

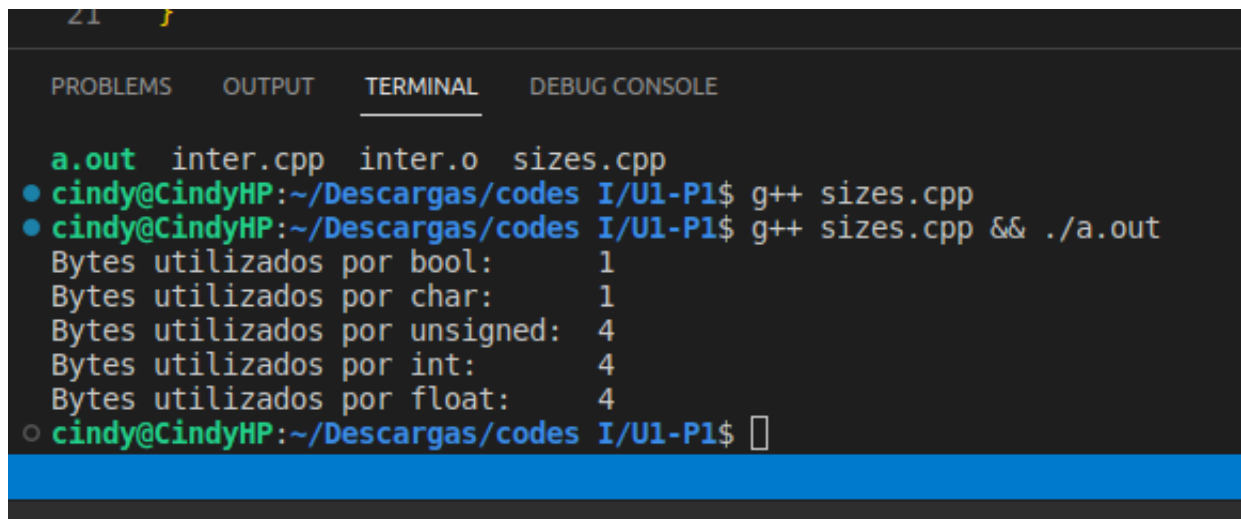
Tarea 1.I

- Compile, corra y analice el funcionamiento de los códigos:
 - sizes.cpp
 - inter.cpp
- Investigue la representación binaria con complemento a 2 para números enteros negativos. Calcule la representación a 32 bits de los siguientes números: -125, -4096, -1000000.

A continuación se presentan las capturas de pantalla donde se compilaron, corrieron y analizaron los códigos especificados.

sizes.cpp

Salida



```
21  f
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

a.out  inter.cpp  inter.o  sizes.cpp
• cindy@CindyHP:~/Descargas/codes I/U1-P1$ g++ sizes.cpp
• cindy@CindyHP:~/Descargas/codes I/U1-P1$ g++ sizes.cpp && ./a.out
Bytes utilizados por bool:      1
Bytes utilizados por char:      1
Bytes utilizados por unsigned:  4
Bytes utilizados por int:        4
Bytes utilizados por float:      4
○ cindy@CindyHP:~/Descargas/codes I/U1-P1$
```

Figura 1: Salida sizes.cpp

Código completo

```
1  #include <cstdlib>
2  #include <iostream>
3
4  using namespace std;
5
6  int main()
7  {
8      bool    boolVar;
9      char    charVar;
10     unsigned unsignedVar;
11     int      intVar;
12     float    floatVar;
13
14     cout<< "Bytes utilizados por bool:\t"      << sizeof(boolVar)      << endl;
15     cout<< "Bytes utilizados por char:\t"      << sizeof(charVar)      << endl;
16     cout<< "Bytes utilizados por unsigned:\t"  << sizeof(unsignedVar) << endl;
17     cout<< "Bytes utilizados por int:\t"       << sizeof(intVar)       << endl;
18     cout<< "Bytes utilizados por float:\t"     << sizeof(floatVar)    << endl;
19
20     return 0;
21 }
```

PROBLEMS OUTPUT **TERMINAL** DEBUG CONSOLE

```
a.out inter.cpp inter.o sizes.cpp
cindy@CindyHP:~/Descargas/codes I/U1-P1$ g++ sizes.cpp
cindy@CindyHP:~/Descargas/codes I/U1-P1$ g++ sizes.cpp && ./a.out
Bytes utilizados por bool:      1
Bytes utilizados por char:     1
Bytes utilizados por unsigned:  4
Bytes utilizados por int:       4
Bytes utilizados por float:     4
cindy@CindyHP:~/Descargas/codes I/U1-P1$
```

Figura 2: Código sizes.cpp

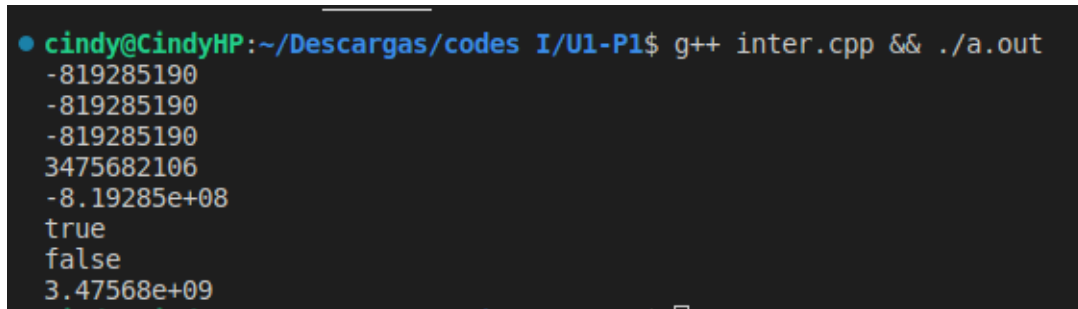
Análisis

Como se puede observar en la figura [1], este código muestra en consola los bytes utilizados por diferentes variables como lo son *bool*, *char*, *unsigned*, *int*, *float*. Se trata de un código sencillo donde:

- De la fila 8-12 se crean las variables tipo *bool*, *char*, *unsigned*, *int* y *float*. A cada una de estas se les da un nombre específico como se muestra en la figura [2].
- De las filas 14-18 se escribieron los comandos para mostrar en pantalla “Bytes utilizados por *tipo de variable”, seguido de un comando que cuenta los bytes que ocupa la variable creada en el paso anterior.

inter.cpp

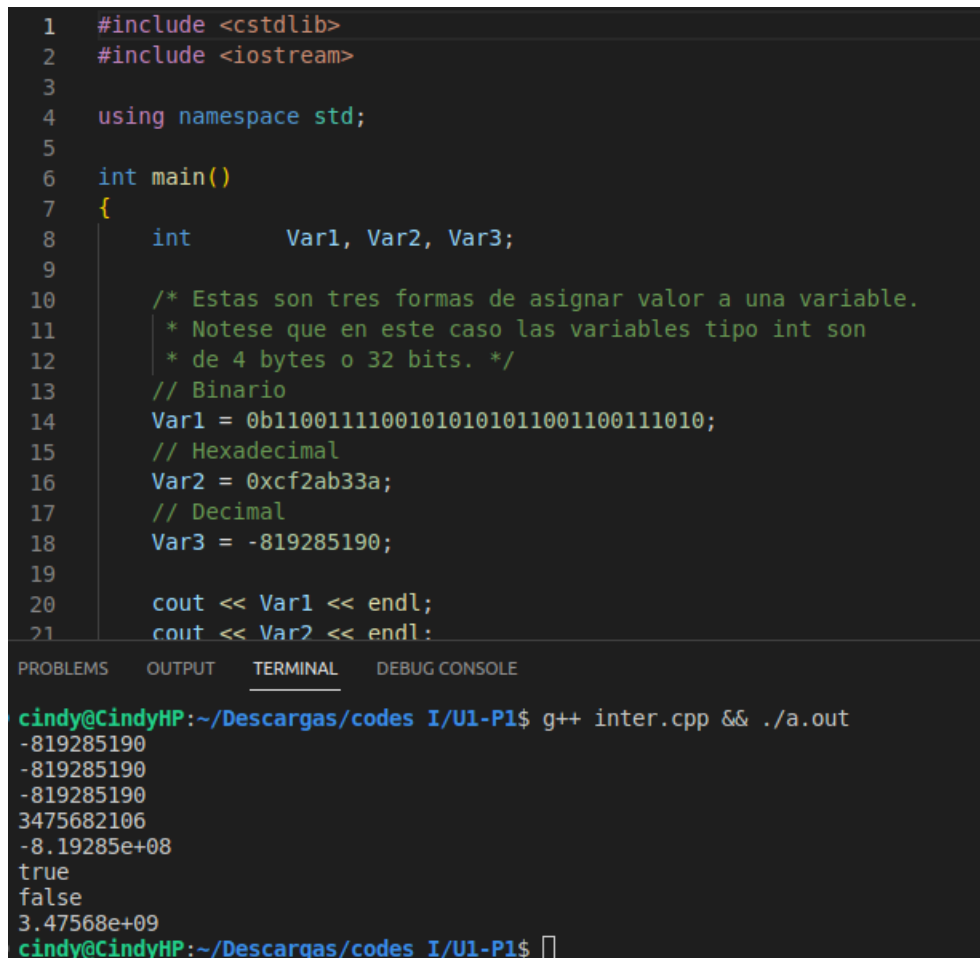
Salida



```
cindy@CindyHP:~/Descargas/codes I/U1-P1$ g++ inter.cpp && ./a.out
-819285190
-819285190
-819285190
3475682106
-8.19285e+08
true
false
3.47568e+09
```

Figura 3: Salida inter.cpp

Código



```
1  #include <cstdlib>
2  #include <iostream>
3
4  using namespace std;
5
6  int main()
7  {
8      int    Var1, Var2, Var3;
9
10     /* Estas son tres formas de asignar valor a una variable.
11      * Notese que en este caso las variables tipo int son
12      * de 4 bytes o 32 bits. */
13     // Binario
14     Var1 = 0b11001111001010101011001100111010;
15     // Hexadecimal
16     Var2 = 0xcf2ab33a;
17     // Decimal
18     Var3 = -819285190;
19
20     cout << Var1 << endl;
21     cout << Var2 << endl;
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
cindy@CindyHP:~/Descargas/codes I/U1-P1$ g++ inter.cpp && ./a.out
-819285190
-819285190
-819285190
3475682106
-8.19285e+08
true
false
3.47568e+09
cindy@CindyHP:~/Descargas/codes I/U1-P1$
```

Figura 4: Código inter.cpp

Análisis

Las tres primeras variables son formas de asignar un valor a una variable. Para la variable 3 es importante notar:

- El valor -819285190 es asignado a una variable tipo entero como notación binaria, hexadecimal y notación decimal.
- El cuarto valor mostrado en consola es el mismo valor como variable tipo entero pero a su interpretación a complemento a 2.
- En la siguiente línea se muestra el valor como punto flotante.
- y en las siguientes líneas se le asigna como valor booleano.
- El último valor corresponde igualmente a una notación como punto flotante pero asignándole el mismo valor binario. Los datos difieren con la otra salida, ya que solamente algunos datos de la expresión binaria son tomadas en cuenta

Representación binaria con complemento a 2 para numeros enteros negativos

El complemento a 1 y a 2 de un número binario son importantes porque permiten la representación de números negativos. El método de complemento a 2 en aritmética es comúnmente usada en computadoras para manipular números negativos.

Para comenzar los números positivos se quedan igual en su representación binaria. Los números negativos se deben invertir el valor de cada una de sus cifras, es decir realizar el complemento a uno, y sumarle 1 al número obtenido. Podemos observar esto en la tabla de ejemplo.

Binario (positivo) - Complemento a 4 (negativo)	Decimal
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

**Complemento a dos con
enteros de 4 bits**

Cabe recordar que debido a la utilización de un bit para representar el signo, el

rango de valores será diferente al de una representación binaria habitual; el rango de valores decimales para n bits será:

$$-2^{n-1} \leq Tango \leq 2^{n-1} \quad (1)$$

Como se mencionó, priemro se debe encontrar el complemento a 1, por lo que a continuación se describirá cómo obtenerlo:

Calcular el complemento a 1

El complemento a 1 de un numero binario es encontrado simplemente cambiando todos los 1s por 0s y todos los 0s por 1s. [1]

Ejemplo:

$$\begin{aligned} \text{Número binario} &= (1010110)_2 = (86)_{10} \\ \text{Complemento a uno} &= (0101001)_2 = (-87)_{10} \end{aligned}$$

Calcular el complemento a 2

El complemento a 2 de un numero binario es encontrado sumando 1 al bit menos significativo de el complemento a 1 del numero.

Ejemplo: Encontrar el complemento a 2 de 10110010

$$\text{Complemento a 1} \rightarrow 01001101$$

$$\begin{array}{r} 01001101 \\ + \quad \quad 1 \\ \hline 01001110 \end{array}$$

Representación a 32bits de

-125

- 00110111 00110101 00111000 00110001 00111001 00111000 00111000 00110011
00110111
- 37 35 38 31 39 38 38 33 37

-4096

- 00110111 00110101 00111000 00110011 00111001 00110100 00111001 00110011
00110111 00100000 00111001 00110000 00110101 00111001 00110110 00111001
00110110 00110110 00110100
- 37 35 38 33 39 34 39 33 37 20 39 30 35 39 36 39 36 36 34

-1000000

- 00110111 00110101 00111000 00110001 00111001 00111000 00110011 00110010
00110000 00100000 00111000 00110000 00111000 00110100 00110110 00110100
00110100 00110011 00110010
- 37 35 38 31 39 38 33 32 30 20 38 30 38 34 36 34 34 33 32

Bibliografía

- [1] TAPIA FABELA JOSE LUIS. Operaciones aritmeticas con numeros binarios signados.