計算機程式語言

教課教授 : 謝東儒

助教 :蔡詠聿、吳品頤

Chapter 10_project 1

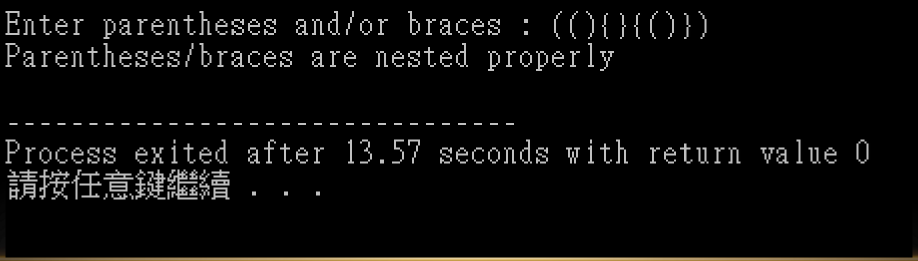
Mocfity the stack example of Section 10.2 so that it stores characters instead of integers. Next, add a main function that asks the user to enter a series of parentheses and/or braces, then indicates whether or not they're properly nested:

Enter parentheses and/or braces：(()()(()))

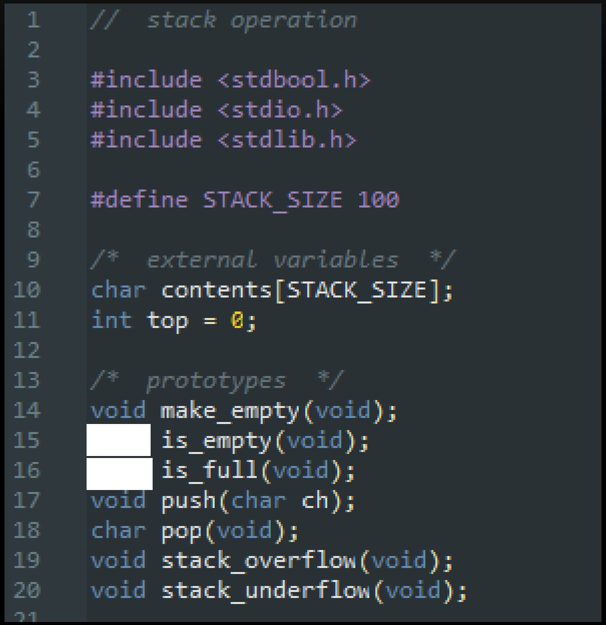
Parentheses/braces are nested properly

Hint: As the program reads characters, have it push each left parenthesis or left brace. When it reads a right parenthesis or brace, have it pop the stack and check that the item popped is a matching parenthesis or brace. (If not, the parentheses/braces aren't nested properly.) When the program reads the new-line character, have it check whether the stack is empty; if so, the parentheses/braces are matched. If the stack isn't empty (or if *stack_underflow* is ever called),

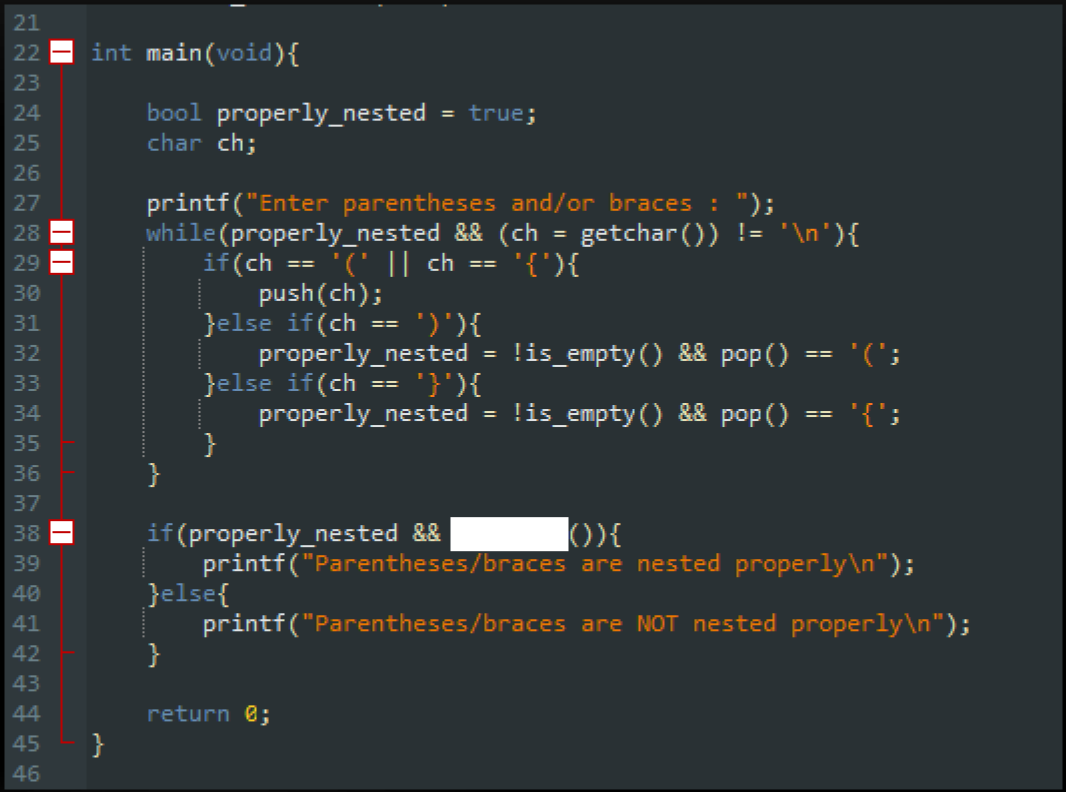
the parentheses/braces aren't matched. If *stack_overflow* is called, have the program print the message Stack overflow and terminate immediately.


```
Enter parentheses and/or braces : (()(}{()})
Parentheses/braces are nested properly

--------------------------------
Process exited after 13.57 seconds with return value 0
請按任意鍵繼續 . . .
```

Solution

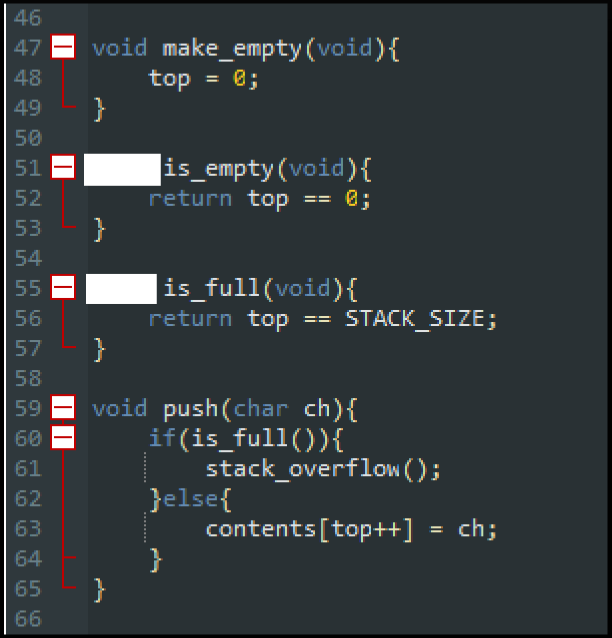


```c
//  stack operation

#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#define STACK_SIZE 100

/*  external variables  */
char contents[STACK_SIZE];
int top = 0;

/*  prototypes  */
void make_empty(void);
     is_empty(void);
     is_full(void);
void push(char ch);
char pop(void);
void stack_overflow(void);
void stack_underflow(void);
```

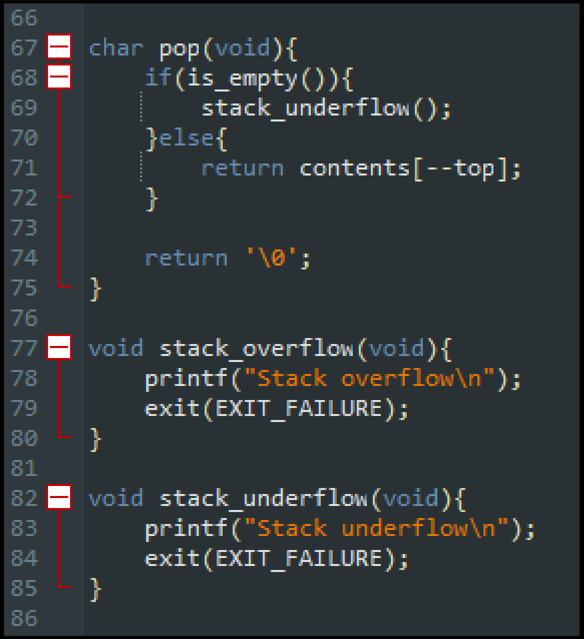


```c

int main(void){

    bool properly_nested = true;
    char ch;

    printf("Enter parentheses and/or braces : ");
    while(properly_nested && (ch = getchar()) != '\n'){
        if(ch == '(' || ch == '{'){
            push(ch);
        }else if(ch == ')'){
            properly_nested = !is_empty() && pop() == '(';
        }else if(ch == '}'){
            properly_nested = !is_empty() && pop() == '{';
        }
    }

    if(properly_nested &&         ()){
        printf("Parentheses/braces are nested properly\n");
    }else{
        printf("Parentheses/braces are NOT nested properly\n");
    }

    return 0;
}

```

Solution

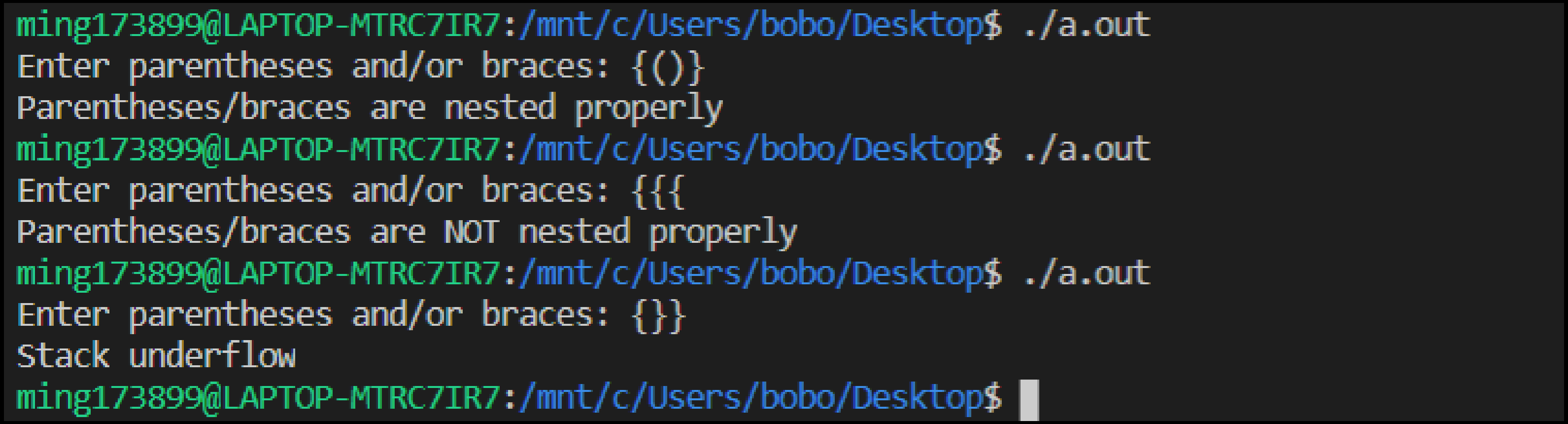


```c

void make_empty(void){
    top = 0;
}

     is_empty(void){
    return top == 0;
}

     is_full(void){
    return top == STACK_SIZE;
}

void push(char ch){
    if(is_full()){
        stack_overflow();
    }else{
        contents[top++] = ch;
    }
}

```



```c

char pop(void){
    if(is_empty()){
        stack_underflow();
    }else{
        return contents[--top];
    }

    return '\0';
}

void stack_overflow(void){
    printf("Stack overflow\n");
    exit(EXIT_FAILURE);
}

void stack_underflow(void){
    printf("Stack underflow\n");
    exit(EXIT_FAILURE);
}

```

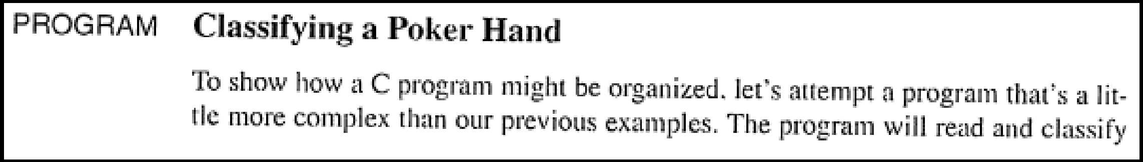
Example



```
ming173899@LAPTOP-MTRC7IR7:/mnt/c/Users/bobo/Desktop$ ./a.out
Enter parentheses and/or braces: {()}
Parentheses/braces are nested properly
ming173899@LAPTOP-MTRC7IR7:/mnt/c/Users/bobo/Desktop$ ./a.out
Enter parentheses and/or braces: {{{
Parentheses/braces are NOT nested properly
ming173899@LAPTOP-MTRC7IR7:/mnt/c/Users/bobo/Desktop$ ./a.out
Enter parentheses and/or braces: {}}
Stack underflow
ming173899@LAPTOP-MTRC7IR7:/mnt/c/Users/bobo/Desktop$
```

Chapter 10_project 2

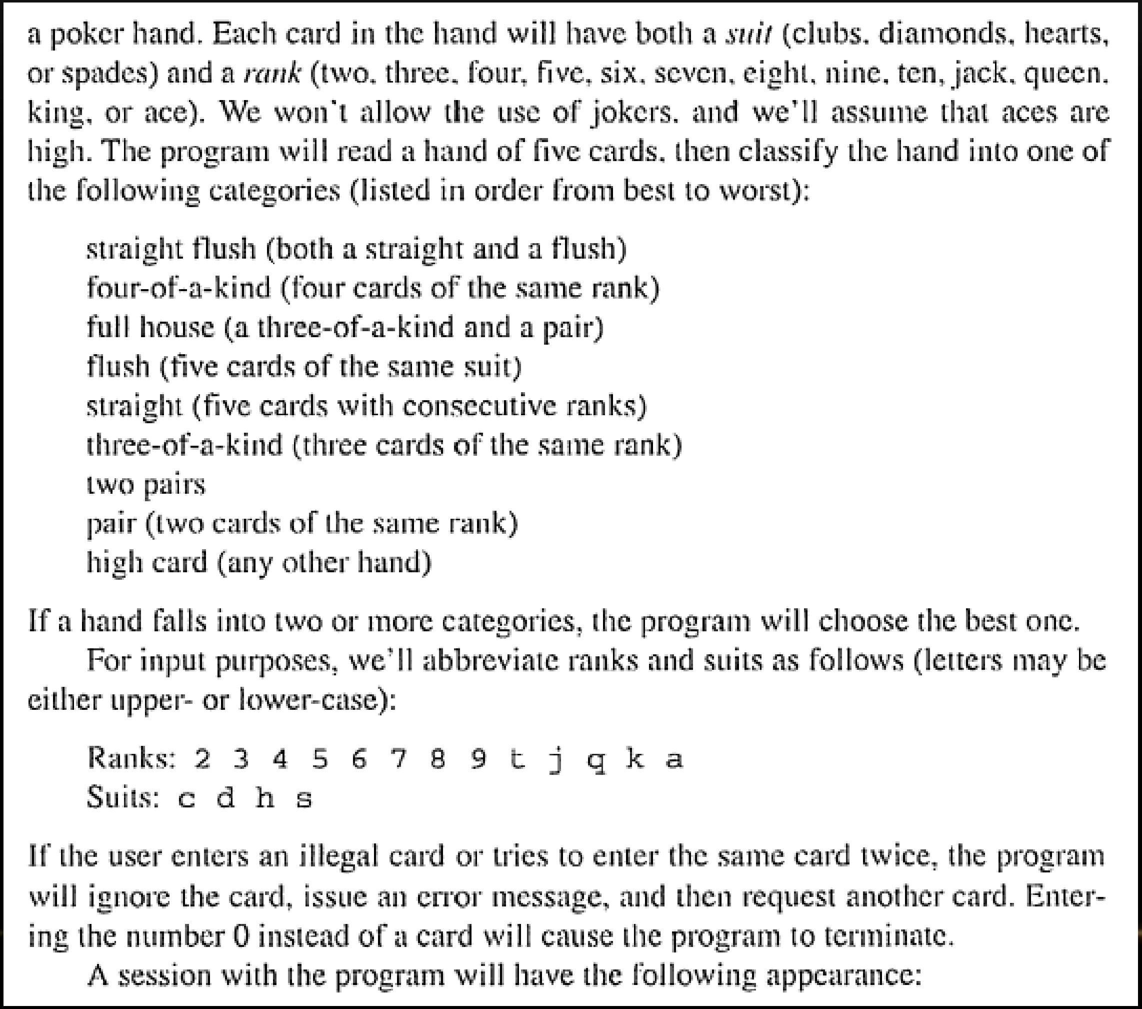
Modify the poker, c program of Section 10.5 by moving the num_in_rank and num_in_suit arrays into main, which will pass them as arguments to read_cards and analyze_hand.



PROGRAM **Classifying a Poker Hand**

To show how a C program might be organized, let's attempt a program that's a little more complex than our previous examples. The program will read and classify

Chapter 10_project 2



a poker hand. Each card in the hand will have both a *suit* (clubs, diamonds, hearts, or spades) and a *rank* (two, three, four, five, six, seven, eight, nine, ten, jack, queen, king, or ace). We won't allow the use of jokers, and we'll assume that aces are high. The program will read a hand of five cards, then classify the hand into one of the following categories (listed in order from best to worst):

straight flush (both a straight and a flush)
four-of-a-kind (four cards of the same rank)
full house (a three-of-a-kind and a pair)
flush (five cards of the same suit)
straight (five cards with consecutive ranks)
three-of-a-kind (three cards of the same rank)
two pairs
pair (two cards of the same rank)
high card (any other hand)

If a hand falls into two or more categories, the program will choose the best one.

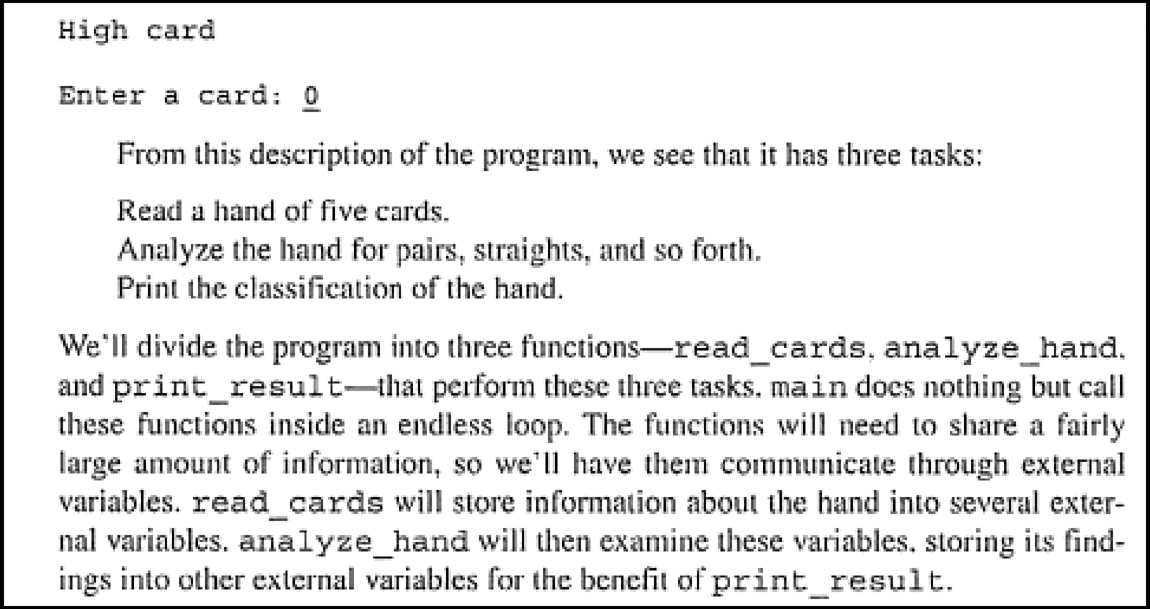
For input purposes, we'll abbreviate ranks and suits as follows (letters may be either upper- or lower-case):

Ranks: `2 3 4 5 6 7 8 9 t j q k a`
Suits: `c d h s`

If the user enters an illegal card or tries to enter the same card twice, the program will ignore the card, issue an error message, and then request another card. Entering the number 0 instead of a card will cause the program to terminate.

A session with the program will have the following appearance:

Chapter 10_project 2



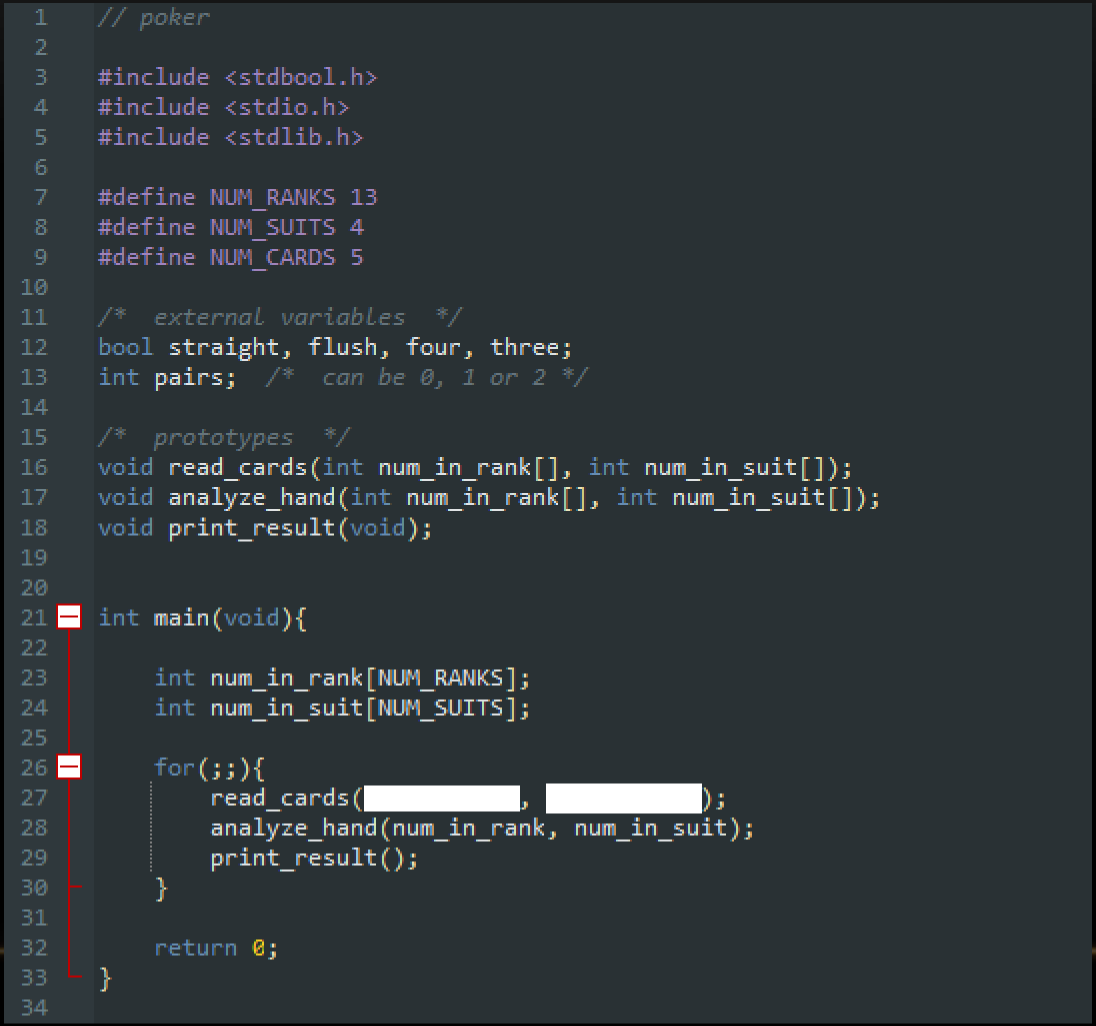
```
High card

Enter a card: 0
```

From this description of the program, we see that it has three tasks:

Read a hand of five cards.
Analyze the hand for pairs, straights, and so forth.
Print the classification of the hand.

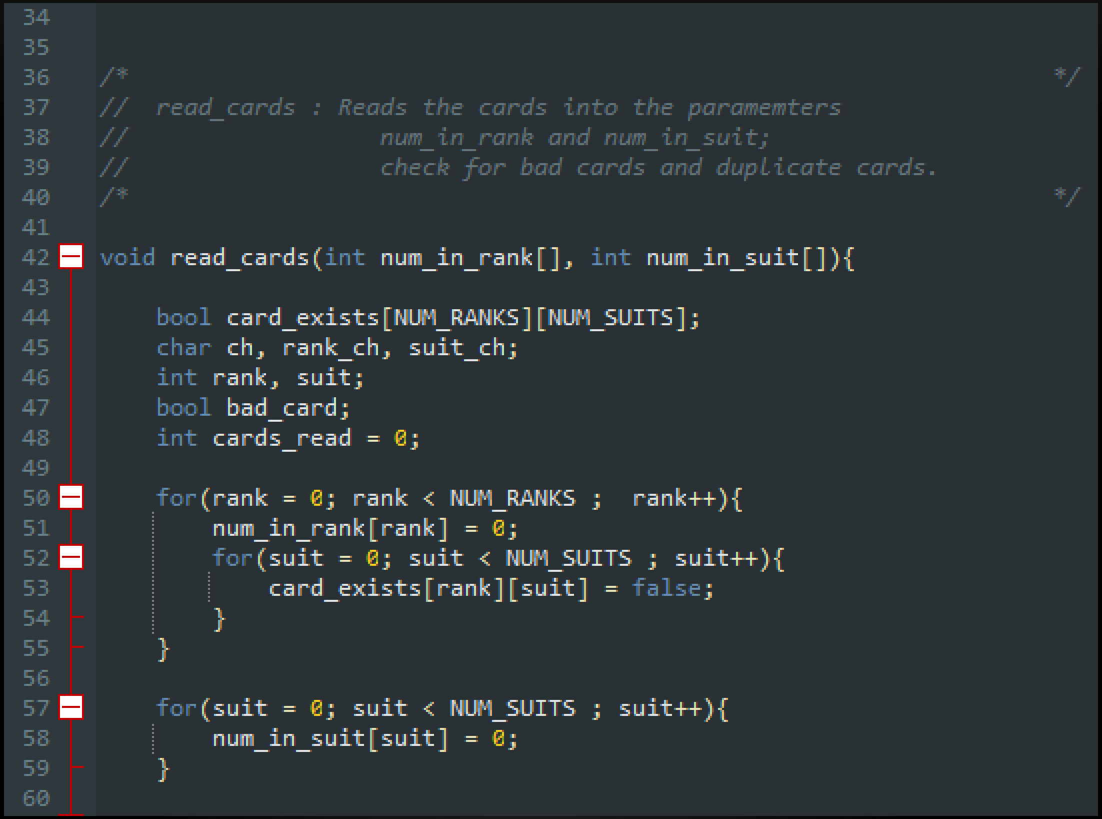
We'll divide the program into three functions—`read_cards`, `analyze_hand`, and `print_result`—that perform these three tasks. `main` does nothing but call these functions inside an endless loop. The functions will need to share a fairly large amount of information, so we'll have them communicate through external variables. `read_cards` will store information about the hand into several external variables. `analyze_hand` will then examine these variables, storing its findings into other external variables for the benefit of `print_result`.

Solution



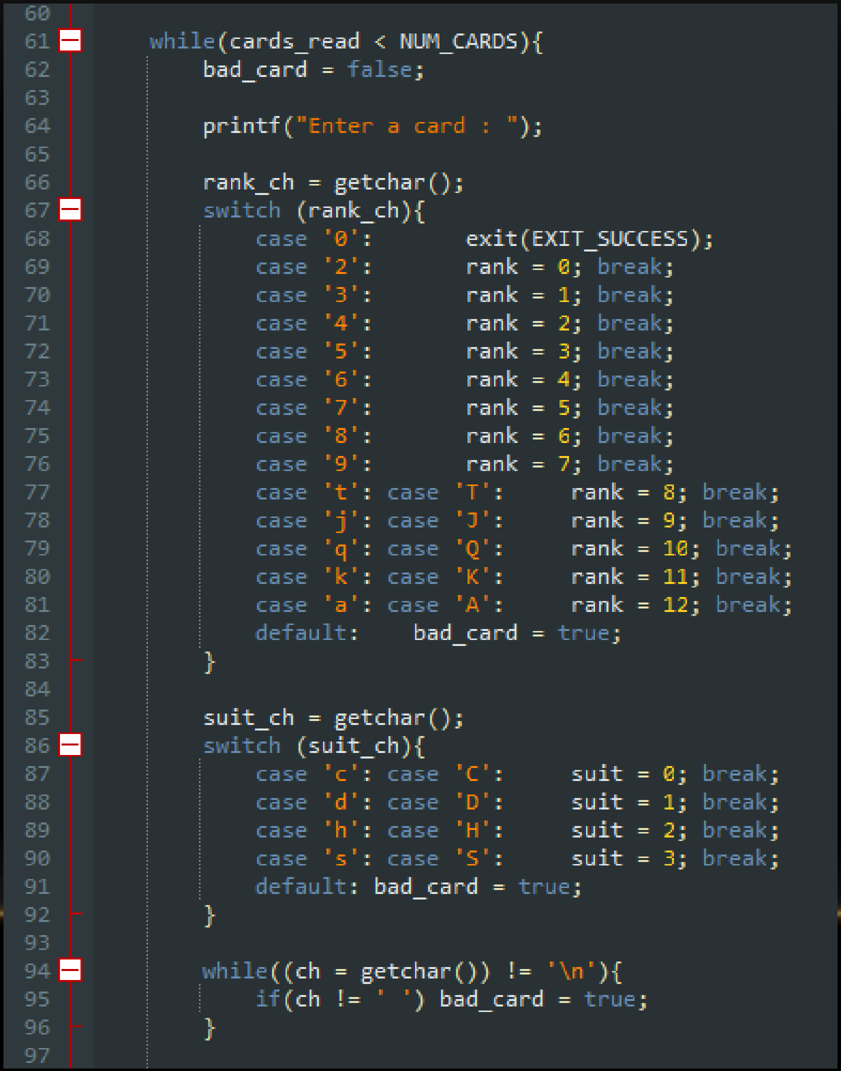
```c
// poker

#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#define NUM_RANKS 13
#define NUM_SUITS 4
#define NUM_CARDS 5

/*  external variables  */
bool straight, flush, four, three;
int pairs;  /*  can be 0, 1 or 2 */

/*  prototypes  */
void read_cards(int num_in_rank[], int num_in_suit[]);
void analyze_hand(int num_in_rank[], int num_in_suit[]);
void print_result(void);


int main(void){

    int num_in_rank[NUM_RANKS];
    int num_in_suit[NUM_SUITS];

    for(;;){
        read_cards(           ,            );
        analyze_hand(num_in_rank, num_in_suit);
        print_result();
    }

    return 0;
}

```

Solution



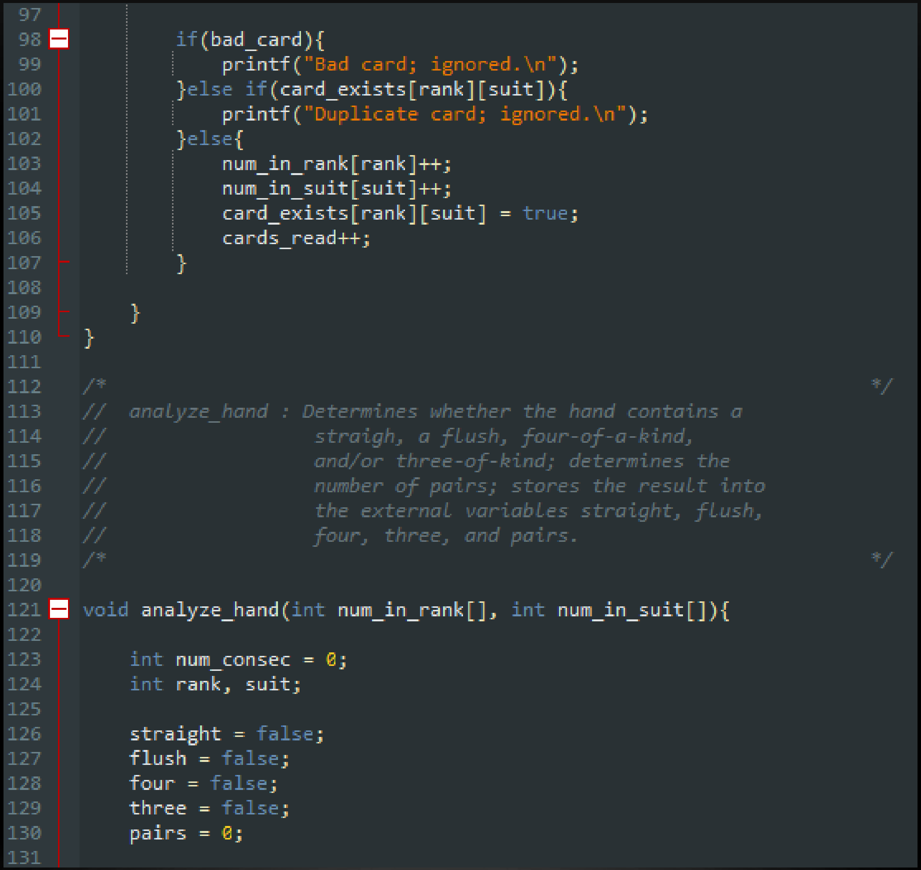
```


/*                                                                          */
//  read_cards : Reads the cards into the paramemters
//                      num_in_rank and num_in_suit;
//                      check for bad cards and duplicate cards.
/*                                                                          */

void read_cards(int num_in_rank[], int num_in_suit[]){

    bool card_exists[NUM_RANKS][NUM_SUITS];
    char ch, rank_ch, suit_ch;
    int rank, suit;
    bool bad_card;
    int cards_read = 0;

    for(rank = 0; rank < NUM_RANKS ;  rank++){
        num_in_rank[rank] = 0;
        for(suit = 0; suit < NUM_SUITS ; suit++){
            card_exists[rank][suit] = false;
        }
    }

    for(suit = 0; suit < NUM_SUITS ; suit++){
        num_in_suit[suit] = 0;
    }

```

Solution



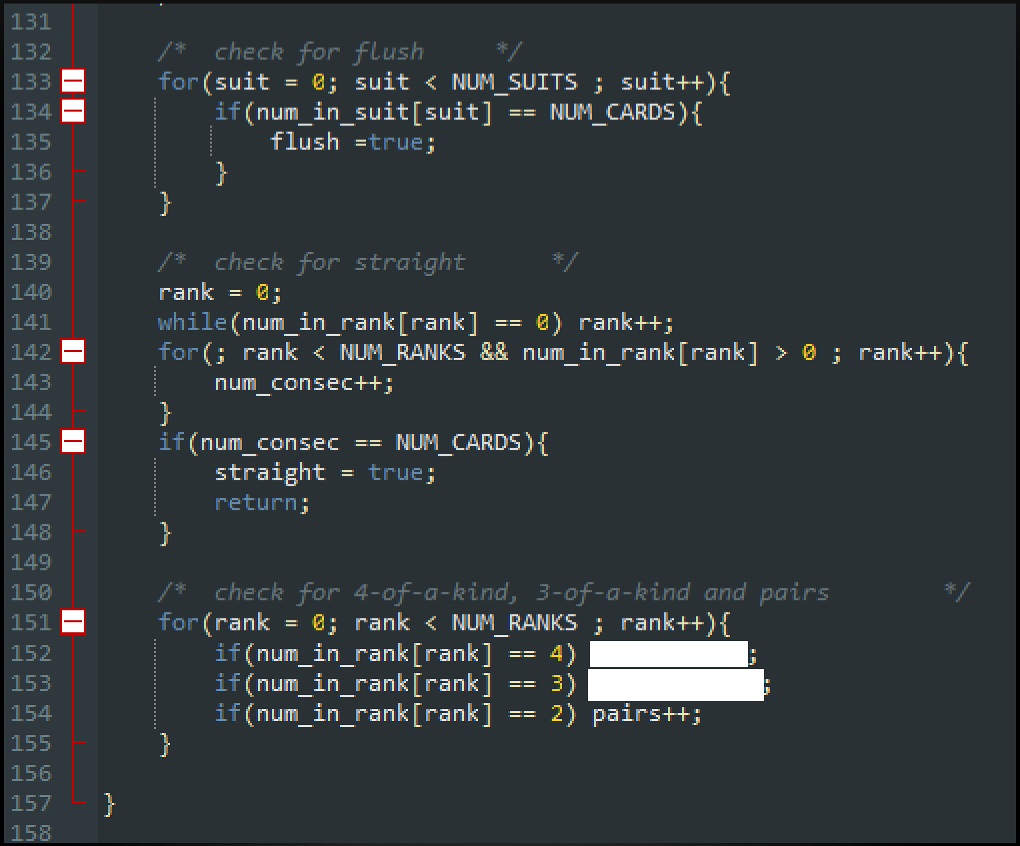
```

    while(cards_read < NUM_CARDS){
        bad_card = false;

        printf("Enter a card : ");

        rank_ch = getchar();
        switch (rank_ch){
            case '0':        exit(EXIT_SUCCESS);
            case '2':        rank = 0; break;
            case '3':        rank = 1; break;
            case '4':        rank = 2; break;
            case '5':        rank = 3; break;
            case '6':        rank = 4; break;
            case '7':        rank = 5; break;
            case '8':        rank = 6; break;
            case '9':        rank = 7; break;
            case 't': case 'T':     rank = 8; break;
            case 'j': case 'J':     rank = 9; break;
            case 'q': case 'Q':     rank = 10; break;
            case 'k': case 'K':     rank = 11; break;
            case 'a': case 'A':     rank = 12; break;
            default:     bad_card = true;
        }

        suit_ch = getchar();
        switch (suit_ch){
            case 'c': case 'C':     suit = 0; break;
            case 'd': case 'D':     suit = 1; break;
            case 'h': case 'H':     suit = 2; break;
            case 's': case 'S':     suit = 3; break;
            default: bad_card = true;
        }

        while((ch = getchar()) != '\n'){
            if(ch != ' ') bad_card = true;
        }

```

Solution



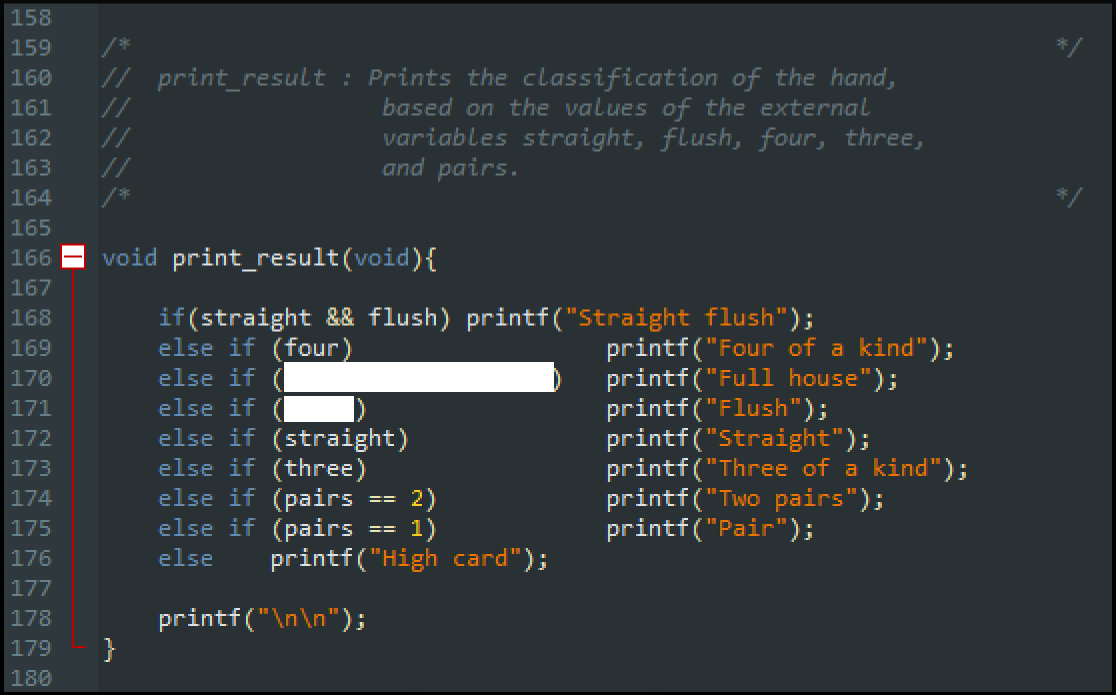
```

        if(bad_card){
            printf("Bad card; ignored.\n");
        }else if(card_exists[rank][suit]){
            printf("Duplicate card; ignored.\n");
        }else{
            num_in_rank[rank]++;
            num_in_suit[suit]++;
            card_exists[rank][suit] = true;
            cards_read++;
        }

    }
}

/*                                                                  */
//  analyze_hand : Determines whether the hand contains a
//                 straigh, a flush, four-of-a-kind,
//                 and/or three-of-kind; determines the
//                 number of pairs; stores the result into
//                 the external variables straight, flush,
//                 four, three, and pairs.
/*                                                                  */

void analyze_hand(int num_in_rank[], int num_in_suit[]){

    int num_consec = 0;
    int rank, suit;

    straight = false;
    flush = false;
    four = false;
    three = false;
    pairs = 0;

```

Solution



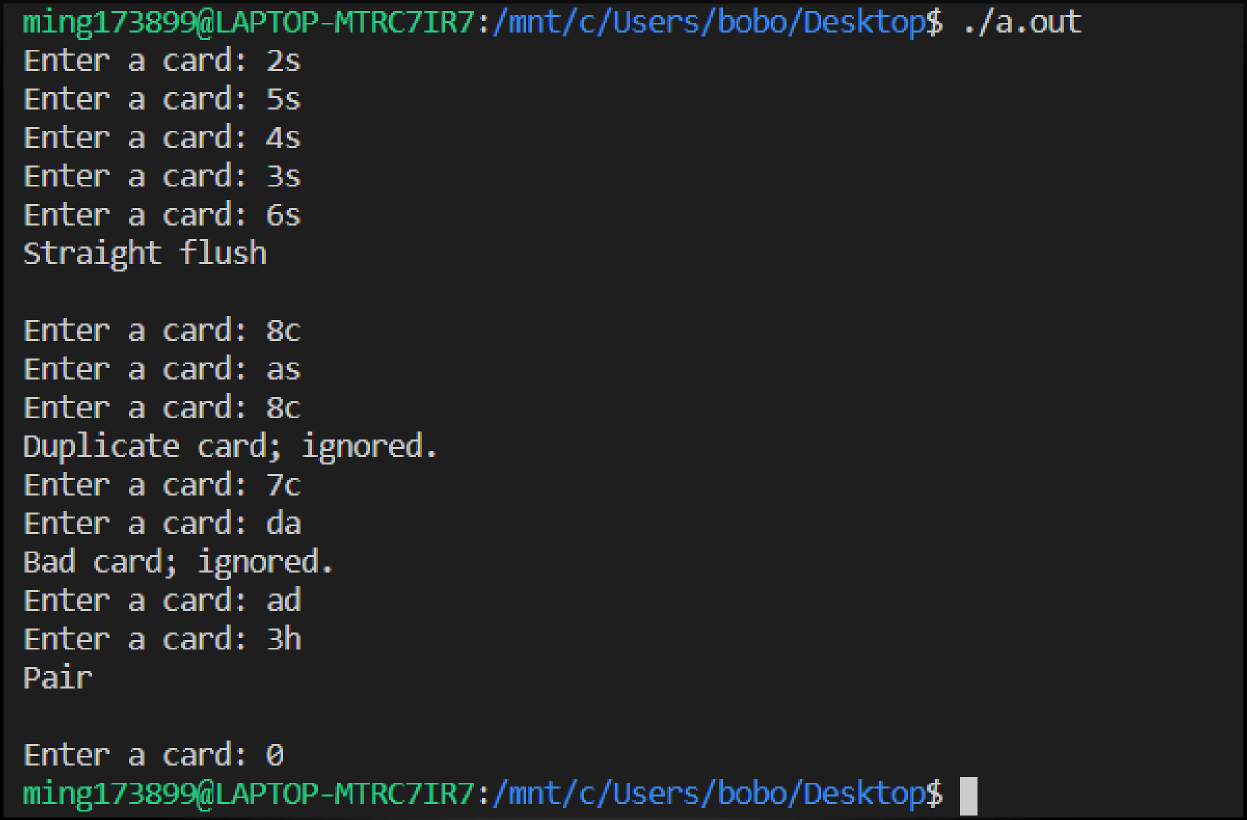
```
    /*  check for flush     */
    for(suit = 0; suit < NUM_SUITS ; suit++){
        if(num_in_suit[suit] == NUM_CARDS){
            flush =true;
        }
    }

    /*  check for straight      */
    rank = 0;
    while(num_in_rank[rank] == 0) rank++;
    for(; rank < NUM_RANKS && num_in_rank[rank] > 0 ; rank++){
        num_consec++;
    }
    if(num_consec == NUM_CARDS){
        straight = true;
        return;
    }

    /*  check for 4-of-a-kind, 3-of-a-kind and pairs        */
    for(rank = 0; rank < NUM_RANKS ; rank++){
        if(num_in_rank[rank] == 4)            ;
        if(num_in_rank[rank] == 3)             ;
        if(num_in_rank[rank] == 2) pairs++;
    }

}
```

Solution



```
/*                                                                      */
//  print_result : Prints the classification of the hand,
//                 based on the values of the external
//                 variables straight, flush, four, three,
//                 and pairs.
/*                                                                      */

void print_result(void){

    if(straight && flush) printf("Straight flush");
    else if (four)                  printf("Four of a kind");
    else if (                   )   printf("Full house");
    else if (     )                 printf("Flush");
    else if (straight)              printf("Straight");
    else if (three)                 printf("Three of a kind");
    else if (pairs == 2)            printf("Two pairs");
    else if (pairs == 1)            printf("Pair");
    else    printf("High card");

    printf("\n\n");
}
```

Example

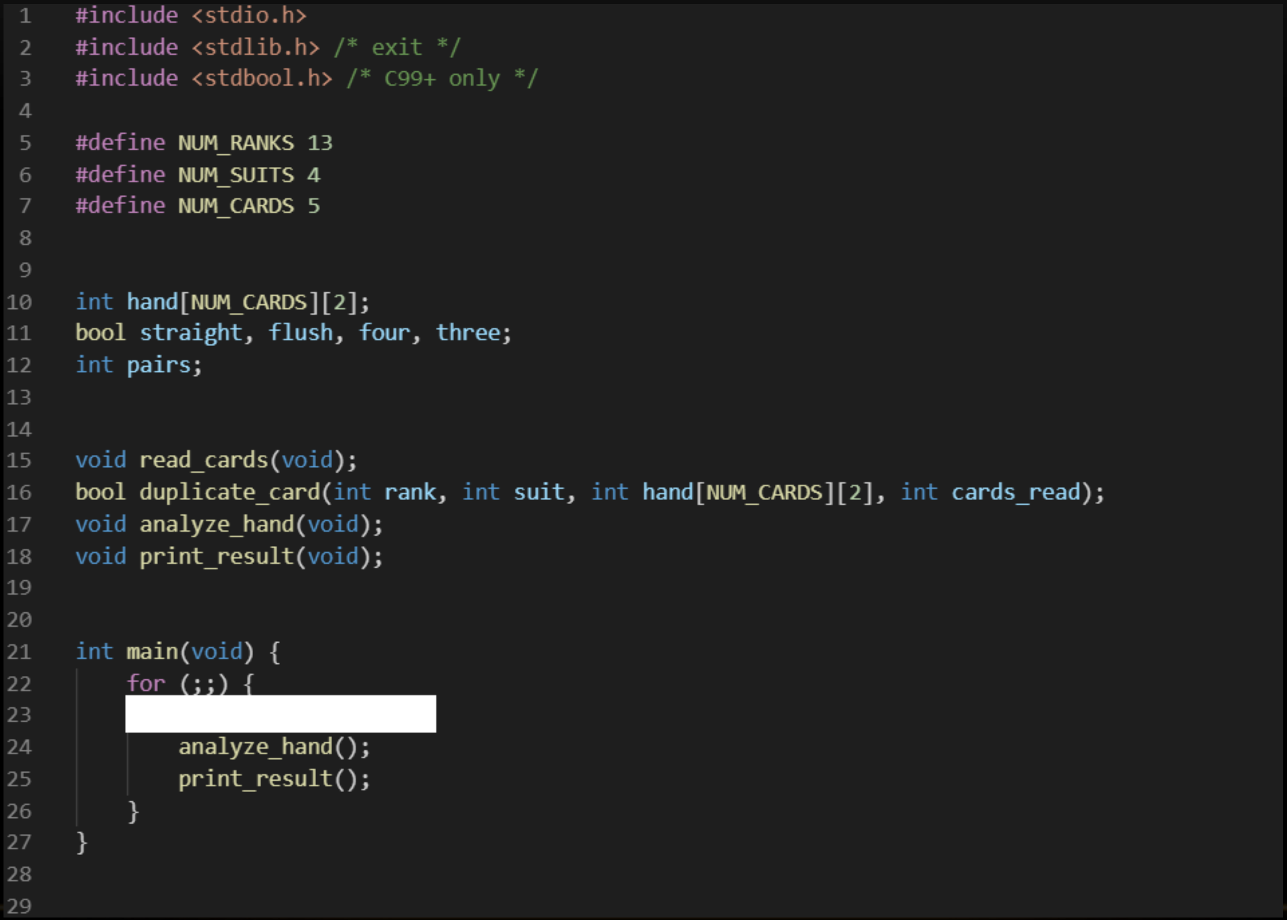


```
ming173899@LAPTOP-MTRC7IR7:/mnt/c/Users/bobo/Desktop$ ./a.out
Enter a card: 2s
Enter a card: 5s
Enter a card: 4s
Enter a card: 3s
Enter a card: 6s
Straight flush

Enter a card: 8c
Enter a card: as
Enter a card: 8c
Duplicate card; ignored.
Enter a card: 7c
Enter a card: da
Bad card; ignored.
Enter a card: ad
Enter a card: 3h
Pair

Enter a card: 0
ming173899@LAPTOP-MTRC7IR7:/mnt/c/Users/bobo/Desktop$
```

Chapter 10_project 3

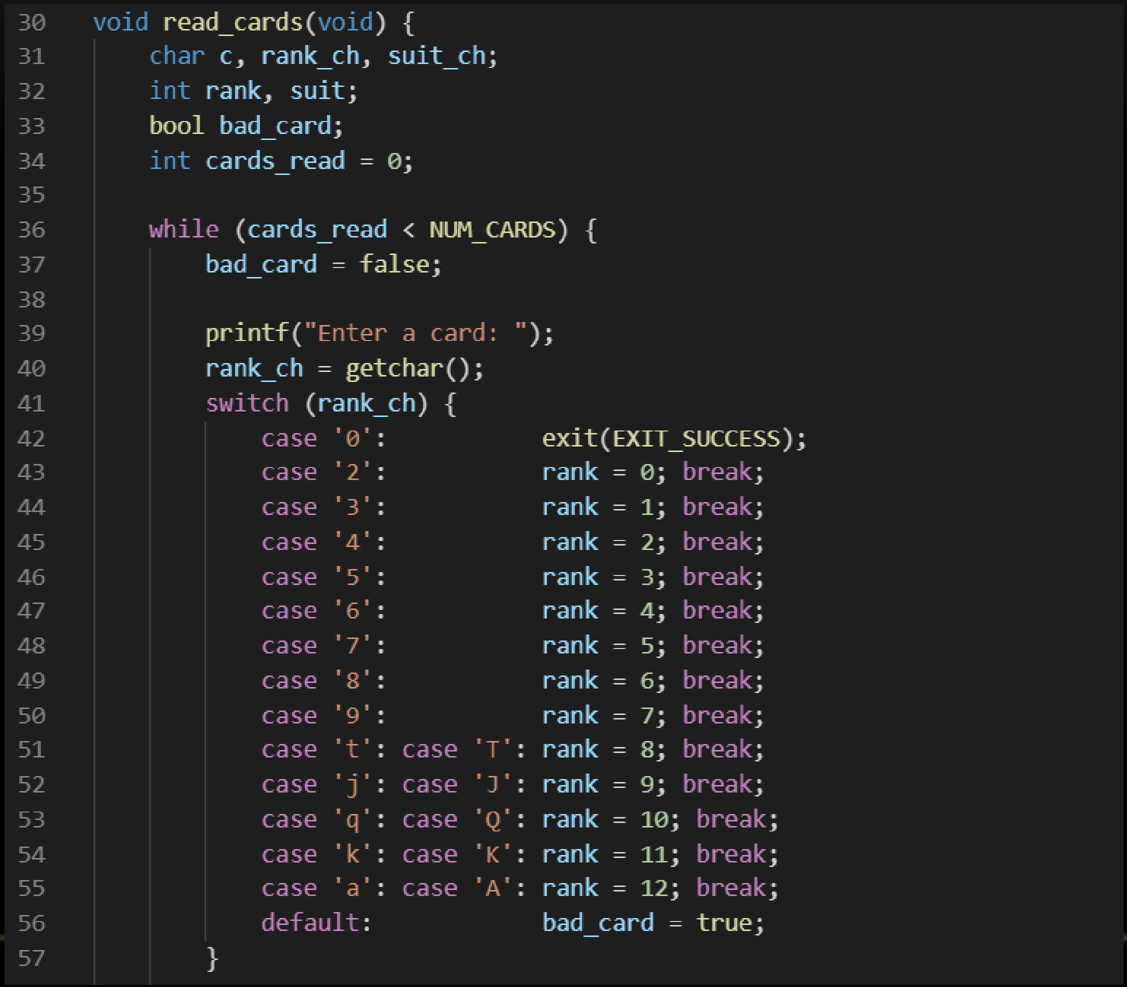
Remove the num_in_rank, num_in_suit, and card_exists arrays from the poker.c program of Section 10.5. Have the program store the cards in a 5 X 2 array instead. Each row of the array will represent a card. For example, if the array is named hand, then hand[0][0] will store the rank of the first card and hand[0][1] will store the suit of the first card.

Solution



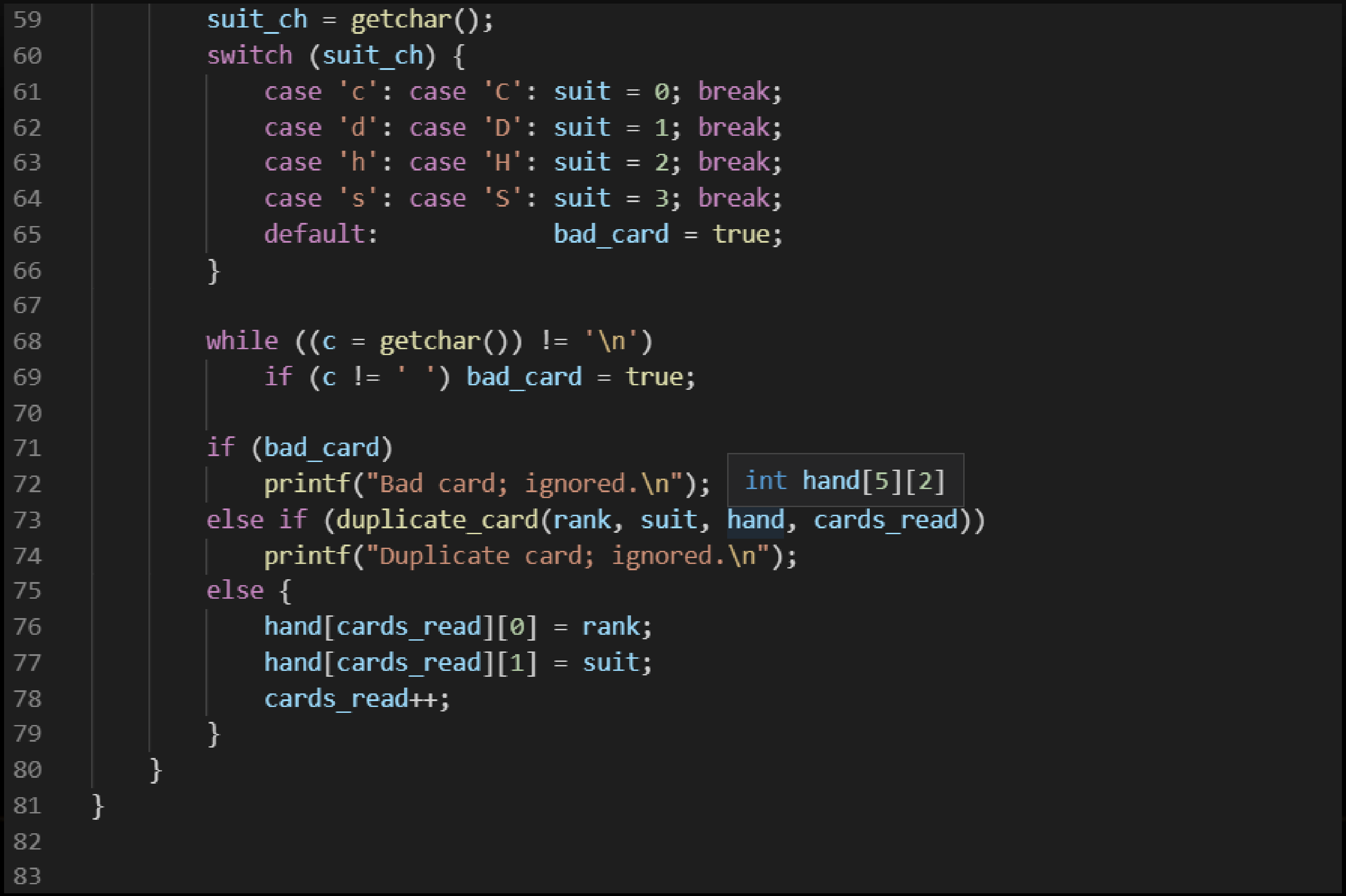
```c
#include <stdio.h>
#include <stdlib.h> /* exit */
#include <stdbool.h> /* C99+ only */

#define NUM_RANKS 13
#define NUM_SUITS 4
#define NUM_CARDS 5


int hand[NUM_CARDS][2];
bool straight, flush, four, three;
int pairs;


void read_cards(void);
bool duplicate_card(int rank, int suit, int hand[NUM_CARDS][2], int cards_read);
void analyze_hand(void);
void print_result(void);


int main(void) {
    for (;;) {
        [redacted]
        analyze_hand();
        print_result();
    }
}


```

Solution



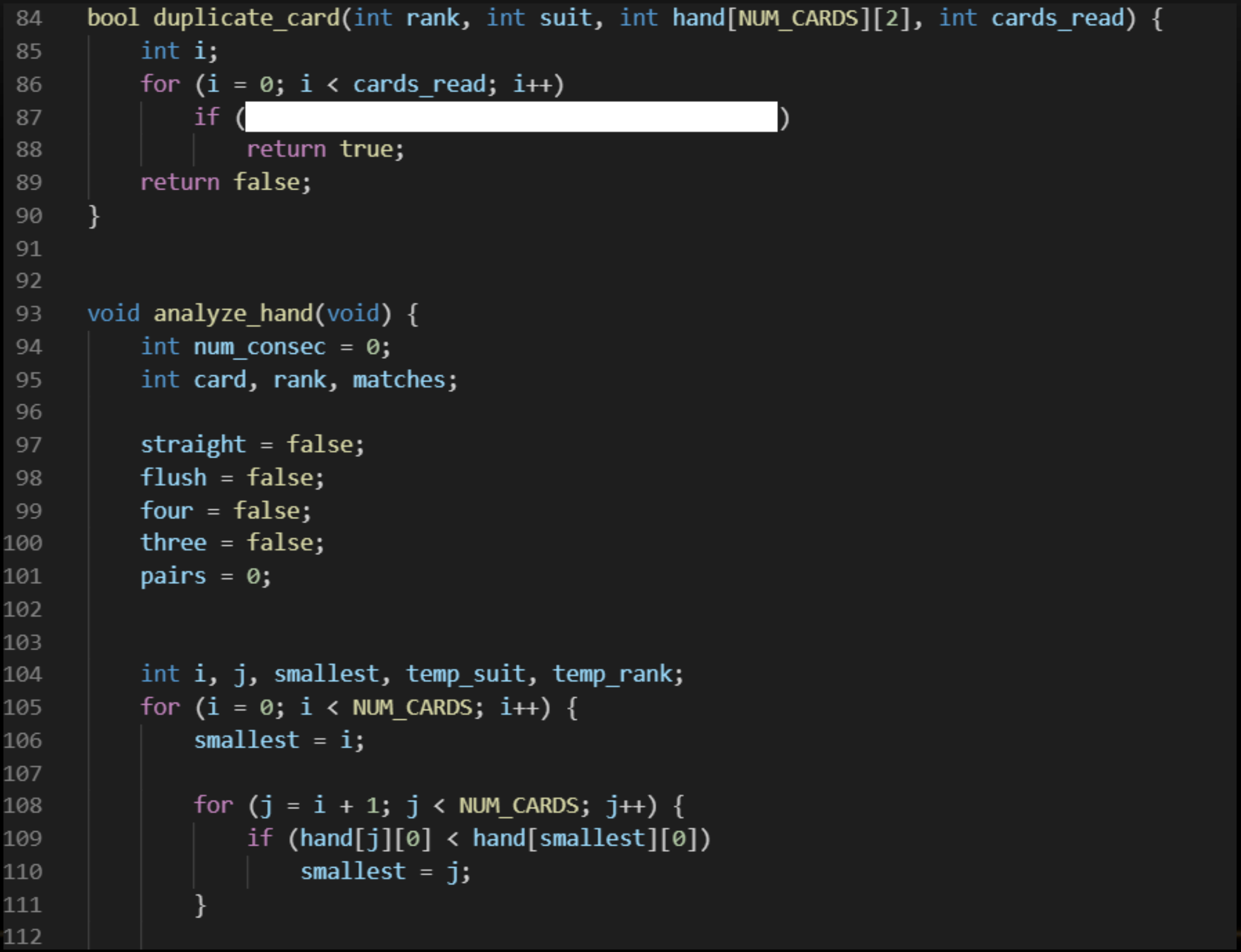
```c
void read_cards(void) {
    char c, rank_ch, suit_ch;
    int rank, suit;
    bool bad_card;
    int cards_read = 0;

    while (cards_read < NUM_CARDS) {
        bad_card = false;

        printf("Enter a card: ");
        rank_ch = getchar();
        switch (rank_ch) {
            case '0':           exit(EXIT_SUCCESS);
            case '2':           rank = 0; break;
            case '3':           rank = 1; break;
            case '4':           rank = 2; break;
            case '5':           rank = 3; break;
            case '6':           rank = 4; break;
            case '7':           rank = 5; break;
            case '8':           rank = 6; break;
            case '9':           rank = 7; break;
            case 't': case 'T': rank = 8; break;
            case 'j': case 'J': rank = 9; break;
            case 'q': case 'Q': rank = 10; break;
            case 'k': case 'K': rank = 11; break;
            case 'a': case 'A': rank = 12; break;
            default:            bad_card = true;
        }
```

Solution



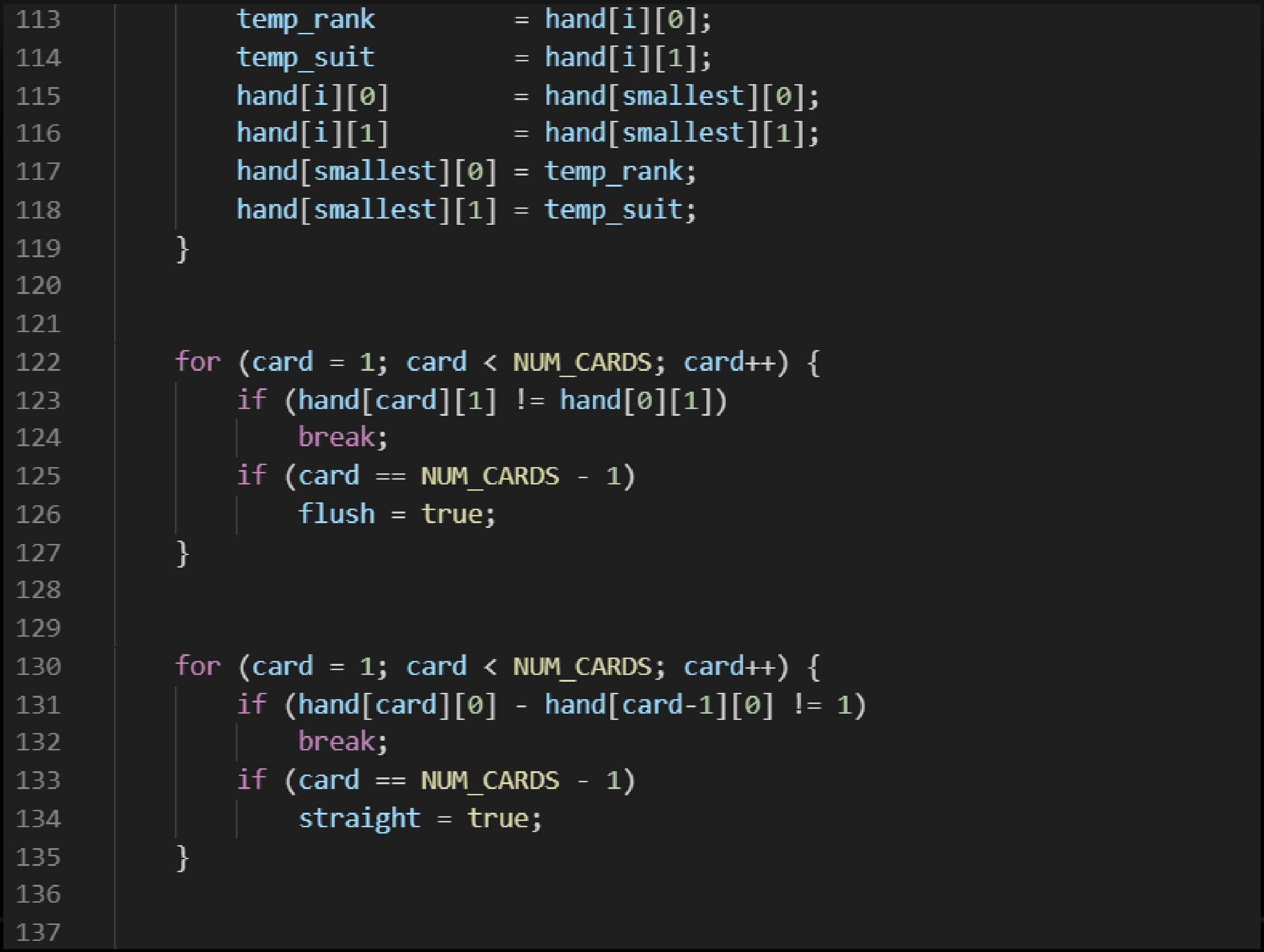
```c
            suit_ch = getchar();
            switch (suit_ch) {
                case 'c': case 'C': suit = 0; break;
                case 'd': case 'D': suit = 1; break;
                case 'h': case 'H': suit = 2; break;
                case 's': case 'S': suit = 3; break;
                default:            bad_card = true;
            }

            while ((c = getchar()) != '\n')
                if (c != ' ') bad_card = true;

            if (bad_card)
                printf("Bad card; ignored.\n");
            else if (duplicate_card(rank, suit, hand, cards_read))
                printf("Duplicate card; ignored.\n");
            else {
                hand[cards_read][0] = rank;
                hand[cards_read][1] = suit;
                cards_read++;
            }
        }
    }
```

Solution



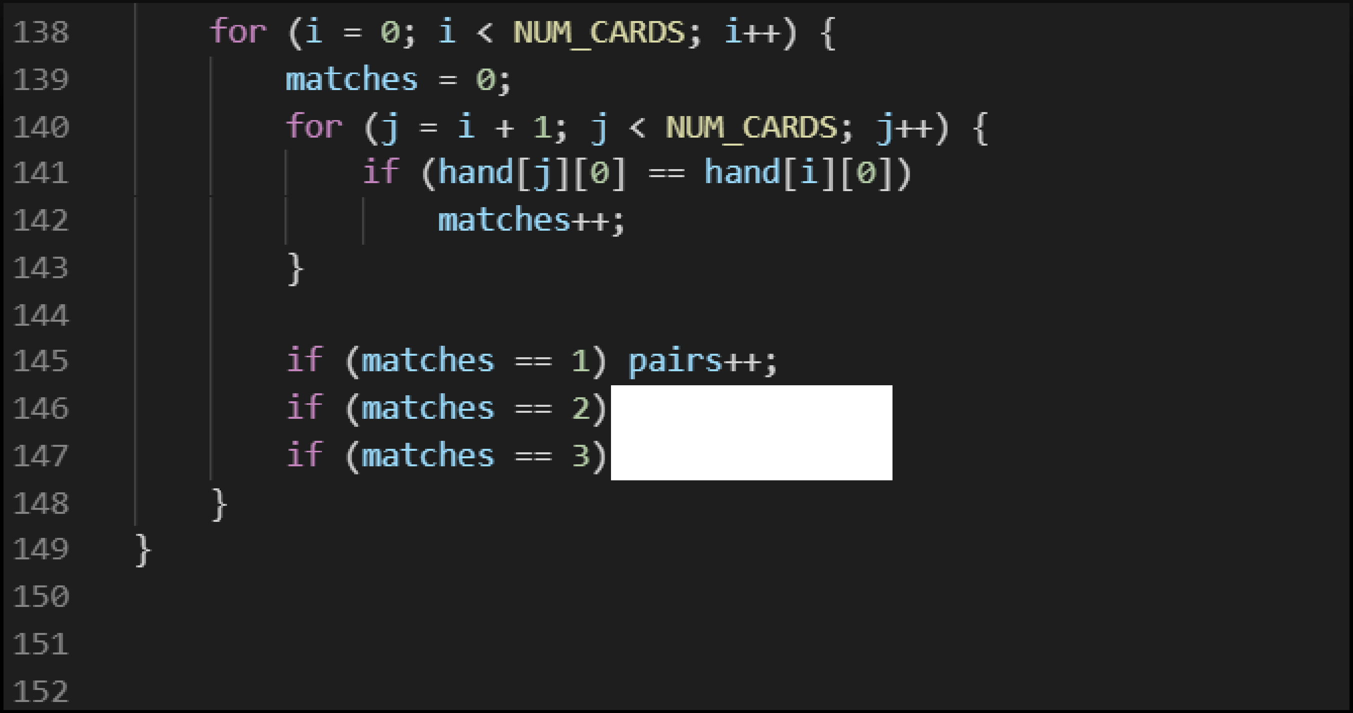
```
bool duplicate_card(int rank, int suit, int hand[NUM_CARDS][2], int cards_read) {
    int i;
    for (i = 0; i < cards_read; i++)
        if ([redacted])
            return true;
    return false;
}


void analyze_hand(void) {
    int num_consec = 0;
    int card, rank, matches;

    straight = false;
    flush = false;
    four = false;
    three = false;
    pairs = 0;


    int i, j, smallest, temp_suit, temp_rank;
    for (i = 0; i < NUM_CARDS; i++) {
        smallest = i;

        for (j = i + 1; j < NUM_CARDS; j++) {
            if (hand[j][0] < hand[smallest][0])
                smallest = j;
        }

```

Solution



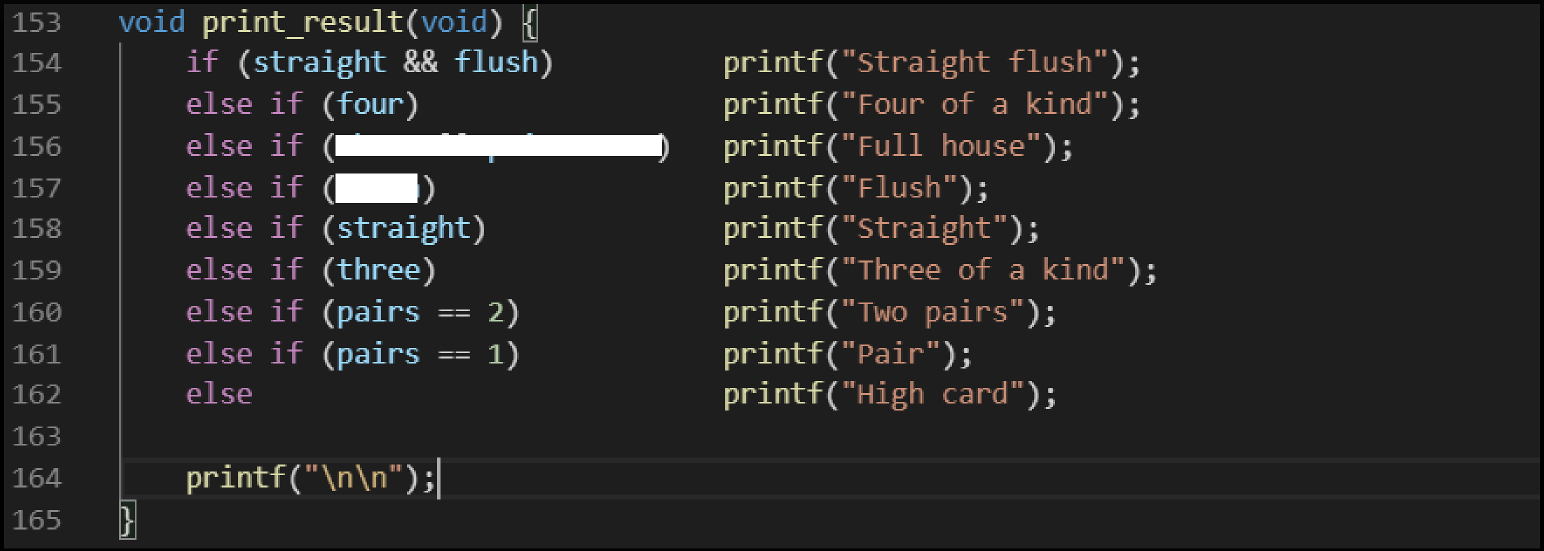
```
        temp_rank         = hand[i][0];
        temp_suit         = hand[i][1];
        hand[i][0]        = hand[smallest][0];
        hand[i][1]        = hand[smallest][1];
        hand[smallest][0] = temp_rank;
        hand[smallest][1] = temp_suit;
    }


    for (card = 1; card < NUM_CARDS; card++) {
        if (hand[card][1] != hand[0][1])
            break;
        if (card == NUM_CARDS - 1)
            flush = true;
    }


    for (card = 1; card < NUM_CARDS; card++) {
        if (hand[card][0] - hand[card-1][0] != 1)
            break;
        if (card == NUM_CARDS - 1)
            straight = true;
    }


```

Solution



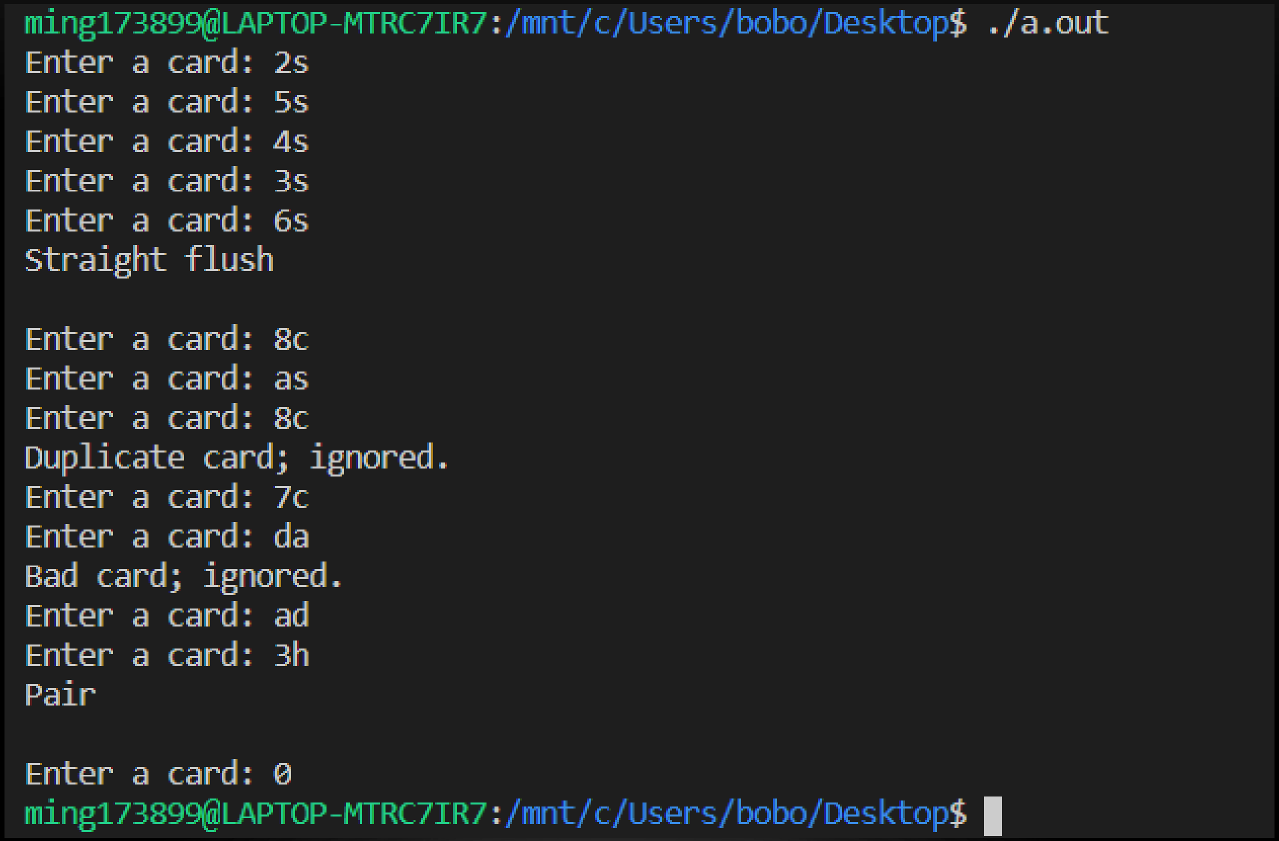
```
    for (i = 0; i < NUM_CARDS; i++) {
        matches = 0;
        for (j = i + 1; j < NUM_CARDS; j++) {
            if (hand[j][0] == hand[i][0])
                matches++;
        }

        if (matches == 1) pairs++;
        if (matches == 2) [redacted]
        if (matches == 3)
    }
}
```

Solution



```c
void print_result(void) {
    if (straight && flush)          printf("Straight flush");
    else if (four)                  printf("Four of a kind");
    else if ([redacted])            printf("Full house");
    else if ([redacted])            printf("Flush");
    else if (straight)              printf("Straight");
    else if (three)                 printf("Three of a kind");
    else if (pairs == 2)            printf("Two pairs");
    else if (pairs == 1)            printf("Pair");
    else                            printf("High card");

    printf("\n\n");
}
```

Example



```
ming173899@LAPTOP-MTRC7IR7:/mnt/c/Users/bobo/Desktop$ ./a.out
Enter a card: 2s
Enter a card: 5s
Enter a card: 4s
Enter a card: 3s
Enter a card: 6s
Straight flush

Enter a card: 8c
Enter a card: as
Enter a card: 8c
Duplicate card; ignored.
Enter a card: 7c
Enter a card: da
Bad card; ignored.
Enter a card: ad
Enter a card: 3h
Pair

Enter a card: 0
ming173899@LAPTOP-MTRC7IR7:/mnt/c/Users/bobo/Desktop$
```

Chapter 10_project 6

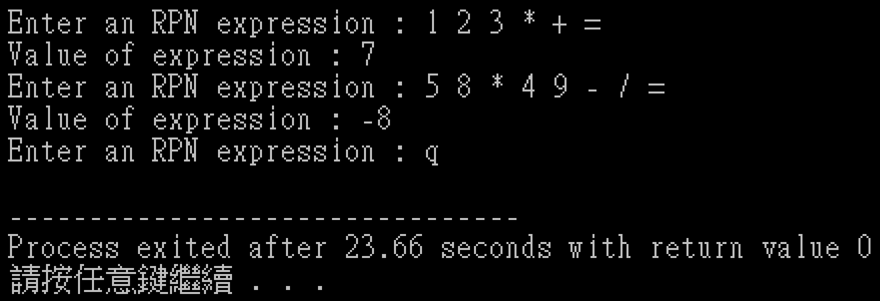
Some calculators (notably those from Hewlett-Packard) use a system of writing mathematic-cal expressions known as Reverse Polish Notation (RPN). In this notation, operators are placed *after* their operands instead *of between* their operands. For example, 1 + 2 would be written 1 2 + in RPN, and 1 + 2 * 3 would be written 1 2 3 * + . RPN expressions can easily be evaluated using a stack. The algorithm involves reading the operators and operands in an expression from left to right, performing the following actions;

When an operand is encountered, push it onto the stack.

When an operator is encountered, pop its operands from the stack, perform the operation on those operands, and then push the result onto the stack.

Chapter 10_project 6

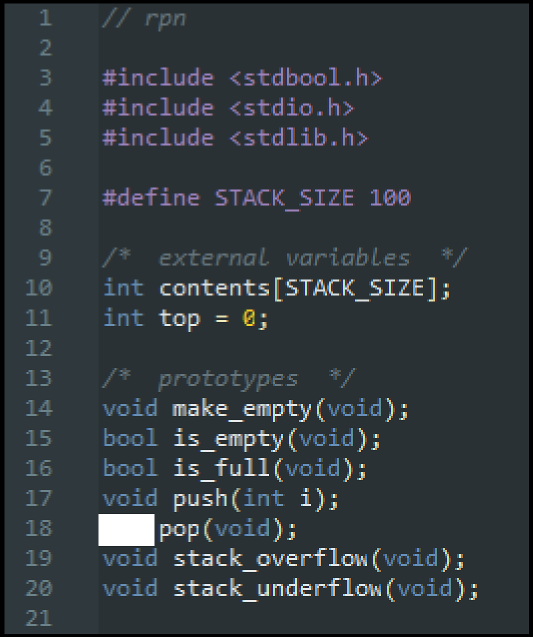
Write a program that evaluates RPN expressions. The operands will be single-digit integers. The operators are +, -, *, / and =. The = operator causes the top stack item to be displayed; afterwards, the stack is cleared and the user is prompted to enter another expression. The process continues until the user enters a character that is not an operator or operand:



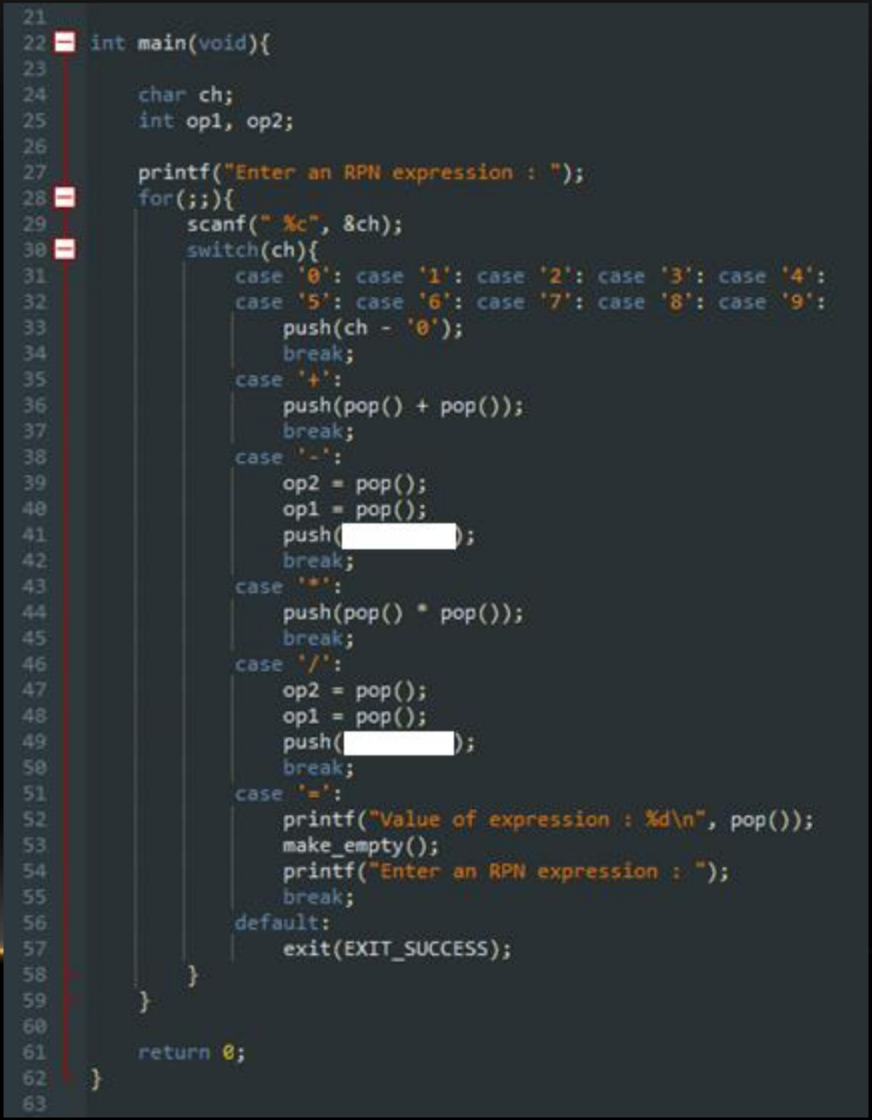
```
Enter an RPN expression : 1 2 3 * + =
Value of expression : 7
Enter an RPN expression : 5 8 * 4 9 - / =
Value of expression : -8
Enter an RPN expression : q

--------------------------------
Process exited after 23.66 seconds with return value 0
請按任意鍵繼續 . . .
```

If the stack overflows, the program will display the message *Expression is too com-plex* and terminate. If the stack undertlows (because of an expression such as 1 2 + +), the program will display the message *Not enough operands in expression* and termi-nate. Hints: Incorporate the stack code from Section 10.2 into your program. Use scanf (" %c", &ch) to read the operators and operands.

Solution

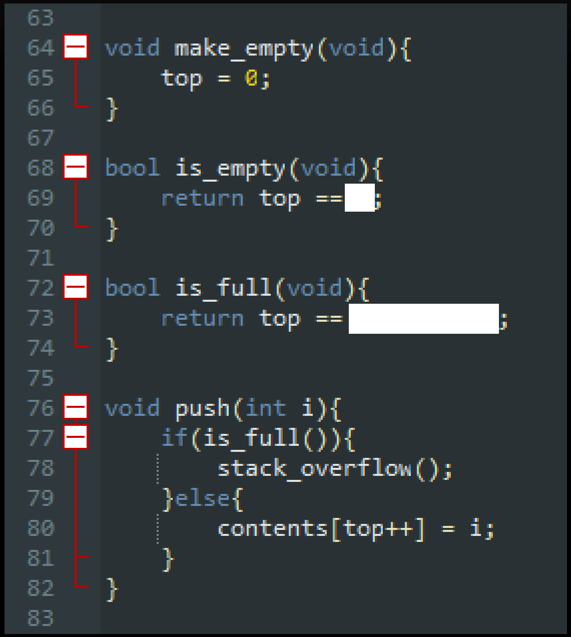


```c
// rpn

#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#define STACK_SIZE 100

/*  external variables  */
int contents[STACK_SIZE];
int top = 0;

/*  prototypes  */
void make_empty(void);
bool is_empty(void);
bool is_full(void);
void push(int i);
     pop(void);
void stack_overflow(void);
void stack_underflow(void);

```

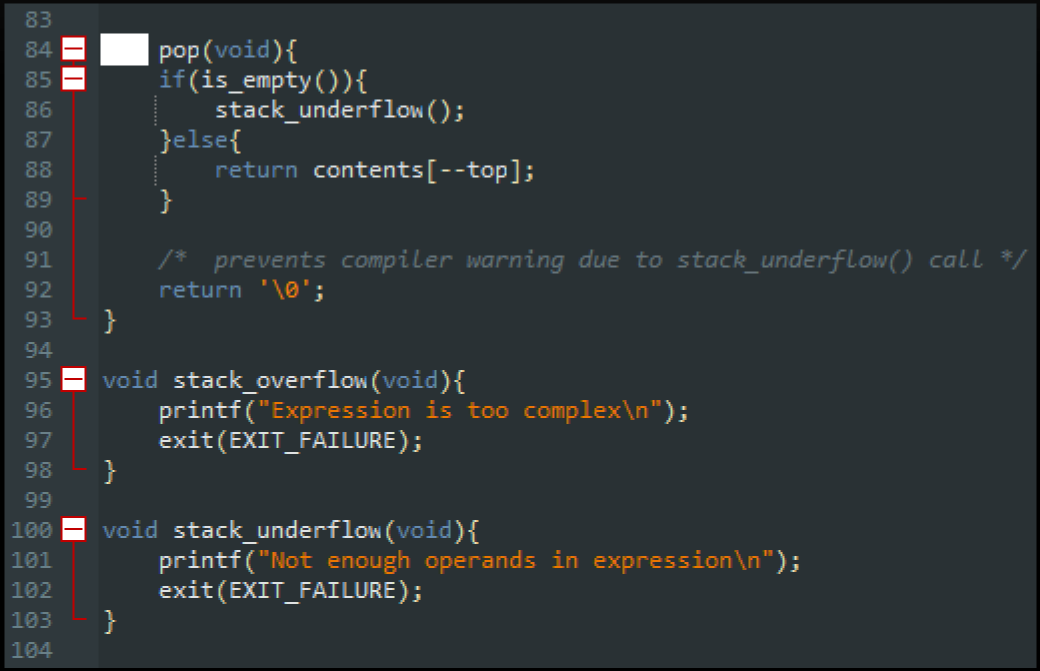


```c

int main(void){

    char ch;
    int op1, op2;

    printf("Enter an RPN expression : ");
    for(;;){
        scanf(" %c", &ch);
        switch(ch){
            case '0': case '1': case '2': case '3': case '4':
            case '5': case '6': case '7': case '8': case '9':
                push(ch - '0');
                break;
            case '+':
                push(pop() + pop());
                break;
            case '-':
                op2 = pop();
                op1 = pop();
                push(          );
                break;
            case '*':
                push(pop() * pop());
                break;
            case '/':
                op2 = pop();
                op1 = pop();
                push(          );
                break;
            case '=':
                printf("Value of expression : %d\n", pop());
                make_empty();
                printf("Enter an RPN expression : ");
                break;
            default:
                exit(EXIT_SUCCESS);
        }
    }

    return 0;
}

```

Solution

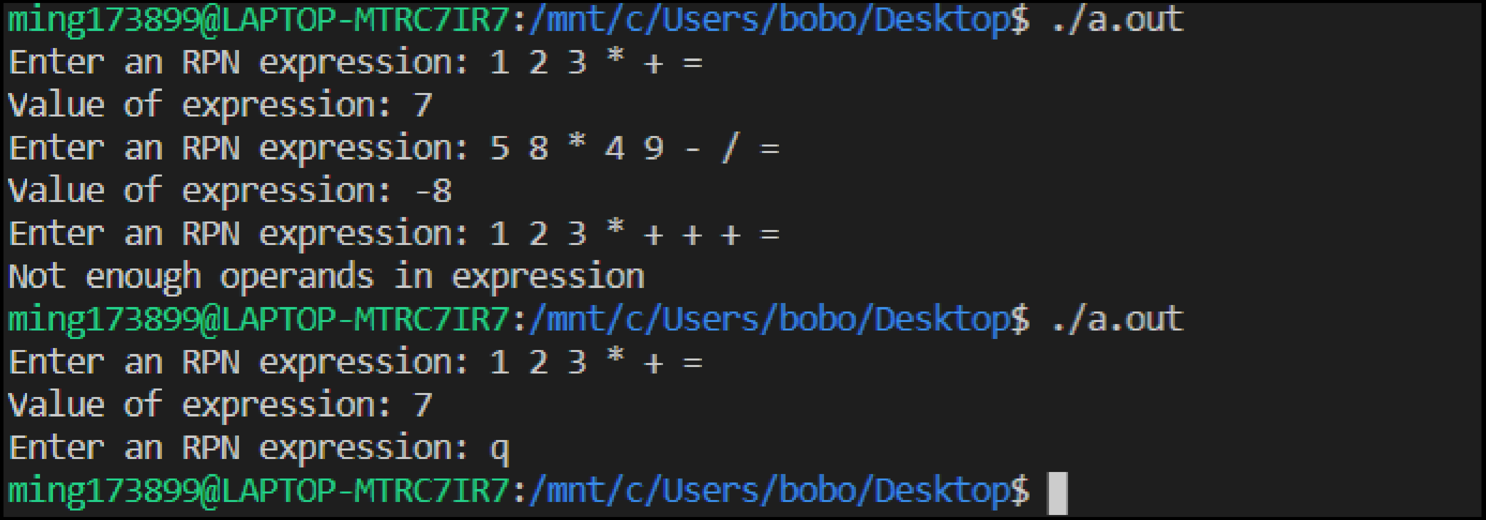


```c

void make_empty(void){
    top = 0;
}

bool is_empty(void){
    return top ==   ;
}

bool is_full(void){
    return top ==             ;
}

void push(int i){
    if(is_full()){
        stack_overflow();
    }else{
        contents[top++] = i;
    }
}

```



```c

     pop(void){
    if(is_empty()){
        stack_underflow();
    }else{
        return contents[--top];
    }

    /*  prevents compiler warning due to stack_underflow() call */
    return '\0';
}

void stack_overflow(void){
    printf("Expression is too complex\n");
    exit(EXIT_FAILURE);
}

void stack_underflow(void){
    printf("Not enough operands in expression\n");
    exit(EXIT_FAILURE);
}

```

Example



```
ming173899@LAPTOP-MTRC7IR7:/mnt/c/Users/bobo/Desktop$ ./a.out
Enter an RPN expression: 1 2 3 * + =
Value of expression: 7
Enter an RPN expression: 5 8 * 4 9 - / =
Value of expression: -8
Enter an RPN expression: 1 2 3 * + + + =
Not enough operands in expression
ming173899@LAPTOP-MTRC7IR7:/mnt/c/Users/bobo/Desktop$ ./a.out
Enter an RPN expression: 1 2 3 * + =
Value of expression: 7
Enter an RPN expression: q
ming173899@LAPTOP-MTRC7IR7:/mnt/c/Users/bobo/Desktop$
```