

Análisis y Diseño de la Práctica 1

¿Qué es un hexadecimal?

Es un sistema posicional de base 16, osea que tiene 16 dígitos en vez de 10, que se usa en la informática para las operaciones del cpu moderna, ya que estos facilita la lectura de una gran cantidad de números binarios.

¿Qué es un Octal?

Es un sistema posicional de base 8, osea que tiene 8 dígitos en vez de 10, que se utilizaba inicialmente en computación cuando se agrupaban 3 bits para leer números binarios.

1. Descripción

Crear un algoritmo que tome dos números, uno en forma binaria y uno en forma hexadecimal, y un operador (+, -, x, /, ^) y calcule el resultado en forma decimal y octal de la operación entre los dos números.

2. Describir Entradas y Salidas

- Entradas: El código toma como entrada, al llamar el ejecutable, un número binario, un número hexadecimal y un operador en cualquier orden.
- Salidas: El código imprime el resultado de la operación en sistema decimal y en sistema octal.

3. Identificar Procesos

Proceso 1: Verificación del argc

Proceso 2: Asignación dinámica de memoria

Proceso 3: Verificación de signo válido

Proceso 4: Asignación correcta de valores a variables dependiendo de posición de argc

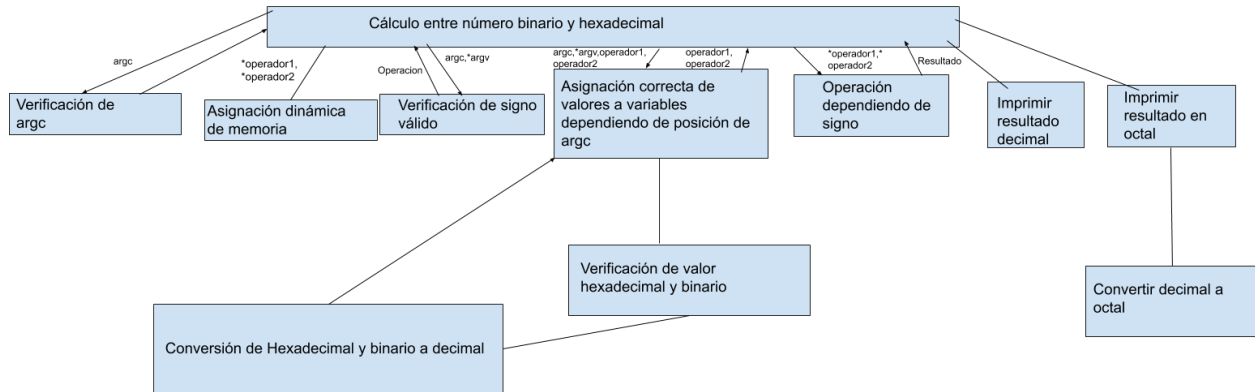
Proceso 5: Operación dependiendo del signo elegido

Proceso 6: Impresión de resultados

4. Pasos de Solución de Algoritmo

1. Ingresar datos en línea de comando
2. Revisar que datos sean válidos
3. Leer datos de manera correcta
4. Realizar cálculo indicado
5. Imprimir resultado en decimal y octal
6. Fin del Programa

5. Diagrama IPO



6. Algoritmo en Pseudocódigo

Procedimiento misDefiniciones.h

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <string.h>
#include <limits.h>

#define True 1
#define False 0
```

Algoritmo 1: calcula.c

Input : Un set de tres entradas, un número hexadecimal, uno binario y una operación básica (+, -, x, /).

Output : Imprime el resultado de la operación en decimal y en octal.

Parameter:

Integer: i, argc

Pointer Integer: operador1 \leftarrow malloc(int), operador2 \leftarrow malloc(int), error_val \leftarrow malloc(int)

Pointer Char: operacion \leftarrow malloc(char)

Double Pointer Char: argv

Float: resultado

```

begin
  if argc = 4 then
    write("Número incorrecto de entradas, se esperan 3 (sin incluir el ejecutable)
      y se encontraron %d", argc)
  end_if
  else
    encontrarOperacion(argc, argv, operacion, error_val)
    if error_val = 0 then
      if (i = 1) then
        | pasrseEntradas(argv[2], argv[3], operador1, operador2, error_val)
      end_if
      else if (i = 2) then
        | pasrseEntradas(argv[1], argv[3], operador1, operador2, error_val)
      end_if
      else
        | pasrseEntradas(argv[1], argv[2], operador1, operador2, error_val)
      end_if
      resultado  $\leftarrow$  hacerOperacion(operador1, operador2, operacion,
        error_val)
    end_if
    free(operador1)
    free(operador2)
    free(operacion)
    if error_val = 0 then
      | write(Resultado en Decimal:", resultado) write(Resultado en Octal: ",
        | decimalAOctal ((int) resultado)
    end_if
  end_if
  free(error_val)
  return 0
end

```

6.1. Procedimientos en funciones.c

Utiliza misDefiniciones.h

Procedimiento integer encontrarOperacion(integer argc, character **argv, character *operacion, integer *error_val)

Input : Encuentra el caracter de la operación.

Output : Regresa el valor que ocupa el caracter.

Parameter:

Integer: i

Boolean: condicion_longitud, condición_operador

begin

for ($i \leftarrow 1$ to argc) **do**

 condicion_longitud \leftarrow strlen(argv[i] = 1)

 condicion_operador \leftarrow (*argv[i] = 'x') **OR** (*argv[i] = '/') **OR** (*argv[i] = '+') **OR** (*argv[i] = '-') **OR** (*argv[i] = '^')

if (condicion_longitud **AND** condicion_operador) **then**

 *operacion \leftarrow argv[i][0]

break;

end_if

end_for

if (*operacion = "") **then**

 write("No se metió un operador válido")

 *error_val \leftarrow 1

end_if

return i

end

Procedimiento integer esBinario(character *arg)

Input : Toma una cadena.

Output : Regresa un booleano, 0 si no es binario y 1 si sí es binario.

Parameter:

Integer i \leftarrow 0

begin

if $arg[i] = -$ **then**

 i \leftarrow i + 1

endif

if $(arg[i] = 0)$ **AND** $(arg[i + 1] = b)$ **then**

for i \leftarrow i + 2 **to** strlen(arg) **do**

if (**NOT** $((arg[i] = '0')$ **OR** $(arg[i] = '1'))$) **then**

return False

endif

end

endif

else

for i \leftarrow i **to** strlen(arg) **do**

if (**NOT** $((arg[i] = 0)$ **OR** $(arg[i] = 1))$) **then**

return False

endif

end

endif

return True

end

Procedimiento integer esHexadecimal(character *arg)

Input : Toma una cadena.

Output : Regresa un booleano, 0 si no es un número hexadecimal y 1 si sí es hexadecimal.

Parameter:

Integer i \leftarrow 0

begin

if $arg[i] = -$ **then**

 i \leftarrow i + 1

endif

if ($arg[i] = 0$) **AND** ($arg[i + 1] = x$) **then**

for i \leftarrow i + 2 **to** strlen(arg) **do**

if (**NOT** isxdigit(arg[i])) **then**

return False

endif

end

endif

else

for i \leftarrow i **to** strlen(arg) **do**

if (**NOT** isxdigit(arg[i])) **then**

return False

endif

end

endif

return True

end

Procedimiento int binarioADecimal(character *arg)

Input : Toma una cadena.

Output : Regresa un número decimal.

Parameter:

Integer cantidad \leftarrow strlen(arg), i \leftarrow 0

Integer: resultado \leftarrow 0

begin

if (arg[i] = -) **then**

 i \leftarrow i + 1

end_if

for i \leftarrow i to cantidad **do**

if (arg[i] = 1) **then**

 resultado \leftarrow resultado + $2^{\text{cantidad}-1-i}$

end_if

end_for

if (arg[0] = -) **then**

 resultado \leftarrow resultado \times -1

end_if

return resultado

end

Procedimiento integer parseEntradas(character *arg1, character *arg2, integer *operador1, integer *operador2, integer *error_val)

Input : Toma dos cadenas, dos apuntadores donde se guardan los números y el apuntador al valor de error.

Output : Regresa un entero y asocia los valores de operador1 y operador2.

Parameter: VOID

begin

if ((esBinario(arg1)) **AND** (esHexadecimal(arg2))) **then**

 *operador1 \leftarrow binarioADecimal(arg1)

 sscanf(arg2, "%x", operador2)

end_if

else if ((esBinario(arg2)) **AND** (esHexadecimal(arg1))) **then**

 *operador1 \leftarrow binarioADecimal(arg2)

 sscanf(arg1, "%x", operador2)

end_if

else

 write("Las entradas numéricas no son correctas")

 *error_val \leftarrow 1

end_if

return 0

end

Procedimiento float hacerOperacion(integer *operador1, integer *operador2, caracter operaci3n, integer *error_val)

Input : Toma dos valores num3ricos, la operaci3n y el valor de error.

Output : Regresa el resultado de la operaci3n.

Parameter:

Float: resultado

```
begin
    switch operacion do
        case +
            resultado  $\leftarrow$  *operador1 + *operador2
            break;
        end
        case -
            resultado  $\leftarrow$  *operador1 - *operador2
            break;
        end
        case x
            resultado  $\leftarrow$  (*operador1) * (*operador2)
            break;
        end
        case /
            resultado  $\leftarrow$  (float)  $\frac{*operador1}{*operador2}$ 
            break;
        end
        case ^
            resultado  $\leftarrow$  (*operador1)(*operador2)
            break;
        end
    end
    return resultado
end
```

Procedimiento integer decimalAOctal(integer valor_decimal)

Input : Toma dos valores numéricos, la operación y el valor de error.

Output : Regresa el resultado de la operación.

Parameter:

Integer: numero_octal \leftarrow , i \leftarrow 1, es_negativo \leftarrow 1

begin

if *valor_decimal* < 0 **then**

 es_negativo \leftarrow -1

 valor_decimal \leftarrow -1

end_if

while *valor_decimal* > 0

 numero_octal \leftarrow numero_octal + (valor_decimal **mod** 8) * i

 valor_decimal \leftarrow valor_decimal / 8

 i \leftarrow i * 10

end_while

 numero_octal \leftarrow numero_octal * es_negativo

return *numero_octal*

end
