Hristiyan Pavlov

Software Developer

0886946779

Hristian55555@gmail.com

□□. □□□□□□□2 , □□□□ 236, Sofia, 1220

SUMMERY

As a lifelong enthusiast of computers and technology, I pursued my passion by studying computer science in school. Despite initially struggling to keep up with the material, I persevered and developed a strong foundation in object-oriented programming, C, C#, and SQL. However, I knew that there was much more to learn and that the best way to continue my development was to work in a supportive and collaborative environment.

One year ago, I made the decision to focus more seriously on self-study and rediscovered my love for programming. I am now eager to join a team where I can continue to learn and grow as a developer. I am confident that my dedication, problem-solving skills, and passion for technology will make me a valuable asset to any team.

EXPERIENCES

Sales Consultant **9/2022 - 11/2022**

Technomarket - Sofia

EDUCATION

Secondary education - first degree - PGEA

9/2019 - 6/2024

LANGUAGES

- **English - FCE** B2

SKILLS

- **Collaboration & Teamwork** Advanced
- **Creative Thinking** Advanced
- **Analytical Thinking** Intermediate
- **C** Good
- **C#** Beginner
- **SQL** Beginner

HOBBIES

- Snowboarding
- Teakwondo - second degree black belt
- Solving Codewars problems
- Learning new skills

Programs in C

A program for checking if a credit card number is valid

```c
#include <cs50.h>
#include <stdio.h>

string check_card(int four, int TwoNumb, int sum_lastDigit, int count);
long long get_Num(void);
int get_count(long long Num);
int get_everyOther(long long Num);
int get_everyOther2(long long Num);
int get_firstTwo(long long Num);
int get_first(long long Num);
int main(void)

{
    //prompt for input
    long long Num = get_Num();
    int count = get_count(Num);
    int everyOther2 = get_everyOther2(Num);
    int everyOther = get_everyOther(Num);
    int first = get_first(Num);
    int firstTwo = get_firstTwo(Num);
    int sum_lastDigit = (everyOther2 + everyOther) % 10;


    string h = check_card(first, firstTwo, sum_lastDigit, count);
    printf("%s", h);

}

long long get_Num(void)
{
    long long get_Num;
    do
    {
        get_Num = get_long("Number: ");
    }
    while (get_Num < 1);
    return get_Num;
}



string check_card(int four, int TwoNumb, int sum_lastDigit, int count)
```

```
{
    string Name;
    if (count == 15 && (TwoNumb == 34 || TwoNumb == 37) && sum_lastDigit == 0)
    {
        return Name = "AMEX\n";
    }
    if (count == 16 && (TwoNumb == 51 || TwoNumb == 52 || TwoNumb == 53 ||
TwoNumb == 54 || TwoNumb == 55)  && sum_lastDigit == 0)
    {
        return Name = "MASTERCARD\n";
    }
    if (((count == 16) || (count == 13)) && four == 4  && sum_lastDigit == 0)
    {
        return Name = "VISA\n";
    }
    else
    {
        return Name = "INVALID\n";
    }

}

int get_count(long long Num)
{
    int count = 0;
    while (Num != 0)
    {
        Num /= 10;
        count++;
    }
    return count;
}

int get_everyOther(long long Num)
{
    int count2 = 0;
    long long b = Num;
    long everyOther;
    long sum = 0;
    while (b > 0)
    {

        long lastNumber = b / 10;
        everyOther = lastNumber % 10;
        everyOther *= 2;
```

```
        int everyOther2 = everyOther;

        for (count2 = 0; everyOther2 != 0; count2++)
        {
            everyOther2 /= 10;
        }
        if (count2 == 2)
        {
            int x = everyOther % 10;
            int m = everyOther / 10;
            everyOther = x + m;
            sum = sum + everyOther;
        }
        else
        {
            sum = sum + everyOther;
        }
        b /= 100;
    }
    return sum;
}

int get_everyOther2(long long Num)
{
    long everyOther3;
    long long z = Num;
    long sum2 = 0;
    while (z > 0)
    {
        everyOther3 = z % 10;
        long lastNumber = z / 10;
        z /= 100;
        sum2 = everyOther3 + sum2;
    }
    return sum2;
}

int get_firstTwo(long long Num)
{
    long long TwoNumb = Num;
    while (TwoNumb > 99)
    {
        TwoNumb /= 10;
    }
    return TwoNumb;
```

```
}

int get_first(long long Num)
{
    long long four = Num;
    while (four >= 10)
    {
        four /= 10;
    }
    return four;
}
```

A program for ranking candidates in an election:

```
#include <cs50.h>
#include <stdio.h>
#include <string.h>

// Max voters and candidates
#define MAX_VOTERS 100
#define MAX_CANDIDATES 9

// preferences[i][j] is jth preference for voter i
int preferences[MAX_VOTERS][MAX_CANDIDATES];

// Candidates have name, vote count, eliminated status
typedef struct
{
    string name;
    int votes;
    bool eliminated;
}
candidate;

// Array of candidates
candidate candidates[MAX_CANDIDATES];

// Numbers of voters and candidates
int voter_count;
int candidate_count;

// Function prototypes
bool vote(int voter, int rank, string name);
```

```c
void tabulate(void);
bool print_winner(void);
int find_min(void);
bool is_tie(int min);
void eliminate(int min);

int main(int argc, string argv[])
{
    // Check for invalid usage
    if (argc < 2)
    {
        printf("Usage: runoff [candidate ...]\n");
        return 1;
    }

    // Populate array of candidates
    candidate_count = argc - 1;
    if (candidate_count > MAX_CANDIDATES)
    {
        printf("Maximum number of candidates is %i\n", MAX_CANDIDATES);
        return 2;
    }
    for (int i = 0; i < candidate_count; i++)
    {
        candidates[i].name = argv[i + 1];
        candidates[i].votes = 0;
        candidates[i].eliminated = false;
    }

    voter_count = get_int("Number of voters: ");
    if (voter_count > MAX_VOTERS)
    {
        printf("Maximum number of voters is %i\n", MAX_VOTERS);
        return 3;
    }

    // Keep querying for votes
    for (int i = 0; i < voter_count; i++)
    {

        // Query for each rank
        for (int j = 0; j < candidate_count; j++)
        {
            string name = get_string("Rank %i: ", j + 1);
            // Record vote, unless it's invalid
```

```
            if (!vote(i, j, name))
            {
                printf("Invalid vote.\n");
                return 4;
            }
        }

        printf("\n");
    }

    // Keep holding runoffs until winner exists
    while (true)
    {
        // Calculate votes given remaining candidates
        tabulate();

        // Check if election has been won
        bool won = print_winner();
        if (won)
        {
            break;
        }

        // Eliminate last-place candidates
        int min = find_min();
        bool tie = is_tie(min);

        // If tie, everyone wins
        if (tie)
        {
            for (int i = 0; i < candidate_count; i++)
            {
                if (!candidates[i].eliminated)
                {
                    printf("%s\n", candidates[i].name);
                }
            }
            break;
        }

        // Eliminate anyone with minimum number of votes
        eliminate(min);

        // Reset vote counts back to zero
        for (int i = 0; i < candidate_count; i++)
```

```
        {
            candidates[i].votes = 0;
        }
    }
    return 0;
}

// Record preference if vote is valid
bool vote(int voter, int rank, string name)
{
    for (int i = 0; i < candidate_count; i++)
    {
        if (strcmp(name, candidates[i].name) == 0)
        {
            preferences[voter][rank] = i;
            return true;
        }
    }
    // TODO
    return false;
}

// Tabulate votes for non-eliminated candidates
void tabulate(void)
{
    for (int i = 0; i < voter_count; i++)
    {
        for (int j = 0; j < candidate_count; j++)
        {
            if (candidates[preferences[i][j]].eliminated == false)
            {
                candidates[preferences[i][j]].votes++;
                break;
            }
        }
    }
    return;
}

// Print the winner of the election, if there is one
bool print_winner(void)
{
    int v = voter_count / 2;
    for (int i = 0; i < candidate_count; i++)
    {
```

```
        if (candidates[i].votes > v)
        {
            printf("%s\n", candidates[i].name);
            return true;
        }
    }
    // TODO
    return false;
}

// Return the minimum number of votes any remaining candidate has
int find_min(void)
{
    int loser = voter_count;
    for (int i = 0; i < candidate_count; i++)
    {
        if (candidates[i].eliminated == false)
        {
            if (candidates[i].votes < loser)
            {
                loser = candidates[i].votes;
            }
        }
    }
    return loser;
}

// Return true if the election is tied between all candidates, false otherwise
bool is_tie(int min)
{
    for (int i = 0; i < candidate_count; i++)
    {
        if (candidates[i].eliminated == false && candidates[i].votes > min)
        {
            return false;
        }
    }
    // TODO
    return true;
}

// Eliminate the candidate (or candidates) in last place
void eliminate(int min)
{
    for (int i = 0; i < candidate_count; i++)
```

```
    {
        if (candidates[i].eliminated == false && candidates[i].votes == min)
        {
            candidates[i].eliminated = true;
        }
    }
    // TODO
    return;
}
```

Sql code for a database of a bus station

```
-- Create a table for bus routes
CREATE TABLE routes (
  route_id INTEGER PRIMARY KEY,
  route_name TEXT NOT NULL,
  start_location TEXT NOT NULL,
  end_location TEXT NOT NULL
);

-- Create a table for bus stops
CREATE TABLE stops (
  stop_id INTEGER PRIMARY KEY,
  stop_name TEXT NOT NULL,
  location TEXT NOT NULL
);

-- Create a table for bus trips
CREATE TABLE trips (
  trip_id INTEGER PRIMARY KEY,
  route_id INTEGER NOT NULL,
  trip_date DATE NOT NULL,
```

```
  trip_time TIME NOT NULL,
  FOREIGN KEY (route_id) REFERENCES routes(route_id)
);

-- Create a table for bus tickets
CREATE TABLE tickets (
  ticket_id INTEGER PRIMARY KEY,
  trip_id INTEGER NOT NULL,
  passenger_name TEXT NOT NULL,
  seat_number INTEGER NOT NULL,
  FOREIGN KEY (trip_id) REFERENCES trips(trip_id)
);

-- Create a table for bus stop times
CREATE TABLE stop_times (
  stop_time_id INTEGER PRIMARY KEY,
  trip_id INTEGER NOT NULL,
  stop_id INTEGER NOT NULL,
  arrival_time TIME NOT NULL,
  departure_time TIME NOT NULL,
  FOREIGN KEY (trip_id) REFERENCES trips(trip_id),
  FOREIGN KEY (stop_id) REFERENCES stops(stop_id)
);
```