DOMAIN WINTER WINNING CAMP

Student Name: Dhairyaa **UID: 22BCS15785**

Branch: BE CSE **Section/Group: TPP_FL_603-B**

DAY 1:

VERY EASY

QUES 1: Sum of Natural Numbers up to N

Calculate the sum of all natural numbers from 1 to n, where n is a positive integer. Use the formula: Sum=n×(n+1)/2.

Take n as input and output the sum of natural numbers from 1 to n.

Solution:

```
#include <iostream>

using namespace std; int

main() {

   int n;     cout << "Enter a positive integer: ";     cin >> n;     if (n > 0) {         int sum

= n * (n + 1) / 2;         cout << "The sum of natural numbers from 1 to " << n << " is: "

<< sum << endl;

   } else {         cout << "Please enter a positive

integer." << endl;

   }

return 0;

}
```

```
Enter a positive integer: 32
The sum of natural numbers from 1 to 32 is: 528
```

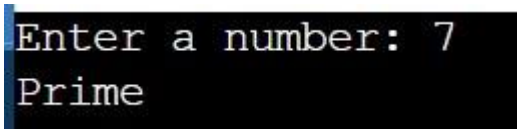
QUES 2: Check if a Number is Prime

Objective Check if a given number n is a prime number. A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself. To determine if a number is

prime, iterate from 2 to √n and check if n is divisible by any number in this range. If it is divisible, it is not a prime number; otherwise, it is a prime.

Solution:

```
#include <iostream> #include
<cmath> using namespace std;
bool isPrime(int n) {     if (n <= 1)
return false;      for (int i = 2; i <=
sqrt(n); i++) {        if (n % i == 0)
{            return false;
      }     }
return true;
} int main()
{
   int n;
   cout << "Enter a number: ";
cin >> n;     if (n >= 2 && n <=
100000) {        if (isPrime(n)) {
cout << "Prime" << endl;
      } else {            cout << "Not
Prime" << endl;
      }
   } else {        cout << "Number out of range. Please enter a number between 2 and
100000." << endl; }     return 0;
}
```



QUES 3: Print Multiplication Table of a Number

Objective: Print the multiplication table of a given number n. A multiplication table for a number n is a list of products of n with integers from 1 to 10. For example, the multiplication table for 3 is:
3×1=3,3×2=6,…,3×10=30.

Solution: #include

<iostream> using

namespace std; int

main() {

int n;

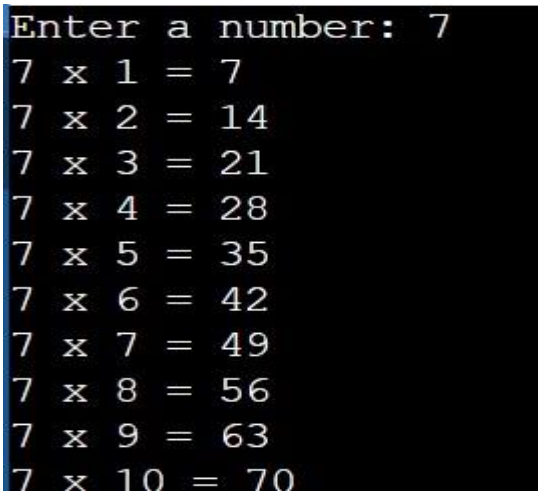
cout << "Enter a number: "; cin >> n; for (int i

= 1; i <= 10; i++) { cout << n << " x " << i << "

= " << n * i << endl;

}

return 0;}



QUES 4: Print Odd Numbers up to N

Objective: Print all odd numbers between 1 and n, inclusive. Odd numbers are integers that are not divisible by 2. These numbers should be printed in ascending order, separated by spaces. This problem is a simple introduction to loops and conditional checks. The goal is to use a loop to iterate over the numbers and check if they are odd using the condition i%2≠0.

Solution: #include

<iostream> using

namespace std; int

main() {

int n;

cout << "Enter a number: ";

cin >> n; for (int i = 1; i <=

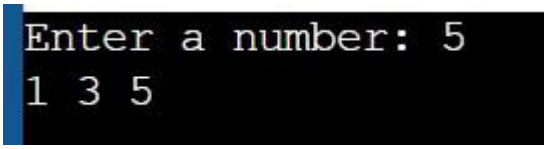
n; i++) { if (i % 2 != 0) {

cout << i << " ";

} }

return 0;

}



QUES 5: Sum of Odd Numbers up to N

Objective: Calculate the sum of all odd numbers from 1 to n. An odd number is an integer that is not divisible by 2.The sum of odd numbers, iterate through all the numbers from 1 to n, check if each number is odd, and accumulate the sum.

Solution: #include

<iostream> using

namespace std; int

main() { int n,

sum = 0; cout <<

"Enter a number: ";

cin >> n; for (int i

= 1; i <= n; i++) {

```
if (i % 2 != 0) {
sum += i;
      }
   }
   cout << "Sum of odd numbers up to " << n << " is: " << sum << endl;
   return 0;
}
```

```
Enter a number: 5
Sum of odd numbers up to 5 is: 9
```

EASY

QUES 6: Count Digits in a Number

Objective: Count the total number of digits in a given number n. The number can be a positive integer. For example, for the number 12345, the count of digits is 5. For a number like 900000, the count of digits is 6.

Given an integer n, your task is to determine how many digits are present in n. This task will help you practice working with loops, number manipulation, and conditional logic.

Solution:

```
#include <iostream>
using namespace std; int
main() {
   int n;     cout << "Enter a positive
integer: ";     cin >> n;     if (n > 0) {
int count = 0;
      while (n > 0) {
n /= 10;
count++;
```

```
        }
        cout << "The number of digits is: " << count << endl;
    } else {        cout << "Please enter a positive
integer." << endl;
    }
return 0;
}
```

```
Enter a positive integer: 67353428
The number of digits is: 8
```

QUES 7: Reverse a Number

Objective: Reverse the digits of a given number n. For example, if the input number is 12345, the output should be 54321. The task involves using loops and modulus operators to extract the digits and construct the reversed number.

Solution:

```
#include <iostream> using namespace std; int main() {     int
n, reversedNumber = 0;     cout << "Enter a number: ";     cin
>> n;     while (n > 0) {         int digit = n % 10;
reversedNumber = reversedNumber * 10 + digit; number
        n /= 10;
    }
    cout << "Reversed Number: " << reversedNumber << endl;
    return 0;
}
```

```
Enter a number: 12345
Reversed Number: 54321
```

QUES 8: Find the Largest Digit in a Number

Objective: Find the largest digit in a given number n. For example, for the number 2734, the largest digit is 7. You need to extract each digit from the number and determine the largest one. The task will involve using loops and modulus operations to isolate the digits.

Solution: #include

<iostream> using namespace

std; int main() { int n,

largestDigit = 0; cout <<

"Enter a number: "; cin >>

n; while (n > 0) { int

digit = n % 10; if (digit

> largestDigit) {

largestDigit = digit;

}

n /= 10;

}

cout << "Largest Digit: " << largestDigit << endl;

return 0;

}

```
Enter a number: 37937670
Largest Digit: 9
```

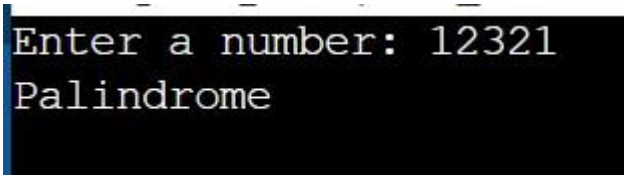
QUES 9: Check if a Number is a Palindrome

Objective: Check whether a given number is a palindrome or not. A number is called a palindrome if it reads the same backward as forward. For example, 121 is a palindrome because reading it from left to right is the same as reading it from right to left. Similarly, 12321 is also a palindrome, but 12345 is not.

Solution: #include <iostream> using namespace std;

int main() { int n, originalNumber,

```
reversedNumber = 0;     cout << "Enter a number: ";
cin >> n;     originalNumber = n;     while (n > 0) {
int digit = n % 10;         reversedNumber =
reversedNumber * 10 + digit;
        n /= 10;
    }
    if (originalNumber == reversedNumber) {
cout << "Palindrome" << endl;
    } else {         cout << "Not a Palindrome"
<< endl;    }     return 0;
}
```



QUES 10: Find the Sum of Digits of a Number

Objective: Calculate the sum of the digits of a given number n. For example, for the number 12345, the sum of the digits is 1+2+3+4+5=15. To solve this, you will need to extract each digit from the number and calculate the total sum.

Solution: #include

<iostream> using namespace

std; int main() {   int n, sum

= 0;   cout << "Enter a

number: ";   cin >> n;

while (n > 0) {   int digit =

n % 10;   sum += digit;

n /= 10;

    }

```
    cout << "Sum of digits: " << sum << endl;
return 0;
}
```

```
Enter a number: 478758
Sum of digits: 39
```

MEDIUM

QUES 11: Function Overloading for Calculating Area.

Objective: Write a program to calculate the area of different shapes using function overloading. Implement overloaded functions to compute the area of a circle, a rectangle, and a triangle.

Solution: #include

<iostream> using

namespace std;

```
double calculateArea(double radius) {
    return 3.14159 * radius * radius;
}
double calculateArea(double length, double breadth) {
return length * breadth;
}
double calculateArea(double base, double height, int isTriangle) {
return 0.5 * base * height;
} int main() {     double radius, length, breadth, base, height;     cout << "Enter
the radius of the circle: ";     cin >> radius;     cout << "Area of the circle: " <<
calculateArea(radius) << endl;     cout << "Enter the length and breadth of the
rectangle: ";     cin >> length >> breadth;     cout << "Area of the rectangle: "
<< calculateArea(length, breadth) << endl;     cout << "Enter the base and
```

height of the triangle: "; cin >> base >> height; cout << "Area of the

triangle: " << calculateArea(base, height, 1) << endl; return 0;

}

```
Enter the radius of the circle: 6
Area of the circle: 113.097
Enter the length and breadth of the rectangle: 7
8
Area of the rectangle: 56
Enter the base and height of the triangle: 9
10
Area of the triangle: 45
```

QUES 12: Encapsulation with Employee Details

Objective: Write a program that demonstrates encapsulation by creating a class Employee. The class should have private attributes to store:

Employee ID.

Employee Name.

Employee Salary.

Provide public methods to set and get these attributes, and a method to display all details of the employee.

Solution:

#include <iostream>

#include <string> using

namespace std; class

Employee { private: int

employeeID; string

employeeName; double

employeeSalary; public:

```
    void setEmployeeID(int id) {
employeeID = id;
    }
    void setEmployeeName(string name) {
employeeName = name;
    }
    void setEmployeeSalary(double salary) {
employeeSalary = salary;
    }
    int getEmployeeID() const {
return employeeID;
    }
    string getEmployeeName() const {
return employeeName;
    }
    double getEmployeeSalary() const {
return employeeSalary;
    }
    void displayDetails() const {         cout << "Employee ID: "
<< employeeID << endl;         cout << "Employee Name: " <<
employeeName << endl;         cout << "Employee Salary: $" <<
employeeSalary << endl;
    } }; int main() {     Employee emp;
emp.setEmployeeID(101);
emp.setEmployeeName("John Doe");
emp.setEmployeeSalary(55000.50);
```

```
cout << "Employee Details:" << endl;

emp.displayDetails();     return 0;

}
```

```
Employee Details:
Employee ID: 101
Employee Name: John Doe
Employee Salary: $55000.5
```

QUES 13: Inheritance with Student and Result Classes.

Objective: Create a program that demonstrates inheritance by defining:

- A base class Student to store details like Roll Number and Name.
- A derived class Result to store marks for three subjects and calculate the total and percentage.

Solution:

```
#include <iostream> #include

<string> using namespace std;

class Student { protected:     int

rollNumber;     string name;

public:     void setDetails(int r,

string n) {         rollNumber = r;

name = n;

   }

   void displayDetails() const {         cout << "Roll

Number: " << rollNumber << endl;         cout <<

"Name: " << name << endl;

   } };

class Result : public Student { private:

   float marks[3];  public:     void

setMarks(float m1, float m2, float m3) {
```

```
marks[0] = m1;        marks[1] = m2;
marks[2] = m3;
   }     float calculateTotal()
const {        return marks[0] +
marks[1] + marks[2];

   }
   float calculatePercentage() const {        return (calculateTotal() / 300) * 100;
// Assuming each subject is out of 100
   }
   void displayResult() const {        displayDetails(); // Call base class method
cout << "Marks: " << marks[0] << ", " << marks[1] << ", " << marks[2] << endl;
cout << "Total Marks: " << calculateTotal() << endl;        cout << "Percentage: " <<
calculatePercentage() << "%" << endl;
   } }; int main() {     Result student;
student.setDetails(101, "Alice");
student.setMarks(85, 90, 88);     cout <<
"Student Result Details:" << endl;
student.displayResult();     return 0;
}
```

```
Student Result Details:
Roll Number: 101
Name: Alice
Marks: 85, 90, 88
Total Marks: 263
Percentage: 87.6667%
```

QUES 14: Polymorphism with Shape Area Calculation.

Objective: Create a program that demonstrates polymorphism by calculating the area of different shapes using a base class Shape and derived classes for Circle, Rectangle, and Triangle. Each derived class should override a virtual function to compute the area of the respective shape.

Solution:

```
#include <iostream> #include

<cmath> using namespace std;

class Shape { public:     virtual

double getArea() = 0;

virtual ~Shape() {}

};

class Circle : public Shape { private:

   double radius; public:

   Circle(double r) : radius(r) {}     double getArea()

override {        return M_PI * radius * radius;  // Area =

π * radius²

   } };

class Rectangle : public Shape { private:

   double length, breadth; public:

   Rectangle(double l, double b) : length(l), breadth(b) {}

double getArea() override {        return length * breadth;

// Area = length × breadth

   } };

class Triangle : public Shape { private:

   double base, height; public:

   Triangle(double b, double h) : base(b), height(h) {}

double getArea() override {        return 0.5 * base * height;

// Area = ½ × base × height
```

```
    } }; int
main() {
    Shape* shapes[3];        shapes[0] = new Circle(5);          shapes[1] = new
Rectangle(4, 6);      shapes[2] = new Triangle(3, 7);       for (int i = 0; i < 3; i++)
{         cout << "Area of shape " << i + 1 << ": " << shapes[i]->getArea() <<
endl;
    }     for (int i = 0; i < 3;
i++) {         delete shapes[i];
    }
return 0;
}
```

```
Area of shape 1: 78.5398
Area of shape 2: 24
Area of shape 3: 10.5
```

HARD

QUES 15: Matrix Multiplication Using Function Overloading

Objective: Implement matrix operations in C++ using function overloading. Write a function operate() that can perform:

- Matrix Addition for matrices of the same dimensions.
- Matrix Multiplication where the number of columns of the first matrix equals the number of rows of the second matrix.

Solution:

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

```
void printMatrix(const vector<vector<int>>& matrix) {
for (const auto& row : matrix) {        for (int elem :
row) {            cout << elem << " ";
      }
      cout << endl;
   }
   cout << endl;
} vector<vector<int>> operate(const vector<vector<int>>& mat1, const
vector<vector<int>>& mat2) {    int rows = mat1.size();    int cols = mat1[0].size();
vector<vector<int>> result(rows, vector<int>(cols, 0));    for (int i = 0; i < rows; i++) {
for (int j = 0; j < cols; j++) {            result[i][j] = mat1[i][j] + mat2[i][j];
      }    }
return result;
}
vector<vector<int>> operate(const vector<vector<int>>& mat1, const vector<vector<int>>&
mat2, bool multiply) {    int rows = mat1.size();
   int cols = mat2[0].size();    int common =
mat1[0].size();    vector<vector<int>> result(rows,
vector<int>(cols, 0));    for (int i = 0; i < rows; i++) {
for (int j = 0; j < cols; j++) {            for (int k = 0; k <
common; k++) {                result[i][j] += mat1[i][k] *
mat2[k][j];
         }
      }    }
return result;
} int main() {    vector<vector<int>> mat1 = {{1, 2}, {3, 4}};
vector<vector<int>> mat2 = {{5, 6}, {7, 8}};    vector<vector<int>>
```

```
mat3 = {{1, 2, 3}, {4, 5, 6}};     cout << "Matrix 1:" << endl;
printMatrix(mat1);     cout << "Matrix 2:" << endl;     printMatrix(mat2);
cout << "Matrix 3:" << endl;     printMatrix(mat3);     cout << "Matrix
Addition (mat1 + mat2):" << endl;     vector<vector<int>>
additionResult = operate(mat1, mat2);     printMatrix(additionResult);
cout << "Matrix Multiplication (mat1 x mat3):" << endl;
vector<vector<int>> multiplicationResult = operate(mat1, mat3, true);
printMatrix(multiplicationResult);     return 0;
}
```

```
Matrix Addition (mat1 + mat2):
6 8
10 12

Matrix Multiplication (mat1 x mat3):
9 12 15
19 26 33
```

QUES 17: Polymorphism in Shape Classes

Objective: Design a C++ program using polymorphism to calculate the area of different shapes:

A Rectangle (Area = Length × Breadth).

A Circle (Area = $\pi \times \text{Radius}^2$).

A Triangle (Area = ½ × Base × Height).

Create a base class Shape with a pure virtual function getArea(). Use derived classes Rectangle, Circle, and Triangle to override this function.

Solution:

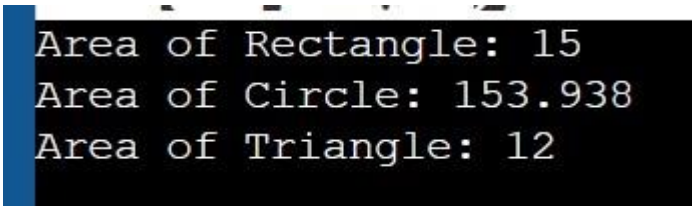
#include <iostream> #include

<cmath> using namespace std; class

```
Shape { public:       virtual double
getArea() const = 0;      virtual
~Shape() {}
};
class Rectangle : public Shape {
private:    double length,
breadth; public:

   Rectangle(double l, double b) : length(l), breadth(b) {}
double getArea() const override {        return length *
breadth;
   } };
class Circle : public Shape { private:
   double radius; public:
   Circle(double r) : radius(r) {}
double getArea() const override {
return M_PI * radius * radius;
   } };
class Triangle : public Shape { private:
   double base, height; public:
   Triangle(double b, double h) : base(b), height(h) {}
double getArea() const override {        return 0.5 *
base * height;
   } }; int main() {       Shape*
shapes[3];      shapes[0] = new
Rectangle(5, 3);    shapes[1] =
new Circle(7); shapes[2] = new
```

```
Triangle(4, 6); cout << "Area of
Rectangle: " << shapes[0]-
>getArea() << endl;     cout <<
"Area of Circle: " << shapes[1]-
>getArea() << endl;     cout <<
"Area of Triangle: " << shapes[2]-
>getArea() << endl;     for (int i =
0; i < 3; ++i) {         delete
shapes[i];
   }
return 0;
}
```



VERY HARD

QUES 18: Implement Polymorphism for Banking Transactions

Objective: Design a C++ program to simulate a banking system using polymorphism. Create a base class Account with a virtual method calculateInterest(). Use the derived classes SavingsAccount and CurrentAccount to implement specific interest calculation logic:

- SavingsAccount: Interest = Balance × Rate × Time.
- CurrentAccount: No interest, but includes a maintenance fee deduction.

Solution: #include

```
<iostream> using
namespace std; class
Account { protected:
```

```
    double balance; public:
    Account(double bal) : balance(bal) {}
virtual void calculateInterest() = 0; void
displayBalance() const {        cout <<
"Balance: $" << balance << endl;
    }
    virtual ~Account() {}
};
class SavingsAccount : public Account {
private:     double interestRate;
double time;        public:
    SavingsAccount(double bal, double rate, double t) : Account(bal), interestRate(rate),
time(t) {}     void calculateInterest() override {        double interest = balance *
(interestRate / 100) * time;        balance += interest;          cout << "Interest calculated
for Savings Account: $" << interest << endl;
    }
};
class CurrentAccount : public Account { private:
    double maintenanceFee;  public:
    CurrentAccount(double bal, double fee) : Account(bal), maintenanceFee(fee) {}
void calculateInterest() override {        balance -= maintenanceFee;
       cout << "Maintenance fee deducted for Current Account: $" << maintenanceFee <<
endl;
    } }; int
main() {
Account*
savingsAcc
```

```
= new
SavingsAc
count(1000
0, 5, 2);
savingsAcc
-
>displayBa
lance();
savingsAcc
-
>calculateI
nterest();
savingsAcc
-
>displayBa
lance();
cout <<
endl;
   Account* currentAcc = new CurrentAccount(5000, 50);
currentAcc->displayBalance();     currentAcc-
>calculateInterest();     currentAcc->displayBalance();
delete savingsAcc;     delete currentAcc;     return 0;
}
```

```
Balance: $10000
Interest calculated for Savings Account: $1000
Balance: $11000

Balance: $5000
Maintenance fee deducted for Current Account: $50
Balance: $4950
```

QUES 19: Implement Polymorphism for Banking Transactions

Objective: Design a C++ program to simulate a banking system using polymorphism. Create a base class Account with a virtual method calculateInterest(). Use the derived classes SavingsAccount and CurrentAccount to implement specific interest calculation logic:

SavingsAccount: Interest = Balance × Rate × Time.

CurrentAccount: No interest, but includes a maintenance fee deduction.

Solution: #include

<iostream> using

namespace std; class

Account { protected:

double balance;

public:

Account(double bal) : balance(bal) {}

virtual void calculateInterest() = 0;

virtual ~Account() {} // Virtual destructor

};

class SavingsAccount : public Account {

private: double rate; int time;

public:

SavingsAccount(double bal, double r, int t) : Account(bal), rate(r), time(t) {}

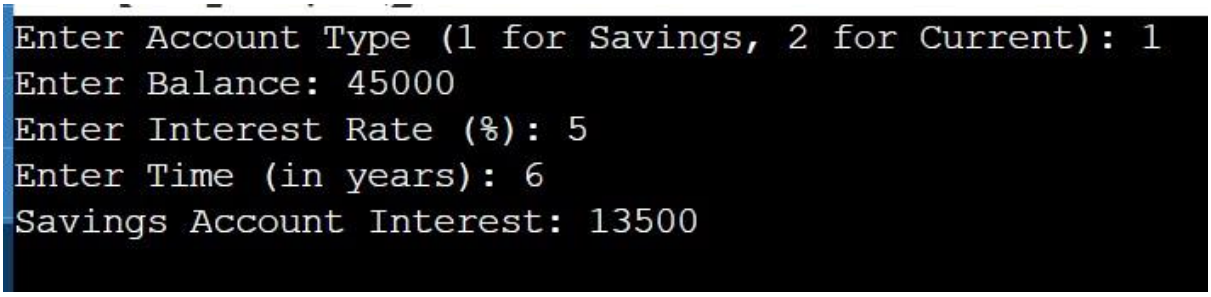
void calculateInterest() override { double interest = balance * (rate / 100)

* time; cout << "Savings Account Interest: " << interest << endl;

```
    }
}; class CurrentAccount : public
Account { private:
    double maintenanceFee; public:
    CurrentAccount(double bal, double fee) : Account(bal), maintenanceFee(fee) {}
void calculateInterest() override {          balance -= maintenanceFee;          cout <<
"Current Account Balance after maintenance fee: " << balance << endl;
    } }; int main() {
int accountType;
cout << "Enter
Account Type (1
for Savings, 2 for
Current): "; cin >>
accountType;     if
(accountType ==
1) {          double
balance, rate;
int time;          cout
<< "Enter
Balance: ";
cin >> balance;
cout << "Enter
Interest Rate (%):
";          cin >> rate;
cout << "Enter
Time (in years): ";
```

```
cin >> time;
if (balance >=
1000 && rate >= 1
&& rate <= 15
&& time >= 1 &&
time <= 10) {
SavingsAccount
sa(balance, rate,
time);
sa.calculateInterest
();
      } else {            cout << "Invalid input for Savings
Account." << endl;
      }
   } else if (accountType == 2) {        double balance, fee;
cout << "Enter Balance: ";        cin >> balance;        cout
<< "Enter Monthly Maintenance Fee: ";        cin >> fee;
if (balance >= 1000 && fee >= 50 && fee <= 500) {
CurrentAccount ca(balance, fee);
ca.calculateInterest();
      } else {            cout << "Invalid input for Current
Account." << endl;
      }
   } else {        cout << "Invalid Account Type. Please enter 1 or
2." << endl;
```

```
    }
return 0;
}
```


Enter Account Type (1 for Savings, 2 for Current): 1
Enter Balance: 45000
Enter Interest Rate (%): 5
Enter Time (in years): 6
Savings Account Interest: 13500