**Module-1**

1. **Sum and difference of two numbers**

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

int main()

{

int a,b;

float c,d;

scanf("%d %d",&a,&b);

scanf("%f %f",&c ,&d);

int int\_sum = a+b;

int int\_diff = a-b;

float float\_sum = c+d;

float float\_diff = c-d;

printf("%d %d\n",int\_sum,int\_diff);

printf("%0.1f %0.1f", float\_sum,float\_diff);

return 0;

}

A black screen with white text

Description automatically generated

**2. Playing with characters**

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

int main()

{

char ch;

char s[24];

char t[100];

scanf("%c", &ch);

scanf("%s", s);

getchar();

scanf("%[^\n]%\*c", t);

printf("%c\n", ch);

printf("%s\n", s);

printf("%s\n", t);

return 0;

}

A screenshot of a computer

Description automatically generated

**3.Conditional statements in C**

#include <assert.h>

#include <limits.h>

#include <math.h>

#include <stdbool.h>

#include <stddef.h>

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char\* readline();

int main()

{

char\* n\_endptr;

char\* n\_str = readline();

int n = strtol(n\_str, &n\_endptr, 10);

if (n\_endptr == n\_str || \*n\_endptr != '\0') { exit(EXIT\_FAILURE); }

if (n==1){

printf("one");

}

else if(n==2){

printf("two");

}

else if(n==3){

printf("three");

}

else if(n==4){

printf("four");

}

else if(n==5){

printf("five");

}

else if(n==6){

printf("six");

}

else if(n==7){

printf("seven");

}

else if(n==8){

printf("eight");

}

else if(n==9){

printf("nine");

}

else if(n>9){

printf("Greater than 9");

}

return 0;

}

char\* readline() {

size\_t alloc\_length = 1024;

size\_t data\_length = 0;

char data[1024];

while (true) {

char\* cursor = data + data\_length;

char\* line = fgets(cursor, alloc\_length - data\_length, stdin);

if (!line) { break; }

data\_length += strlen(cursor);

if (data\_length < alloc\_length - 1 || data[data\_length - 1] == '\n') { break; }

size\_t new\_length = alloc\_length << 1;

if (!data) { break; }

alloc\_length = new\_length;

}

if (data[data\_length - 1] == '\n') {

data[data\_length - 1] = '\0';

}

return data;

}

**4. Valid Paranthesis**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_SIZE 100

// Global variables for stack and top

char stack[MAX\_SIZE];

int top = -1;

// Function to push a character onto the stack

void push(char data) {

if (top == MAX\_SIZE - 1) {

printf("Overflow stack!\n");

return;

}

top++;

stack[top] = data;

}

// Function to pop a character from the stack

char pop() {

if (top == -1) {

printf("Empty stack!\n");

return ' ';

}

char data = stack[top];

top--;

return data;

}

// Function to check if two characters form a matching pair of parentheses

int is\_matching\_pair(char char1, char char2) {

if (char1 == '(' && char2 == ')') {

return 1;

} else if (char1 == '[' && char2 == ']') {

return 1;

} else if (char1 == '{' && char2 == '}') {

return 1;

} else {

return 0;

}

}

// Function to check if the expression is balanced

int isBalanced(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; // If no opening bracket is present

} else if (!is\_matching\_pair(pop(), text[i])) {

return 0; // If closing bracket doesn't match the last opening bracket

}

}

}

if (top == -1) {

return 1; // If the stack is empty, the expression is balanced

} else {

return 0; // If the stack is not empty, the expression is not balanced

}

}

// Main function

int main() {

char text[MAX\_SIZE];

printf("Input an expression in parentheses: ");

scanf("%s", text);

// Check if the expression is balanced or not

if (isBalanced(text)) {

printf("The expression is balanced.\n");

} else {

printf("The expression is not balanced.\n");

}

return 0;

}

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer screen

Description automatically generated**

**5. Bitwise Operators**

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

void calculate\_the\_maximum(int n, int k) {

   int max\_and = 0, max\_or = 0, max\_xor = 0;

    for (int i = 1; i <= n; i++) {

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

            int temp\_and = i & j;

            int temp\_or = i | j;

            int temp\_xor = i ^ j;

            if (temp\_and > max\_and && temp\_and < k) {

                max\_and = temp\_and;

            }

            if (temp\_or > max\_or && temp\_or < k) {

                max\_or = temp\_or;

            }

            if (temp\_xor > max\_xor && temp\_xor < k) {

                max\_xor = temp\_xor;

            }

        }

    }

    printf("%d\n%d\n%d", max\_and, max\_or, max\_xor);

}

int main() {

    int n, k;

    scanf("%d %d", &n, &k);

    calculate\_the\_maximum(n, k);

    return 0;

}

**A black screen with white x and white crosses

Description automatically generated**