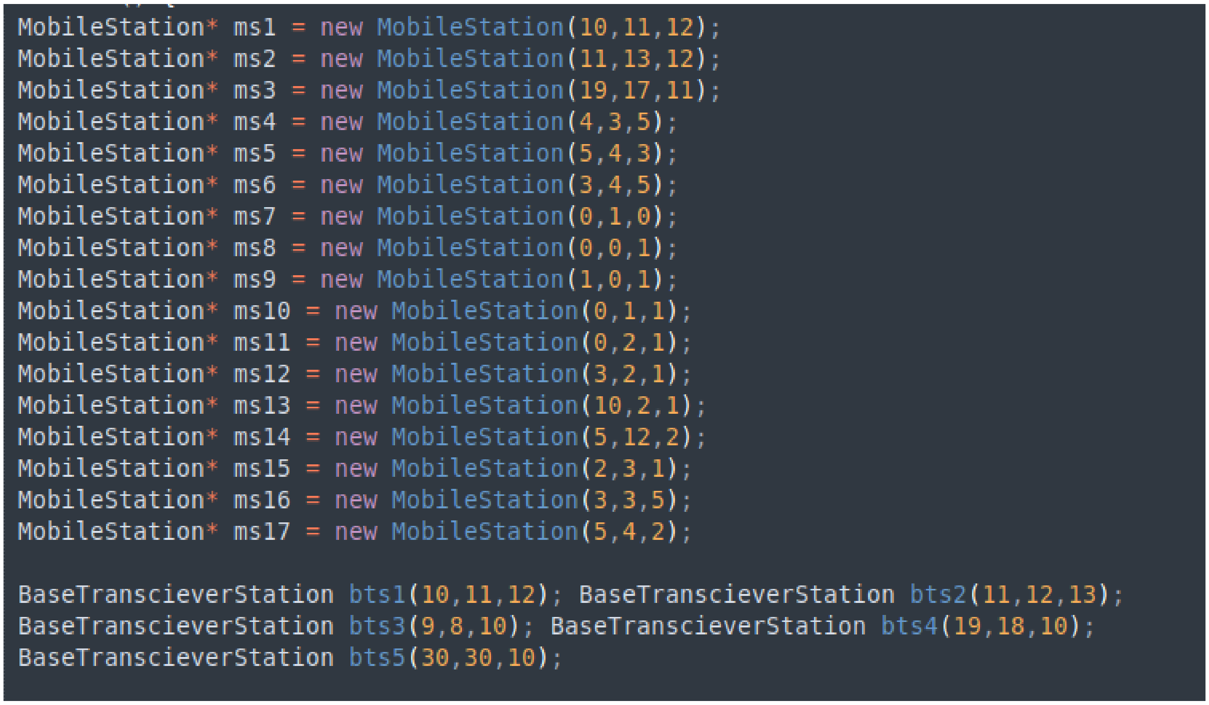
```
        numOfOccupiedSlots[j] = numOfOccupiedSlots[j+1];
      }
      numOfMS--; //decrement the number of the MSs
      check = true;
    }
  }
  if(!check){ // check if there are MSs in the BTS or not
    cout<<"No MS to leave"<<endl;
  }else{//printing a message about which MS left the bts
    cout<<"MS "<< newMS->getID() << " Left the BTS
"<<this->getID_BTS()<<endl;
  }
}
void MobileStation::updateActiveBTS(BaseStationController* BSC, int
numofBTS){
double Dis = 0;
int check = 0;
BaseTranscieverStation* BTS = BSC->getAllBTS(); // getting all the BTSs
double min = 1000000000; // assigning a large number to compare it with
the distance
for (int i = 0; i < numofBTS; i++) {//looping through the number of
objects to see if there is a free space to update the position of ms or
not
  if(BTS[i].num_of_freeslots() == 0){//if there are no free space then
ignore this bts
    continue;
  }
    Dis = sqrt(pow(pos[0] - (BTS[i].getPosBTS()[0]), 2) +
               pow(pos[1] - (BTS[i].getPosBTS()[1]), 2) +
               pow(pos[2] - (BTS[i].getPosBTS()[2]), 2));
    if (Dis < min) {
      min = Dis;
      check = i; //getting the index of the BTS
    }
}//checking if the bts has free space or not
if(BTS[check].connectMSToBTS(this) == false){
  cout<<"The BTS "<<BTS[check].getID_BTS()<<" is full\n";
}
}
```



```
void MobileStation::move(double x, double y, double z,
BaseStationController* newBSC, int n) {
pos[0] = x;
pos[1] = y;
pos[2] = z;
cout<<"MS "<<getID()<<" Moved to: ("<<pos[0]<<" , "<<pos[1]<<" ,
"<<pos[2]<<")"<<endl;
connectedBTS->leave_MS_BTS(this);//checking if it has left the current bts
or not
this->updateActiveBTS(newBSC, n);// updating the position again
}
//print the information of the btss
void BaseStationController::PrintInfo() {
for (int i = 0; i < numBTS; i++) {
  numOfBTS[i].print();
  cout << endl;
}
}
//-----------------MAIN----------------//
int main() {
MobileStation* ms1 = new MobileStation(10,11,12);
MobileStation* ms2 = new MobileStation(11,13,12);
MobileStation* ms3 = new MobileStation(19,17,11);
MobileStation* ms4 = new MobileStation(4,3,5);
MobileStation* ms5 = new MobileStation(5,4,3);
MobileStation* ms6 = new MobileStation(3,4,5);
MobileStation* ms7 = new MobileStation(0,1,0);
MobileStation* ms8 = new MobileStation(0,0,1);
MobileStation* ms9 = new MobileStation(1,0,1);
MobileStation* ms10 = new MobileStation(0,1,1);
MobileStation* ms11 = new MobileStation(0,2,1);
MobileStation* ms12 = new MobileStation(3,2,1);
MobileStation* ms13 = new MobileStation(10,2,1);
MobileStation* ms14 = new MobileStation(5,12,2);
MobileStation* ms15 = new MobileStation(2,3,1);
MobileStation* ms16 = new MobileStation(3,3,5);
MobileStation* ms17 = new MobileStation(5,4,2);
BaseTranscieverStation bts1(10,11,12); BaseTranscieverStation
bts2(11,12,13);
BaseTranscieverStation bts3(9,8,10); BaseTranscieverStation
bts4(19,18,10);
BaseTranscieverStation bts5(30,30,10);
BaseStationController* bsc = new BaseStationController;
bsc->Add_BTS(bts1); bsc->Add_BTS(bts2);
bsc->Add_BTS(bts3); bsc->Add_BTS(bts4);
bsc->Add_BTS(bts5);
ms1->updateActiveBTS(bsc, bsc->getNUM()); ms1->move(18,18,10, bsc,
bsc->getNUM());
ms2->updateActiveBTS(bsc, bsc->getNUM());
ms3->updateActiveBTS(bsc, bsc->getNUM()); ms3->move(10,11,12, bsc,
bsc->getNUM());
ms4->updateActiveBTS(bsc, bsc->getNUM());
ms5->updateActiveBTS(bsc, bsc->getNUM());
ms6->updateActiveBTS(bsc, bsc->getNUM());
ms7->updateActiveBTS(bsc, bsc->getNUM());
ms8->updateActiveBTS(bsc, bsc->getNUM());
ms9->updateActiveBTS(bsc, bsc->getNUM()); ms5->move(21,31,11, bsc,
bsc->getNUM());
ms10->updateActiveBTS(bsc, bsc->getNUM());
ms11->updateActiveBTS(bsc, bsc->getNUM());
ms12->updateActiveBTS(bsc, bsc->getNUM());
ms13->updateActiveBTS(bsc, bsc->getNUM());
ms14->updateActiveBTS(bsc, bsc->getNUM());
ms15->updateActiveBTS(bsc, bsc->getNUM());
ms16->updateActiveBTS(bsc, bsc->getNUM());
ms17->updateActiveBTS(bsc, bsc->getNUM());
cout<<endl;
 bsc->PrintInfo();
 // deallocate the classes
delete ms1; delete ms2;
delete ms3; delete ms4;
delete ms5; delete ms6;
delete ms7; delete ms8;
delete ms9; delete ms10;
delete ms12; delete ms13;
delete ms14; delete ms15;
delete ms16;  delete bsc;
return 0;
}
```

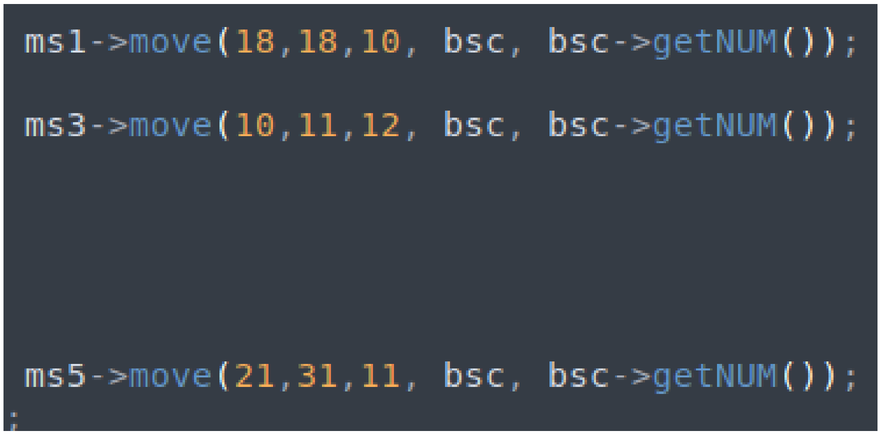
Step 5: Test and verification:

- **Test case 1**

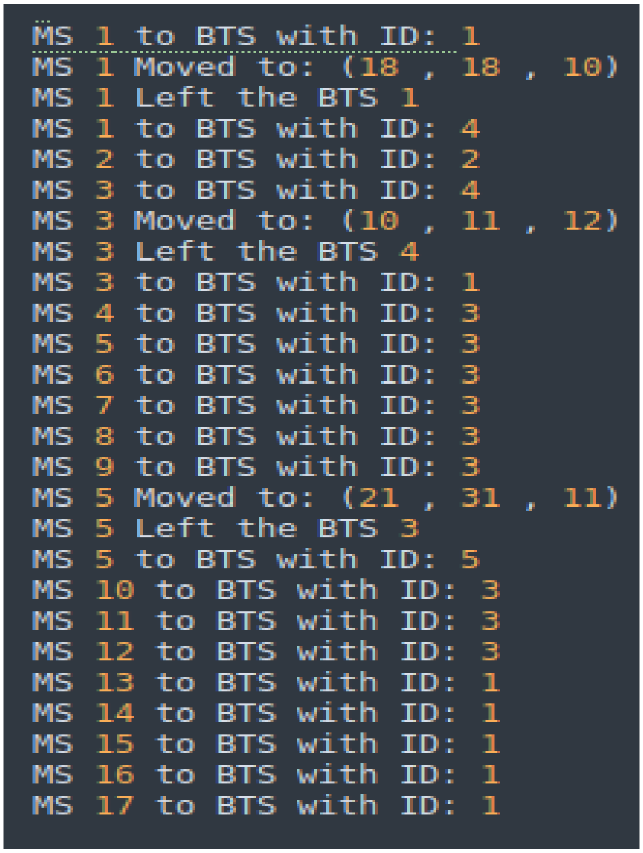


```
MobileStation* ms1 = new MobileStation(10,11,12);
MobileStation* ms2 = new MobileStation(11,13,12);
MobileStation* ms3 = new MobileStation(19,17,11);
MobileStation* ms4 = new MobileStation(4,3,5);
MobileStation* ms5 = new MobileStation(5,4,3);
MobileStation* ms6 = new MobileStation(3,4,5);
MobileStation* ms7 = new MobileStation(0,1,0);
MobileStation* ms8 = new MobileStation(0,0,1);
MobileStation* ms9 = new MobileStation(1,0,1);
MobileStation* ms10 = new MobileStation(0,1,1);
MobileStation* ms11 = new MobileStation(0,2,1);
MobileStation* ms12 = new MobileStation(3,2,1);
MobileStation* ms13 = new MobileStation(10,2,1);
MobileStation* ms14 = new MobileStation(5,12,2);
MobileStation* ms15 = new MobileStation(2,3,1);
MobileStation* ms16 = new MobileStation(3,3,5);
MobileStation* ms17 = new MobileStation(5,4,2);

BaseTranscieverStation bts1(10,11,12); BaseTranscieverStation bts2(11,12,13);
BaseTranscieverStation bts3(9,8,10); BaseTranscieverStation bts4(19,18,10);
BaseTranscieverStation bts5(30,30,10);
```

The output

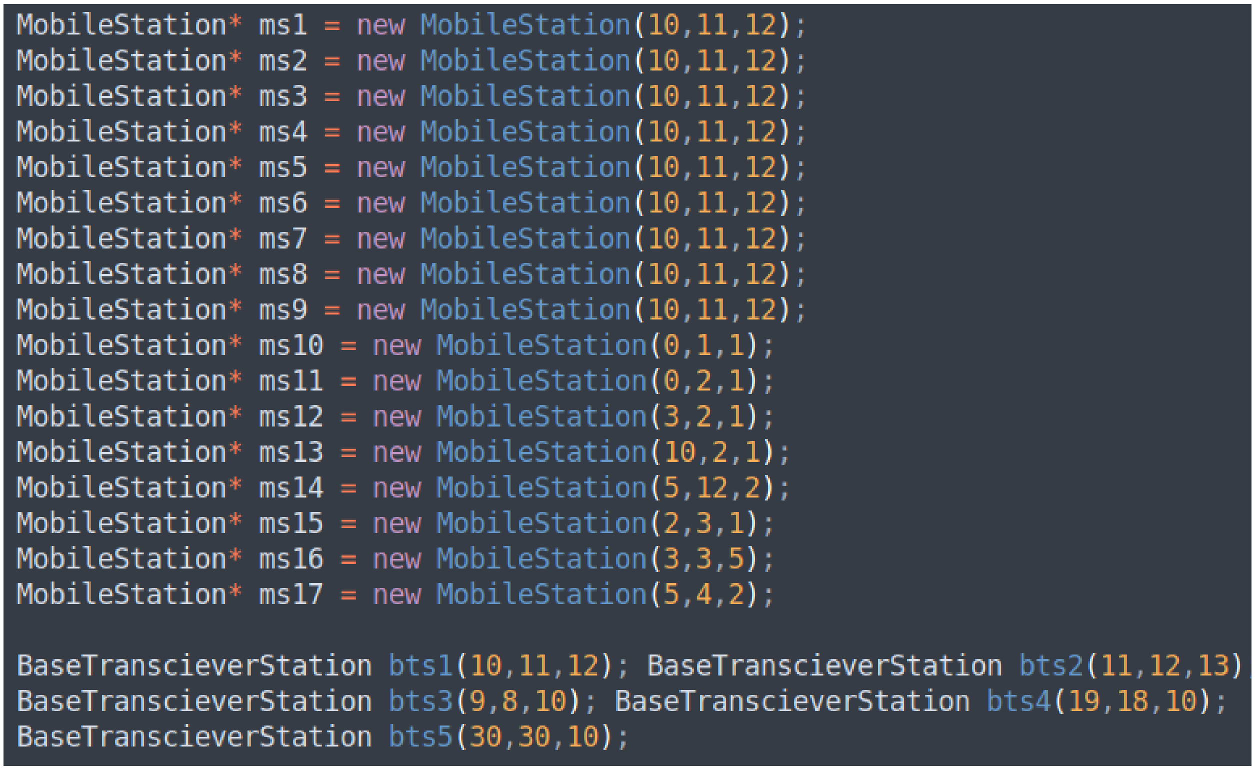


```
ms1->move(18,18,10, bsc, bsc->getNUM());

ms3->move(10,11,12, bsc, bsc->getNUM());




ms5->move(21,31,11, bsc, bsc->getNUM());
```



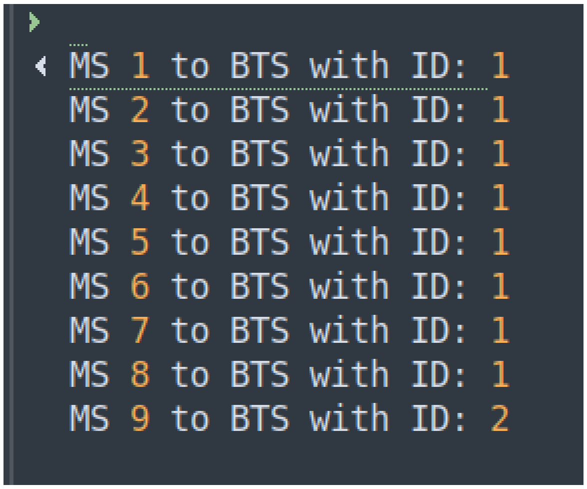
```
MS 1 to BTS with ID: 1
MS 1 Moved to: (18 , 18 , 10)
MS 1 Left the BTS 1
MS 1 to BTS with ID: 4
MS 2 to BTS with ID: 2
MS 3 to BTS with ID: 4
MS 3 Moved to: (10 , 11 , 12)
MS 3 Left the BTS 4
MS 3 to BTS with ID: 1
MS 4 to BTS with ID: 3
MS 5 to BTS with ID: 3
MS 6 to BTS with ID: 3
MS 7 to BTS with ID: 3
MS 8 to BTS with ID: 3
MS 9 to BTS with ID: 3
MS 5 Moved to: (21 , 31 , 11)
MS 5 Left the BTS 3
MS 5 to BTS with ID: 5
MS 10 to BTS with ID: 3
MS 11 to BTS with ID: 3
MS 12 to BTS with ID: 3
MS 13 to BTS with ID: 1
MS 14 to BTS with ID: 1
MS 15 to BTS with ID: 1
MS 16 to BTS with ID: 1
MS 17 to BTS with ID: 1
```

- **Test case 2**



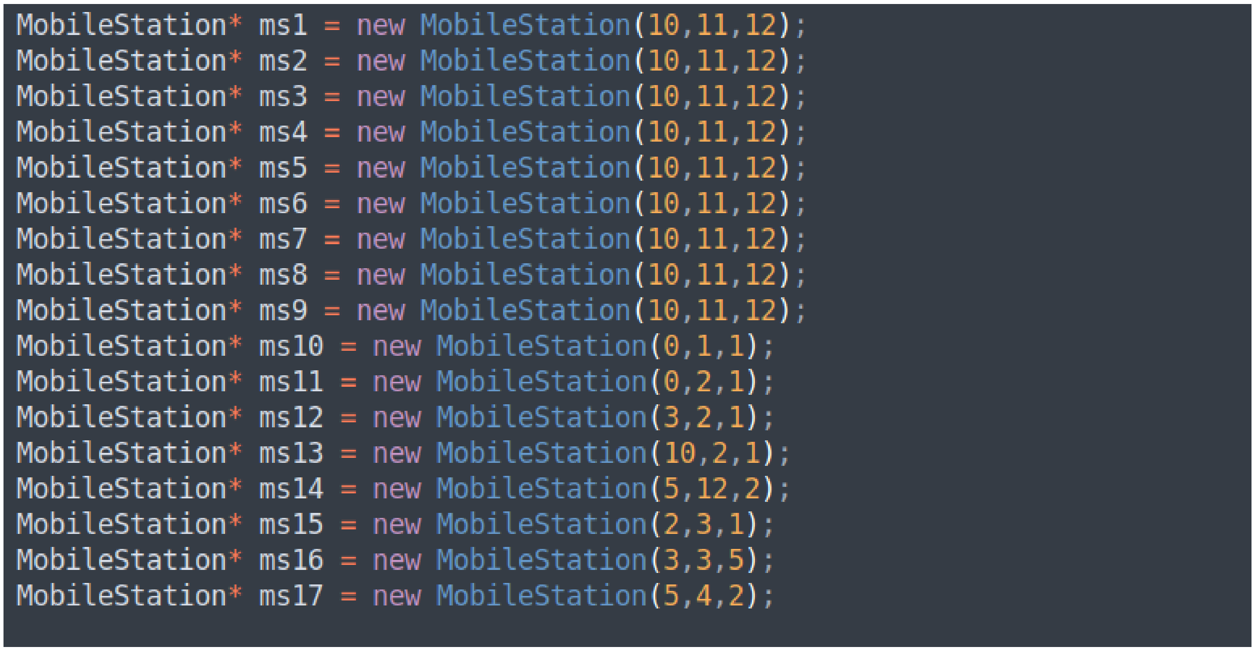
```
MobileStation* ms1 = new MobileStation(10,11,12);
MobileStation* ms2 = new MobileStation(10,11,12);
MobileStation* ms3 = new MobileStation(10,11,12);
MobileStation* ms4 = new MobileStation(10,11,12);
MobileStation* ms5 = new MobileStation(10,11,12);
MobileStation* ms6 = new MobileStation(10,11,12);
MobileStation* ms7 = new MobileStation(10,11,12);
MobileStation* ms8 = new MobileStation(10,11,12);
MobileStation* ms9 = new MobileStation(10,11,12);
MobileStation* ms10 = new MobileStation(0,1,1);
MobileStation* ms11 = new MobileStation(0,2,1);
MobileStation* ms12 = new MobileStation(3,2,1);
MobileStation* ms13 = new MobileStation(10,2,1);
MobileStation* ms14 = new MobileStation(5,12,2);
MobileStation* ms15 = new MobileStation(2,3,1);
MobileStation* ms16 = new MobileStation(3,3,5);
MobileStation* ms17 = new MobileStation(5,4,2);

BaseTranscieverStation bts1(10,11,12); BaseTranscieverStation bts2(11,12,13)
BaseTranscieverStation bts3(9,8,10); BaseTranscieverStation bts4(19,18,10);
BaseTranscieverStation bts5(30,30,10);
```

The output



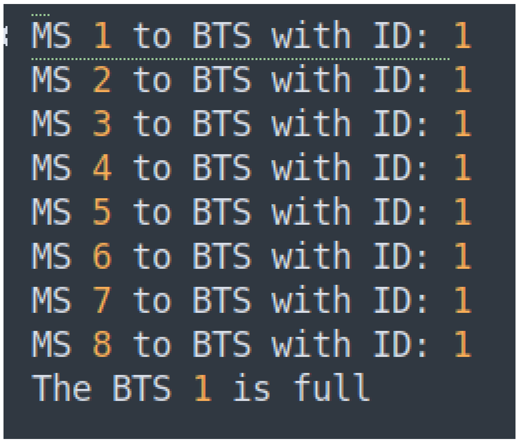
```
MS 1 to BTS with ID: 1
MS 2 to BTS with ID: 1
MS 3 to BTS with ID: 1
MS 4 to BTS with ID: 1
MS 5 to BTS with ID: 1
MS 6 to BTS with ID: 1
MS 7 to BTS with ID: 1
MS 8 to BTS with ID: 1
MS 9 to BTS with ID: 2
```

- **Test case 3:**



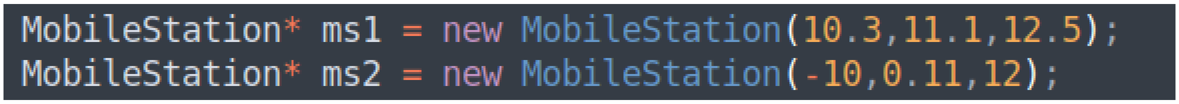
```
MobileStation* ms1 = new MobileStation(10,11,12);
MobileStation* ms2 = new MobileStation(10,11,12);
MobileStation* ms3 = new MobileStation(10,11,12);
MobileStation* ms4 = new MobileStation(10,11,12);
MobileStation* ms5 = new MobileStation(10,11,12);
MobileStation* ms6 = new MobileStation(10,11,12);
MobileStation* ms7 = new MobileStation(10,11,12);
MobileStation* ms8 = new MobileStation(10,11,12);
MobileStation* ms9 = new MobileStation(10,11,12);
MobileStation* ms10 = new MobileStation(0,1,1);
MobileStation* ms11 = new MobileStation(0,2,1);
MobileStation* ms12 = new MobileStation(3,2,1);
MobileStation* ms13 = new MobileStation(10,2,1);
MobileStation* ms14 = new MobileStation(5,12,2);
MobileStation* ms15 = new MobileStation(2,3,1);
MobileStation* ms16 = new MobileStation(3,3,5);
MobileStation* ms17 = new MobileStation(5,4,2);
```

The output

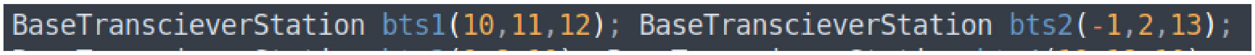


```
MS 1 to BTS with ID: 1
MS 2 to BTS with ID: 1
MS 3 to BTS with ID: 1
MS 4 to BTS with ID: 1
MS 5 to BTS with ID: 1
MS 6 to BTS with ID: 1
MS 7 to BTS with ID: 1
MS 8 to BTS with ID: 1
The BTS 1 is full
```

- **Test case 4:**

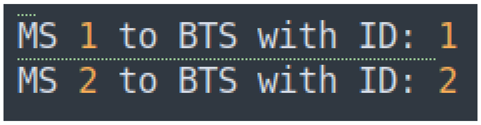


```
MobileStation* ms1 = new MobileStation(10.3,11.1,12.5);
MobileStation* ms2 = new MobileStation(-10,0.11,12);
```



```
BaseTranscieverStation bts1(10,11,12); BaseTranscieverStation bts2(-1,2,13);
```

The output



```
MS 1 to BTS with ID: 1
MS 2 to BTS with ID: 2
```

- **Verification:**
  - **To test the program, the user has to enter the coordinates in the main manually.**
  - **The MAX number of BTSs is 10000**
  - **The MAX number of SLOTS in the BTS is 8**
  - **All the getter and the setter are tested within the program.**