A company must implement a tax calculation system. The system should calculate the tax amount for each employee based on their annual salary. The tax rates vary based on the following rules:

1. Employees with a salary up to rs. 10000 pay no tax   
2. Employees who earn between rs. 10000 and rs. 50000 per year are subject to a 10%tax for the salary amount that is above rs. 10000   
3. Employees who earn between rs. 50000 and rs. 100000 are subject to a 20% tax on the portion of their salary that exceeds rs. 50000 and a rs. 4000 fixed tax   
4. Employees who earn above rs. 100000 pay a 30% tax on the portion of the salary exceeding rs. 100000, plus a fixed tax amount of rs. 12000

Write a C program that takes each employee's annual salary as input and calculates the tax amount using the appropriate tax rules. Display the employee's salary, tax rate and tax amount. The program should continue to prompt the user to input of employee salary until the user stops with a value of -1 for the input.

#include <stdio.h>

int main() {

float salary;

while (1) {

printf("Enter the annual salary (-1 to stop): ");

scanf("%f", &salary);

if (salary == -1)

break;

float tax = 0.0;

float taxRate = 0.0;

if (salary <= 10000) {

tax = 0.0;

taxRate = 0.0;

} else if (salary <= 50000) {

tax = (salary - 10000) \* 0.1;

taxRate = 10.0;

} else if (salary <= 100000) {

tax = (salary - 50000) \* 0.2 + 4000;

taxRate = 20.0;

} else {

tax = (salary - 100000) \* 0.3 + 12000;

taxRate = 30.0;

}

printf("Salary: Rs. %.2f\n", salary);

printf("Tax Rate: %.2f%%\n", taxRate);

printf("Tax Amount: Rs. %.2f\n\n", tax);

}

return 0;

}

The voice message is stored in the system as a sequence of numbers. Write a C program to compute the sum of squares of the 1D signal for the elements in the even positions of the voice message and the sum of the cubes of 1D signal for the elements at the odd position of the message. Develop the program such that the above calculations are performed in different modules. The result of the module ‘even square sum()’is added to all the elements in the odd position of the array as a part of encryption. Similarly, the result of ‘odd cube sum()’ is added to the elements in the even position of the array for the encryption process to be strong.

#include <stdio.h>

int even\_square\_sum(int arr[], int size) {

int sum = 0;

for (int i = 1; i < size; i += 2) {

sum += arr[i] \* arr[i];

}

return sum;

}

int odd\_cube\_sum(int arr[], int size) {

int sum = 0;

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

sum += arr[i] \* arr[i] \* arr[i];

}

return sum;

}

int main() {

int signal[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

int size = sizeof(signal) / sizeof(signal[0]);

int even\_square\_result = even\_square\_sum(signal, size);

int odd\_cube\_result = odd\_cube\_sum(signal, size);

printf("Original Signal: ");

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

printf("%d ", signal[i]);

}

printf("\n");

printf("Even Square Sum: %d\n", even\_square\_result);

printf("Odd Cube Sum: %d\n", odd\_cube\_result);

// Encryption process

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

if (i % 2 == 0) {

signal[i] += odd\_cube\_result;

} else {

signal[i] += even\_square\_result;

}

}

printf("Encrypted Signal: ");

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

printf("%d ", signal[i]);

}

printf("\n");

return 0;

}

A ‘player’ class represents the individual players in a game. The ‘player’ class has a static data member ‘total players’ to keep track of the total number of players currently active in the game. Implement the event handling mechanism to update the *total players* count whenever a new player joins or leaves the game. In addition to the ‘player’ class, you have a *game manager* class that manages the overall game progress. The ‘game manager’ class has a static member function ‘checkGameCompletion()’which checks the completion condition of all players reaching a specific score. Implement the event handling mechanism to call the ‘checkGameCompletion()’,whenever a player’s score is updated. Write a C++ code for the above scenario.

#include <iostream>

class GameManager {

public:

static void checkGameCompletion();

};

class Player {

private:

int score;

public:

static int totalPlayers;

Player() {

totalPlayers++;

score = 0;

}

~Player() {

totalPlayers--;

}

void updateScore(int newScore) {

score = newScore;

GameManager::checkGameCompletion();

}

};

int Player::totalPlayers = 0;

void GameManager::checkGameCompletion() {

// Check completion condition (e.g., if all players reached a specific score)

// Your implementation here

std::cout << "Checking game completion..." << std::endl;

}

int main() {

Player p1;

Player p2;

Player p3;

std::cout << "Total Players: " << Player::totalPlayers << std::endl;

p1.updateScore(100);

p2.updateScore(200);

p3.updateScore(300);

std::cout << "Total Players: " << Player::totalPlayers << std::endl;

// Player p4 leaves the game

{

Player p4;

std::cout << "Total Players: " << Player::totalPlayers << std::endl;

}

std::cout << "Total Players: " << Player::totalPlayers << std::endl;

return 0;

}

Create an employee database system in C++ with a class called ‘employee’representing employee details. Implement a function called ‘UpdateSalary()’that is not a part of the employee class, but allows the class named ‘HR Department’ to directly update the employee’s salary which will be having the salary attribute that cannot be accessed outside the class ‘employee’. Write a C++ code to implement the event handling mechanism of updating the employee salary using the static data member concept in C++.

#include <iostream>

class Employee {

public:

int employeeID;

double salary;

static int totalEmployees;

Employee(int id, double initialSalary) : employeeID(id), salary(initialSalary) {

totalEmployees++;

}

~Employee() {

totalEmployees--;

}

friend class HRDepartment;

};

int Employee::totalEmployees = 0;

class HRDepartment {

public:

static void UpdateSalary(Employee& emp, double newSalary) {

emp.salary = newSalary;

}

};

int main() {

Employee emp1(1, 5000);

Employee emp2(2, 7000);

Employee emp3(3, 9000);

std::cout << "Total Employees: " << Employee::totalEmployees << std::endl;

// Update salary using HRDepartment

HRDepartment::UpdateSalary(emp1, 6000);

HRDepartment::UpdateSalary(emp2, 8000);

HRDepartment::UpdateSalary(emp3, 10000);

std::cout << "Employee 1 Salary: " << emp1.salary << std::endl;

std::cout << "Employee 2 Salary: " << emp2.salary << std::endl;

std::cout << "Employee 3 Salary: " << emp3.salary << std::endl;

std::cout << "Total Employees: " << Employee::totalEmployees << std::endl;

return 0;

}

Write a C++ program that gets the input for the student’s basic and academic information to calculate the result. The program should be designed to display the students’ information such as ‘name’, ’ph\_no’, ’roll\_no’, ’course enrolled’, ’marks’ and ‘average’. The program should have three classes namely ‘person’, ‘student’ and ‘exam’. ‘person’ is the base class which has the basic data members ‘name’ and ‘ph\_no’. The class ‘student’ has ‘roll\_no’ and ‘course’ with limited access for its own class. Another class named ‘exam’ has attributes namely ‘marks’ (five subject marks) and ‘average’ with limited access for its own class. Identify an appropriate inheritance technique to implement these classes. What other object orientation concept is needed to implement the scenario that reads and displays the values through the methods namely ‘read()’ and ‘show()’ in the three classes ‘person’, ‘student’ and ‘exam’? The driver program should control the entire set of inherited classes and also invoke the method of class ‘exam’ named ‘cal()’ which will calculate the total and the average marks obtained by the student.

#include <iostream>

#include <string>

class Person {

protected:

std::string name;

std::string ph\_no;

public:

void read() {

std::cout << "Enter name: ";

std::getline(std::cin >> std::ws, name);

std::cout << "Enter phone number: ";

std::getline(std::cin >> std::ws, ph\_no);

}

void show() {

std::cout << "Name: " << name << std::endl;

std::cout << "Phone Number: " << ph\_no << std::endl;

}

};

class Student : public Person {

private:

int roll\_no;

std::string course;

public:

void read() {

Person::read();

std::cout << "Enter roll number: ";

std::cin >> roll\_no;

std::cout << "Enter course enrolled: ";

std::cin >> course;

}

void show() {

Person::show();

std::cout << "Roll Number: " << roll\_no << std::endl;

std::cout << "Course Enrolled: " << course << std::endl;

}

};

class Exam : public Student {

private:

int marks[5];

float average;

public:

void read() {

Student::read();

std::cout << "Enter marks for 5 subjects:\n";

for (int i = 0; i < 5; i++) {

std::cout << "Subject " << i + 1 << ": ";

std::cin >> marks[i];

}

}

void cal() {

int total = 0;

for (int i = 0; i < 5; i++) {

total += marks[i];

}

average = static\_cast<float>(total) / 5;

}

void show() {

Student::show();

std::cout << "Marks:\n";

for (int i = 0; i < 5; i++) {

std::cout << "Subject " << i + 1 << ": " << marks[i] << std::endl;

}

std::cout << "Average: " << average << std::endl;

}

};

int main() {

Exam student;

student.read();

student.cal();

std::cout << "\nStudent Information:\n";

student.show();

return 0;

}

Develop C code to implement the following scenarios:

There is a computerized word game that consists of n words in a word bag (1...n). These words originate from the same stem. Example: grace, graceful, disgraceful, gracefully, etc. Design the computerized game with the C program that has a module to find the original stem of the word. The stem is the longest consecutive substring that occurs in all the n words. If there are ties, we will choose the smallest one in the alphabetical (lexicographic) order. When the above example is given as input, the stem of the word bag namely ‘grace’ has to be displayed as output.

#include <stdio.h>

#include <string.h>

// Function to find the original stem of a word bag

char\* findStem(char\*\* words, int n) {

int i, j;

int min\_len = strlen(words[0]); // Minimum length of words in the bag

// Find the minimum length among the words

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

int len = strlen(words[i]);

if (len < min\_len) {

min\_len = len;

}

}

// Compare characters at each position in the words

for (i = 0; i < min\_len; i++) {

char ch = words[0][i]; // Assume all words have the same character at this position

// Check if the character is the same in all words

for (j = 1; j < n; j++) {

if (words[j][i] != ch) {

break; // Characters differ, stop checking

}

}

if (j != n) {

break; // Characters differ in at least one word, stop checking

}

}

// Create a new string to store the stem

char\* stem = (char\*)malloc((i + 1) \* sizeof(char));

strncpy(stem, words[0], i);

stem[i] = '\0'; // Add null terminator to create a valid string

return stem;

}

int main() {

int n;

printf("Enter the number of words in the bag: ");

scanf("%d", &n);

// Create an array to store the words

char\*\* words = (char\*\*)malloc(n \* sizeof(char\*));

int i;

for (i = 0; i < n; i++) {

words[i] = (char\*)malloc(100 \* sizeof(char)); // Assuming each word has a maximum length of 99 characters

printf("Enter word %d: ", i + 1);

scanf("%s", words[i]);

}

// Find the stem of the word bag

char\* stem = findStem(words, n);

printf("Stem of the word bag: %s\n", stem);

// Free memory allocated for words and stem

for (i = 0; i < n; i++) {

free(words[i]);

}

free(words);

free(stem);

return 0;

}

Ravi Verma is a curious person. Once, he searched for various famous personalities on Google. When he searched about Neil Armstrong, to his wonder Armstrong numbers’ link was recommended by Google. Now he wanted to know all Armstrong numbers that are less than his birth year number. Get his year of birth as input and print all the Armstrong numbers up to that four-digit number (year of birth). Hint: An Armstrong number is a number that is equal to the sum of digits raised to the power total number of digits in the number.

#include <stdio.h>

#include <math.h>

// Function to calculate the number of digits in a number

int countDigits(int number) {

int count = 0;

while (number != 0) {

number /= 10;

count++;

}

return count;

}

// Function to check if a number is an Armstrong number

int isArmstrong(int number) {

int originalNumber = number;

int numDigits = countDigits(number);

int sum = 0;

while (number != 0) {

int digit = number % 10;

sum += pow(digit, numDigits);

number /= 10;

}

return (sum == originalNumber);

}

int main() {

int birthYear;

printf("Enter your year of birth: ");

scanf("%d", &birthYear);

printf("Armstrong numbers less than %d:\n", birthYear);

for (int i = 1; i < birthYear; i++) {

if (isArmstrong(i)) {

printf("%d\n", i);

}

}

return 0;

}

N balls are to be transported through a tunnel. Each ball has 3 parameters namely color, diameter (cm) and weight (g). One ball can be sent into the tunnel at a time. Store the three parameters for all the balls using an appropriate data structure in C and display whether the ball meets the required constraints or not in the main function. Use a function with a default parameter specified in the formula.

Constraints:

If the ball is red and diameter is <41 cm, the ball can pass through the tunnel.

If the ball is black and the weight is <3 kg, the ball can pass through the tunnel.

If the diameter of the ball is <100 cm and weight is <100 g, the ball can pass through the tunnel.

Pass the data obtained from the user to a function named ‘check()’. The result of the check() is to be passed to another function named ‘findVolume()’. Use pointers concept to avoid returning the volume from the function. Display the result in main().

Formula for finding volume of a sphere: (4/3)\*pi\*r^3

[Note: If the constraints are satisfied, the volume is to be calculated. If not, display “Cannot be transported through the tunnel”.]

#include <stdio.h>

#include <math.h>

// Structure to represent a ball

struct Ball {

char color[20];

float diameter;

float weight;

};

// Function to check if the ball meets the required constraints

int check(struct Ball \*ball) {

if (strcmp(ball->color, "red") == 0 && ball->diameter < 41.0) {

return 1;

} else if (strcmp(ball->color, "black") == 0 && ball->weight < 3000.0) {

return 1;

} else if (ball->diameter < 100.0 && ball->weight < 100.0) {

return 1;

} else {

return 0;

}

}

// Function to calculate the volume of a ball using the provided formula

void findVolume(struct Ball \*ball, float \*volume) {

float radius = ball->diameter / 2.0;

\*volume = (4.0 / 3.0) \* M\_PI \* pow(radius, 3);

}

int main() {

int n;

printf("Enter the number of balls: ");

scanf("%d", &n);

struct Ball balls[n];

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

printf("\nBall %d\n", i + 1);

printf("Enter color: ");

scanf("%s", balls[i].color);

printf("Enter diameter (cm): ");

scanf("%f", &balls[i].diameter);

printf("Enter weight (g): ");

scanf("%f", &balls[i].weight);

}

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

printf("\nBall %d\n", i + 1);

if (check(&balls[i])) {

float volume;

findVolume(&balls[i], &volume);

printf("Ball can be transported through the tunnel.\n");

printf("Volume of the ball: %.2f cubic cm\n", volume);

} else {

printf("Ball cannot be transported through the tunnel.\n");

}

}

return 0;

}

The general manager of the company wants to give a New Year bonus to his employees on 1/1/23. The joining details (name, employee ID and joining date) are maintained by the HR manager of the company. He decides to give the bonus based upon the date of joining (DOJ). If the DOJ is before 31/12/200, the employee will be given rs. 10000 as a bonus. If the DOJ is from 1/1/2001 to 31/12/2005, the employee will be given rs. 6000 as a bonus. If the DOJ is from 1/1/2016 till date, the employee will be given rs. 4000 as a bonus. According to numerology, the density number for 1/1/2023 is 9. In addition, if the employee’s density number for the joining date is 9 (adding up all the numbers present on the joining date), he decides to give rs. 1000 extra. The program should accept the number of employees in the database as input and a loop has to be defined to get the employee joining details. Develop an algorithm and write a C program using structures to display the following:

Name and eligible bonus amount of the employee queried by the employee ID

List of employee names and the bonus amount obtained by the employees who come under the category of the same density number

List of employee details who joined in the queried year

#include <stdio.h>

#include <stdlib.h>

struct Date {

int day;

int month;

int year;

};

struct Employee {

char name[50];

int employeeID;

struct Date joiningDate;

};

int calculateDensityNumber(struct Date date) {

int densityNumber = 0;

int day = date.day;

int month = date.month;

int year = date.year;

while (day > 0) {

densityNumber += day % 10;

day /= 10;

}

while (month > 0) {

densityNumber += month % 10;

month /= 10;

}

while (year > 0) {

densityNumber += year % 10;

year /= 10;

}

return densityNumber;

}

int calculateBonus(struct Date joiningDate, int densityNumber) {

int bonus = 0;

int year = joiningDate.year;

if (year < 2000) {

bonus = 10000;

} else if (year >= 2001 && year <= 2005) {

bonus = 6000;

} else if (year >= 2016) {

bonus = 4000;

}

if (densityNumber == 9) {

bonus += 1000;

}

return bonus;

}

int main() {

int numEmployees;

printf("Enter the number of employees: ");

scanf("%d", &numEmployees);

struct Employee employees[numEmployees];

// Input employee details

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

printf("\nEnter details for Employee %d:\n", i + 1);

printf("Name: ");

scanf("%s", employees[i].name);

printf("Employee ID: ");

scanf("%d", &employees[i].employeeID);

printf("Joining Date (dd mm yyyy): ");

scanf("%d %d %d", &employees[i].joiningDate.day, &employees[i].joiningDate.month, &employees[i].joiningDate.year);

}

int queryEmployeeID;

printf("\nEnter the Employee ID to query: ");

scanf("%d", &queryEmployeeID);

int queryYear;

printf("Enter the year to query: ");

scanf("%d", &queryYear);

int queryDensityNumber;

printf("Enter the density number to query: ");

scanf("%d", &queryDensityNumber);

printf("\n-- Query Result --\n");

// 1. Name and eligible bonus amount of the employee queried by the employee ID

printf("\nQuery Result 1:\n");

int foundEmployee = 0;

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

if (employees[i].employeeID == queryEmployeeID) {

int densityNumber = calculateDensityNumber(employees[i].joiningDate);

int bonus = calculateBonus(employees[i].joiningDate, densityNumber);

printf("Name: %s\n", employees[i].name);

printf("Eligible Bonus Amount: Rs. %d\n", bonus);

foundEmployee = 1;

break;

}

}

if (!foundEmployee) {

printf("Employee not found!\n");

}

// 2. List of employee names and bonus amounts obtained by employees with the same density number

printf("\nQuery Result 2:\n");

int foundSameDensityNumber = 0;

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

int densityNumber = calculateDensityNumber(employees[i].joiningDate);

if (densityNumber == queryDensityNumber) {

int bonus = calculateBonus(employees[i].joiningDate, densityNumber);

printf("Name: %s, Bonus Amount: Rs. %d\n", employees[i].name, bonus);

foundSameDensityNumber = 1;

}

}

if (!foundSameDensityNumber) {

printf("No employees found with the specified density number!\n");

}

// 3. List of employee details who joined in the queried year

printf("\nQuery Result 3:\n");

int foundJoiningYear = 0;

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

if (employees[i].joiningDate.year == queryYear) {

printf("Name: %s, Employee ID: %d\n", employees[i].name, employees[i].employeeID);

foundJoiningYear = 1;

}

}

if (!foundJoiningYear) {

printf("No employees found who joined in the specified year!\n");

}

return 0;

}

Imagine that you are developing a product review system for an online shopping platform. The system allows users to enter the name of a product and rate it on a scale of 1 to 10 (decimals not allowed). Each product review consists of the product name and the rating given by the user being stored together in an array. The array index is the product name. The company decides to give the product review based on the sales percentage of the product which is also stored in the same data structure as the above. Implement a generic function called ‘productReview()’ that returns the following:

Product name and the position of the product in the store when queried with the rating

Product name and position of the product when queried with the sales percentage

Write an object-oriented generic program that allows the user to pass the values in any datatype, through which the name and the position of the product in the store could be obtained as the output.

#include <iostream>

#include <map>

#include <vector>

template <typename T>

class ProductReviewSystem {

public:

void addProductReview(const T& productName, int rating, double salesPercentage) {

productReviews[productName] = std::make\_pair(rating, salesPercentage);

}

void productReview(const T& query, int& positionByRating, int& positionBySales) {

positionByRating = -1;

positionBySales = -1;

int rankByRating = 1;

double rankBySales = 1.0;

for (const auto& product : productReviews) {

if (product.second.first > productReviews[query].first) {

rankByRating++;

}

if (product.second.second > productReviews[query].second) {

rankBySales += 0.1;

}

}

positionByRating = rankByRating;

positionBySales = static\_cast<int>(rankBySales \* 10);

}

private:

std::map<T, std::pair<int, double>> productReviews;

};

int main() {

ProductReviewSystem<std::string> reviewSystem;

reviewSystem.addProductReview("Product A", 8, 0.65);

reviewSystem.addProductReview("Product B", 6, 0.85);

reviewSystem.addProductReview("Product C", 9, 0.45);

reviewSystem.addProductReview("Product D", 7, 0.55);

reviewSystem.addProductReview("Product E", 8, 0.75);

std::string queryProduct;

std::cout << "Enter the name of the product to query: ";

std::cin >> queryProduct;

int positionByRating, positionBySales;

reviewSystem.productReview(queryProduct, positionByRating, positionBySales);

std::cout << "Query Result:\n";

std::cout << "Product: " << queryProduct << "\n";

std::cout << "Position by Rating: " << positionByRating << "\n";

std::cout << "Position by Sales: " << positionBySales << "\n";

return 0;

}