Programs in C

A program for checking if a credit card number is valid

#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);

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)

);