

program 1: stack

CODE: #include <stdio.h>

```
char stack[5];
int top = -1;

void push(char ch) {
    if (top >= 4) {
        printf("Stack overflow");
        return;
    }
    stack[++top] = ch;
}

char pop() {
    if (top == -1) {
        printf("Stack underflow");
        return '\0';
    }
    return stack[top--];
}

char peek() {
    return stack[top];
}

char display() {
    for (int i = 0; i <= top; i++) {
        printf("%c", stack[i]);
    }
}

int main() {
    int x, element;
    printf("1: push 2: peek 3: pop 4: display 5: exit");
    scanf("%d", &x);
    while (x != 5) {
        switch (x) {
            case 1:
                printf("Enter an element to push:");
                break;
            case 2:
                printf("Element at top is %c", peek());
                break;
            case 3:
                printf("Popped element is %c", pop());
                break;
            case 4:
                display();
                break;
        }
        printf("\n");
        scanf("%d", &x);
    }
}
```

```

scant ("%c", &element);
push(element);
break; }

case 2: { // to read binary
    peek(); // (P == 0) ??
    break; }

case 3: { // to read binary
    printf(" popped element: %c\n", pop());
    break; }

case 4: { // to read binary
    display();
    break; }

case 5: { // to read binary
    printf("Exiting...\n");
    break; }

default: { // to read binary
    printf("Wrong choice! Please try again.\n");
    break; }

printf ("1:push 2:peek 3:pop 4:display 5:exit");
scant ("%c", &choice);
if (choice == 1) {
    push();
}
else if (choice == 2) {
    peek();
}
else if (choice == 3) {
    pop();
}
else if (choice == 4) {
    display();
}
else if (choice == 5) {
    exit(0);
}
else {
    printf("Wrong choice! Please try again.\n");
}
}

```

Output: 1: push 2: peek 3: pop 4: display 5: exit 1
Enter an element to push: 1
1: push 2: peek 3: pop 4: display 5: exit 2
top element: 1
1: push 2: peek 3: pop 4: display 5: exit 3
popped element: 1
1: push 2: peek 3: pop 4: display 5: exit 1
Element to be pushed: 2

1: push 2: peek 3: pop 4: display 5: exit 4

ab 1: push 2: peek 3: pop 4: display 5: exit 3
popped element b1: push 2: peek 3: pop 4: display 5: exit 3
popped element a1: push 2: peek 3: pop 4: display 5: exit 3
Stack underflow1: push 2: peek 3: pop 4: display 5: exit 5
Exiting...~~if (first > end) break and return~~~~i = i + 1~~~~if (arr[i] < arr[i+1])~~~~arr[i] = arr[i+1] arr[i+1] = arr[i]~~~~if (arr[i] < arr[i+1])~~~~arr[i] = arr[i+1] arr[i+1] = arr[i]~~~~if (arr[i] < arr[i+1])~~~~if (arr[i] == arr[i+1])~~~~0 number~~~~if (arr[i] < arr[i+1])~~~~if (arr[i] == arr[i+1])~~~~if (arr[i] == arr[i+1])~~~~1 number~~~~0 number == 1~~

E73 : program to print

stack elements in reverse

program 2: balanced parenthesis

```

code: int is_matching_pair (char ch1; char ch2) {
    if (ch1 == 'c' && ch2 == ')') {
        return 1;
    } else if (ch1 == '[' && ch2 == ']') {
        return 1;
    } else if (ch1 == '{' && ch2 == '}') {
        return 1;
    } else {
        return 0;
    }
}

```

```

int is_balanced(char* text) {
    int i;
    for(i=0; i<strlen(text); i++) {
        if (text[i] == '(' || text[i] == '[' || text[i] == '{')
            push(text[i]);
        else if (text[i] == ')' || text[i] == ']' || text[i] == '}')
            if (top == -1)
                return 0;
            else if (!is_matching_pair(pop(), text[i]))
                return 0;
        if (top == -1)
            return 1;
        else
            return 0;
    }
}

```

output

Enter an expression: {[3]}

Balanced parenthesis