1. (a) Describa el funcionamiento general del Sistema Operativo. Referencia: Relationship between Operating System, Computer Hardware, Application Software, and Other Software; An Overview of Computer Operating Systems and Emerging Trends

   (b) Ventajas y desventajas de la programación Multicore y Multiprocesadores: `https://www.tutorialspoint.com/multiprocessor-and-multicore-organization`.

2. (a) ¿Cuál es la diferencia entre los formatos `%i` y `%d`? Dé un ejemplo.

   Cuando se especifica el formato en las funciones

   - `scanf`
   - `fscanf`
   - `sscanf`
   - `scanf_s`
   - `fscanf_s`
   - `sscanf_s`
   - `vscanf`
   - `vfscanf`
   - `vsscanf`
   - `vscanf_s`
   - `vfscanf_s`
   - `vsscanf_s`,

   la diferencia entre `%i` y `%d` es la base que asume en la que se escribio el número insertado por el usuario. `%d` asume que es un número decimal (base 10), mientras que `%i` infieire la base. Es decir, si insertamos `29`, asume base 10, mientras que si insertamos `0x1D` asume base 16. Ver [4, 6].

   Por otra parte, estos son equivalentes al usar las funciones de output e.g. `printf` [3, 5].

   (b) ¿Cuál es la diferencia entre la declaración `bool` y `_Bool` en C?

   Previo a C23, `bool` no era reconocida como una palabra reservada (keyword), por lo que se debía importar `stdbool.h` para que fuese reconocida como una palabra reservada. El propósito de `bool` es simplemente ser mapeada (es decir, un macro) a `_Bool`, que ha existido dato primitivo desde la primer versión de C. Asimismo, `true` era un macro para 1, mientras que `false` era un macro para 0.

   En C23, ya no es necesario el uso de `stdbool.h`, ya que `bool` es una palabra reservada. [1]

3. (a) ¿Qué pasa si al leer un entero con scanf(), el usuario teclea el número seguido con una letra? Ejem: 67f, ¿Cómo explica el resultado?

`stdin`, que es de tipo FILE*, es el lugar donde se guarda lo que insertamos a la consola. Lo que hace `scanf` con formato `%d` es buscar, comenzando por el primer caracter al que apunta `stdin`, un entero válido. Este se detiene cuando ya no se apunta a lo que considera parte del entero. En este caso, lee `67`, y reconoce que el char `f` ya no forma parte del formato que especificamos. Sin embargo, el char `f` sigue en `stdin`, y es el caracter que queda apuntado.

(b) Enseguida de la instrucción anterior, añada ahora la lectura de un carácter, ¿Qué pasa y cómo explica este comportamiento?

Retomando de la respuesta anterior, ahora el siguiente `scanf` lee de `stdin`, que por lo que dijimos, apunta a algo no nulo. En este caso, al char `f`. Entonces, `scanf` lee el char `f` [2].

4. Programa que realice una operación aritmética especificada entre dos fracciones. La entrada debe ser de la forma: $a/b \oplus c/b$, donde $\oplus \in \{+, -, *, /\}$.

Para compilar este codigo, hacer `gcc p4.c` y correr directamente, sin argumentos de consola. Adentro, habrá un ciclo infinito que permite al usuario teclear fracciones en formato $a/b(*,+,-,/)c/d$ y ver su resultado. Por ejemplo, introducir `1/2*2/4` imprime `1/2*2/4 = 2/8 = 0.250000`.

El programa imprime errores cuando se introducen caracteres incorrectos al scanf o un operador no reconocible.

Listing 1: Programa que realiza operaciones de fracciones

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

int main() {
  // ciclo infinito para hacer muchas pruebas sin tener que
     correr
  // el programa de nuevo
  while (1) {
    printf("type an operation between two fractions.\nformat: 'a
       /b(*,/,+,-)c/d'.\n\n");
    int a, b, c, d; // enteros que formaran parte de las
       fracciones a operar
    char op; // operador a utilizar
    int numerator, denominator; // variables con el unico fin de
       representar la fraccion en forma a/b
    // solamente entraremos al parseo de numeros si se han
    // introducido los 5 objetos en el formato correcto
    if (scanf("%d/%d%c%d/%d", &a, &b, &op, &c, &d) == 5) {
      // switch para determinar el operador insertado y realizar
```

```
            operacion
        // de numerador y denominador por separado
        switch (op) {
        case 42: // char 42 == '*'
          numerator = a * c;
          denominator = b * d;
          break;
        case 43: // char 43 == '+'
          numerator = a * d + b * c;
          denominator = b * d;
          break;
        case 45: // char 45 == '-'
          numerator = a * d - b * c;
          denominator = b * d;
          break;
        case 47: // char 47 == '/'
          numerator = a * d;
          denominator = c * b;
          break;
        default:
          fprintf(stderr, "invalid operator.\n"); // si no se
              encontro un operador soportado, imprimir un error
          exit(1);
        }
        // impresion en formato a/b y como decimal.
        printf("%d/%d%c%d/%d=", a, b, op, c, d);
        printf("%d/%d=", numerator, denominator);
        printf("%f\n", (float)(numerator) / (denominator));
      }
      else {
        // si los tipos de argumento no coinciden, imprimir un
            error
        fprintf(stderr, "invalid arguments.\n");
        return 1;
      }
    }
}
```

5. Programa que imprima un número entero dado de $n$ digitos al revés.

Para compilar el programa, usar gcc p5.c y ejecutar sin argumentos de consola. El programa lee un string y recorre el arreglo en dirección contraria, imprimiendo el entero al revés. Tomamos como maximo 10 digitos, ya que $2^31 - 1$ tiene 10 digitos.

Listing 2: Programa que imprime un número entero al revés

```c
#include <stdio.h>
#define MAX_DIGITIS 10 // 2^31 is 10 digits

int main() {
  // inicializar en el char 0 (NULL)
  char input[MAX_DIGITIS] = {0};
  printf("Enter a number:\n");
  // guardar el string en el array input
  scanf("%s", input);

  // loop en direccion contraria
  for (int i = MAX_DIGITIS - 1; i >= 0; i--) {
    // imprimir solo si no es el caracter nulo
    if (input[i] != 0) {
      printf("%c", input[i]);
    }
  }
  printf("\n");
}
```

6. Programa que evalúe la expresión

$$e^{-x^2} = \sum_{n=0}^{\infty} \frac{(-1)^n}{n!} x^{2n}.$$

Debe pedir el número de términos a evaluar.

Para compilar, usar `gcc p6.c` y ejecutar sin argumentos de consola. El programa primero pide un punto de precisión doble, luego un número natural $n$ de términos que se usarán para aproximar $\exp(-x^2)$.

Listing 3: Programa que calcula la serie de $\exp(-x^2)$

```c
#include <stdio.h>

// recibe un punto x real y un numero entero de terminos
// para aproximar e^(-x^2)
double taylor_exp(double x, unsigned int terms) {
  // inicializamos el resultado como 0
  // n_x_squared es -x^2, que usaremos para calcular
  // el siguiente termino de la serie,
  // notando que si el termino n-1 es
  // t_{n-1} = (-1)^(n-1)x^(2(n-1))/(n-1)!,
  // entonces el termino n es
  // t_{n} = t_{n-1} * (-1)x^2/n.
```

```
  double result = 0, n_x_squared = -x * x;
  double curr_term = 1; // el primer termino de la serie es 1
     para cualquier x
  for (int i = 1; i < terms; i++) {
    result += curr_term;
    curr_term *= n_x_squared / i; // actualizamos el valor
       siguiendo el algoritmo mencionado anteriormente
  }
  return result;
}

int main() {
  double x;
  unsigned int terms;
  printf("Enter a point x:\n");
  scanf("%lf", &x);
  printf("Enter number of terms to approximate exp(-x^2):\n");
  scanf("%ud", &terms);
  printf("%lf\n", taylor_exp(x, terms));
}
```

7. Programa que convierta un número decimal a cualquier base.

Para compilar, usar `gcc p7.c`, ejecutar sin argumentos de consola. El programa pedirá un numero decimal y una base a la cual convertir. Por ejemplo, insertar `24.3 5` imprime `24.299999 base 10 = 44.1222222210 base 5`. La parte fraccionaria tiene un limite de 10 iteraciones para converger, si no, hay una truncación.

Listing 4: Programa que convierte un número decimal a uno de cualquier base

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

#define LOOP_LIMIT 10

int main(int argc, char *argv[]) {
  float decimal;
  unsigned int base;

  printf("Enter deciaml and base to convert to.\n");
  printf("For example: 24.3 5 converts 24.3 to base 5.\n");
  printf("To quit, press Ctrl+c\n");
  while (1) {
    if(scanf("%f %ui", &decimal, &base) != 2) {
      fprintf(stderr, "Invalid decimal or base.\n");
```

```
    exit(1);
  }

  // separate into integer and fractional parts of decimal
  int d_int = (int)decimal;
  float d_frac = decimal - d_int;

  // we first process the integral part of the input
  int q = d_int, i = 0, j;
  div_t d;

  // store items in list for printing. a very rough estimate
     of the amount
  // of space needed for the integer part is d_int / base.
  char result[q / base];
  do {
    d = div(q, base);
    q = d.quot;
    result[i] = d.rem + '0'; // store remainder for later
       printing

    i++;
  } while (q > 0); // the cycle ends when we obtain a quotient
     of 0
  printf("%f base 10 =", decimal);
  // print items in list in reverse order (starting from last
     recorded and not last index)
  j = i - 1;
  while (j >= 0) {
    printf("%c", result[j]);
    j--;
  }

  // process fractional part of the decimal
  int iter = 0, integral_part;
  float frac = d_frac;

  if (d_frac > 0) {
    printf(".");
    // multiply fractional part until 0
    // or iterations exceed our set limit
    // the integral parts that arise are
    // the ones we take for the representation
    while (iter < LOOP_LIMIT && frac) {
      frac *= base;
```

```
            integral_part = (int)frac;
            frac = frac - integral_part;

            iter++;
            printf("%d", integral_part);
          }
        }

        printf("-base-%i", base);
        printf("\n");
    }
}
```

8. Programa que acepte una fracción del tipo $a/b$ tal que $a/b \in \mathbb{Z}$ y la convierta a una fracción irreducible.

Compilar con `gcc p8.c`, correr sin argumentos de consola. Inicia pidiendo una fracción en forma $a/b$, si el formato es correcto, calcula la fracción simplificada. Si no, no imprime nada. Utiliza el algoritmo de Euclides para calcular el gcd, luego divide numerador y denominador entre ese término. Usamos `long int` para permitir números de mayor magnitud.

Listing 5: Programa que simplifica una fracción usando el algoritmo de Euclides

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

// NOTE: usage of long ints was merely to check
// difference between algorithms. No substantial
// time difference was found with the current
// implementation.


// gcd obtained using euclid's algorithm
// a = q_1 b + r_1
// b = q_2 r_1 + r_2
// r_1 = q_3 r_2 + r_3
// ...
// r_n = q_{n-2} r_{n-1}
// we iterate until remainder is zero
long int gcd(long int a, long int b) {
  // we assume a > b
  if (a < b) {
    return gcd(b, a);
  }
```

```
  long int temp;
  while (b) {
    temp = b;
    b = a % b;
    a = temp;
  }
  return a;
}

// returns minimum between two numbers
long int min(long int a, long int b) { return (a < b) ? a : b; }

// slight modification to euclid's algorithm, where
// the residue may be negative, and one takes the remainder
// closes to zero. The number of steps is reduced, but more
// comparisons must be done
long int least_remainder_gcd(long int a, long int b) {
  if (a < b) {
    return least_remainder_gcd(b, a);
  }
  long int temp, r;
  while (b && b != a) {
    temp = b;
    r = a % b;
    b = min(labs(r - b), r); // calculates the remainder closest
        to 0.
    a = temp;
  }
  return a;
}

int main() {
  long int a, b, g;
  if (scanf("%ld/%ld", &a, &b) == 2) {
    g = least_remainder_gcd(a, b);
    printf("%ld/%ld\n", a/g, b/g);
  }
}
```

## References


[1] cppreference.com. Arithmetic types — Boolean type, 2025. Accessed: 2025-08-20.

[2] cppreference.com. std-streams, 2025. Accessed: 2025-08-20.

[3] cppreference.com. `fprintf`, 2025. Accessed: 2025-08-20.

[4] cppreference.com. `fscanf`, 2025. Accessed: 2025-08-20.

[5] cppreference.com. `vfprintf`, 2025. Accessed: 2025-08-20.

[6] cppreference.com. `vfscanf`, 2025. Accessed: 2025-08-20.