M.VINOD KUMAR

192111625

1.Write a program that prompts the user to enter a positive integer. Use a do-while loop to validate the input

#include <iostream>

int main() {

int userInput;

do {

std::cout << "Enter a positive integer: ";

std::cin >> userInput;

if (userInput <= 0) {

std::cout << "Invalid input. Please enter a positive integer." << std::endl;

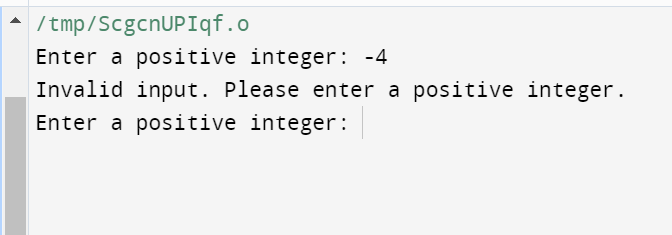
}

} while (userInput <= 0);

std::cout << "You entered a valid positive integer: " << userInput << std::endl;

return 0;

}



**2.The program should randomly select a number between 1 and 100, and then it prompts the user to guess the number**

#include<iostream>

using namespace std;

attempt=0;

int main(){

int secreate ,num;

cout<<"enter the number to limit :";

cin>>"number is :">>num;

cout<<"secreat number is";

cin>>secreate;

while(true){

cout<<("enter the gussing number(1 to 100)");

attempt++;

if(secreate==num){

cout << "Congratulations! You guessed the number in " << attempts << " attempts." << std::endl;

break;

} else {

std::cout << "Incorrect guess. Try again." << std::endl;

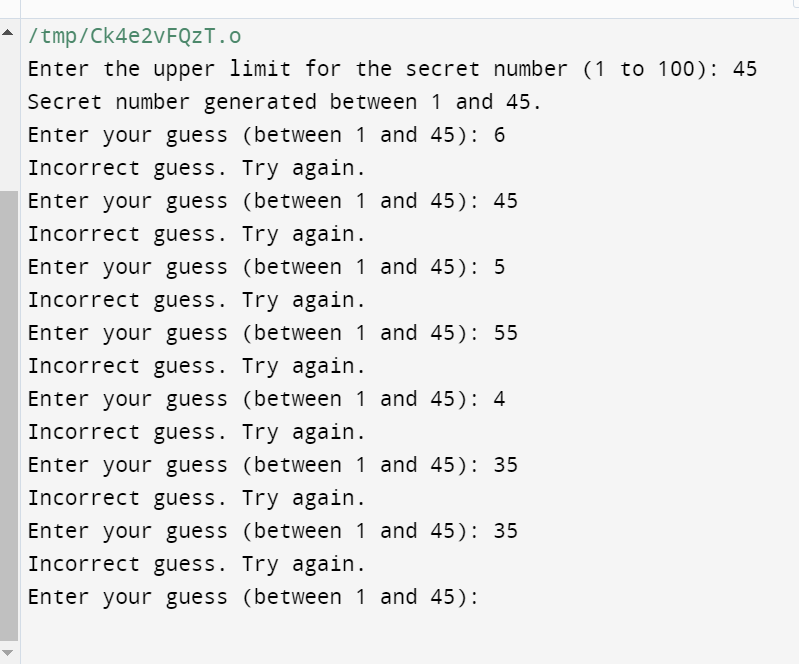
}

}

}

}

OUTPUT:



**3.The competition has three events, and each participant can score between 0 to 100 points in each event. Your task is to write a C++ function to determine the highest score among the three events for a given participant**

#include <iostream>

#include <algorithm> // Needed for std::max

int findHighestScore(int event1, int event2, int event3) {

// Using std::max to find the maximum of the three scores

return std::max({event1, event2, event3});

}

int main() {

// Example usage

int event1Score, event2Score, event3Score;

for(int i=0;i<=2;i++){

// Get scores from the user or any other source

std::cout << "Enter score for event 1: ";

std::cin >> event1Score;

std::cout << "Enter score for event 2: ";

std::cin >> event2Score;

std::cout << "Enter score for event 3: ";

std::cin >> event3Score;

// Find and display the highest score

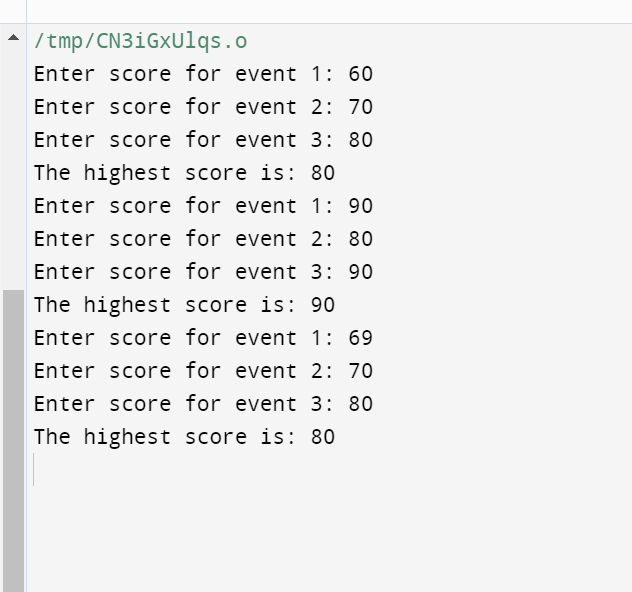
int highestScore = findHighestScore(event1Score, event2Score, event3Score);

std::cout << "The highest score is: " << highestScore << std::endl;

}

return 0;

}



**4. Implement function overloading to handle three types of discounts:**

**A fixed amount discount off the total purchase price.**

**A percentage discount off the total purchase price.**

**A discount for bulk purchases: if the number of items is above a certain threshold, apply a fixed discount per item.**

#include <iostream>

#include <iomanip>

// Function to calculate the total discounted price with fixed amount discount

double calculateDiscountedPriceFixedAmount(double totalPrice, double fixedAmountDiscount) {

// Apply fixed amount discount

return totalPrice - fixedAmountDiscount;

}

// Function to calculate the total discounted price with percentage discount

double calculateDiscountedPricePercentage(double totalPrice, double percentageDiscount) {

// Apply percentage discount

return totalPrice - (totalPrice \* percentageDiscount / 100.0);

}

// Function to calculate the total discounted price with bulk purchase discount

double calculateDiscountedPriceBulk(double totalPrice, double bulkDiscountPerItem, int itemCount, int itemCountThreshold) {

// Apply bulk purchase discount for each item if above the threshold

if (itemCount > itemCountThreshold) {

return totalPrice - (bulkDiscountPerItem \* (itemCount - itemCountThreshold));

} else {

return totalPrice;

}

}

int main() {

// Example usage

double originalPrice, fixedAmountDiscount, percentageDiscount, bulkDiscountPerItem;

int itemCount, itemCountThreshold;

// Get input from the user or any other source

std::cout << "Enter the original price: $";

std::cin >> originalPrice;

std::cout << "Enter the fixed amount discount: $";

std::cin >> fixedAmountDiscount;

std::cout << "Enter the percentage discount: %";

std::cin >> percentageDiscount;

std::cout << "Enter the bulk discount per item: $";

std::cin >> bulkDiscountPerItem;

std::cout << "Enter the item count: ";

std::cin >> itemCount;

std::cout << "Enter the item count threshold for bulk discount: ";

std::cin >> itemCountThreshold;

// Calculate and display the discounted prices using distinct function names

std::cout << std::fixed << std::setprecision(2);

std::cout << "Original Price: $" << originalPrice << std::endl;

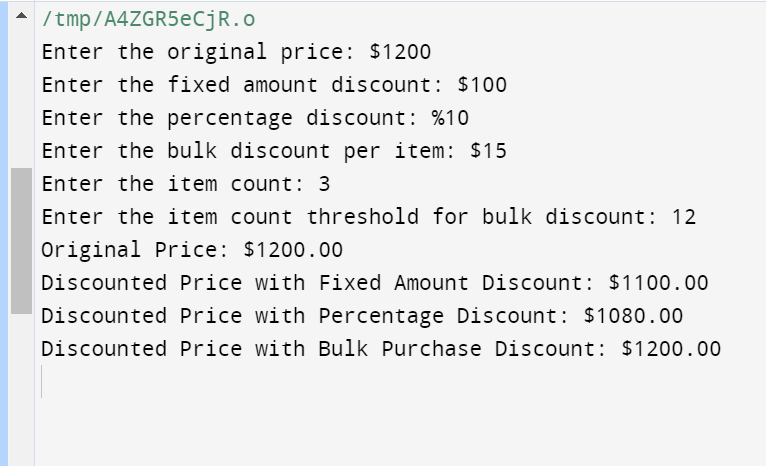
std::cout << "Discounted Price with Fixed Amount Discount: $" << calculateDiscountedPriceFixedAmount(originalPrice, fixedAmountDiscount) << std::endl;

std::cout << "Discounted Price with Percentage Discount: $" << calculateDiscountedPricePercentage(originalPrice, percentageDiscount) << std::endl;

std::cout << "Discounted Price with Bulk Purchase Discount: $" << calculateDiscountedPriceBulk(originalPrice, bulkDiscountPerItem, itemCount, itemCountThreshold) << std::endl;

return 0;

}



#include <iostream>

#include <string>

struct Character {

int health;

int experience;

std::string inventoryItem;

};

void collectExperience(Character& character, int experiencePoints) {

// Use the addition operator to increase experience

character.experience += experiencePoints;

std::cout << "Collected " << experiencePoints << " experience points. Total experience: " << character.experience << std::endl;

}

void collectItem(Character& character, const std::string& item) {

// Use the assignment operator to update the inventory item

character.inventoryItem = item;

std::cout << "Collected item: " << item << std::endl;

}

void encounterTrap(Character& character, int trapDamage) {

// Use the subtraction operator to decrease health

character.health -= trapDamage;

std::cout << "Encountered a trap! Health decreased by " << trapDamage << ". Current health: " << character.health << std::endl;

}

void useHealthBoost(Character& character, int boostAmount) {

// Use the addition operator to increase health

character.health += boostAmount;

std::cout << "Used health boost item. Health increased by " << boostAmount << ". Current health: " << character.health << std::endl;

}

void makeDecision(const Character& character) {

// Use logical operators to make decisions based on health and experience levels

if (character.health <= 0) {

std::cout << "Character has no health. Game over!" << std::endl;

} else if (character.experience >= 100) {

std::cout << "Character has enough experience to level up!" << std::endl;

} else {

std::cout << "Character is still in the game." << std::endl;

}

}

int main() {

// Create a character

Character player = {100, 0, ""};

// Actions

collectExperience(player, 50);

collectItem(player, "Sword");

encounterTrap(player, 20);

useHealthBoost(player, 30);

makeDecision(player);

return 0;

}