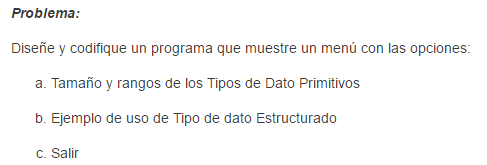
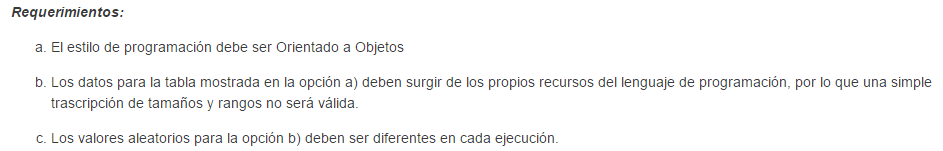
|  |
| --- |
| centro universitario de ciencias exáctas e ingenierías |
| Tipos de Datos Primitivo y Estructurado |
| Tarea 01 |
|  |
| **Aldo Alexandro Vargas Meza 213495653** |
| **24/01/2017** |



|  |
| --- |
| “Reporte de actividad, que consiste en un menú cíclico con 2 opciones y una opción de salir, con tipo de programación orientado a objetos.” |





**Resolución de problema.**

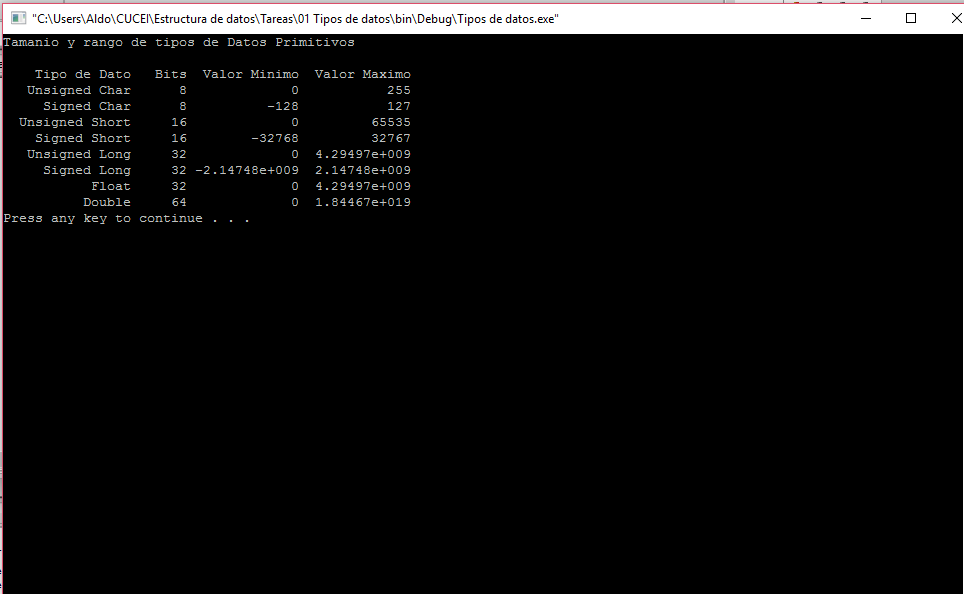
Para la **opción a)** el programa requiere la impresión de las propiedades que manejan los distintos tipos de datos primitivos, y especifica una lista de atributos que es necesario imprimir al usuario, con la condición de no usar una transcripción de texto, y generar los datos por medio de las capacidades del lenguaje de programación.

Para la resolución de este problema, se creó una clase con los atributos nombre, valor mínimo, valor máximo y tamaño en bits. De esta clase derivan dos métodos, los cuales generan la tabla y sus valores y los imprimen en un formato legible.

Para generar los valores, iniciamos obteniendo el tamaño en bits que representa el tipo de dato generado, esto lo hacemos con la función sizeof() que devuelve los bytes de cada tipo de dato. Con este valor, multiplicándolo por 8 obtenemos el tamaño en bits.

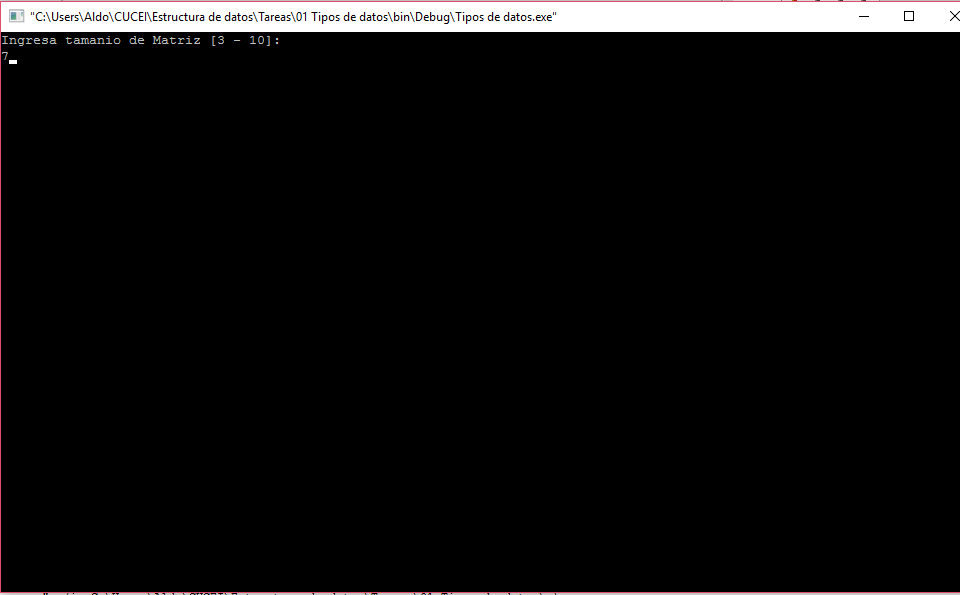
Con los bits obtenidos, podemos generar los rangos de valores de cada tipo de dato, con valores mínimos de cero y máximos de 2^(bits) -1 para los datos unisgned, y mínimos de – 2^(bits)/2 y máximos de 2^(bits)/2 – 1.

Una vez los rangos definidos, el segundo método de la clase imprime los valores en una tabla, con un diseño legible con ayuda de la función setw(n) para reservar un ancho de espacio para la impresión.

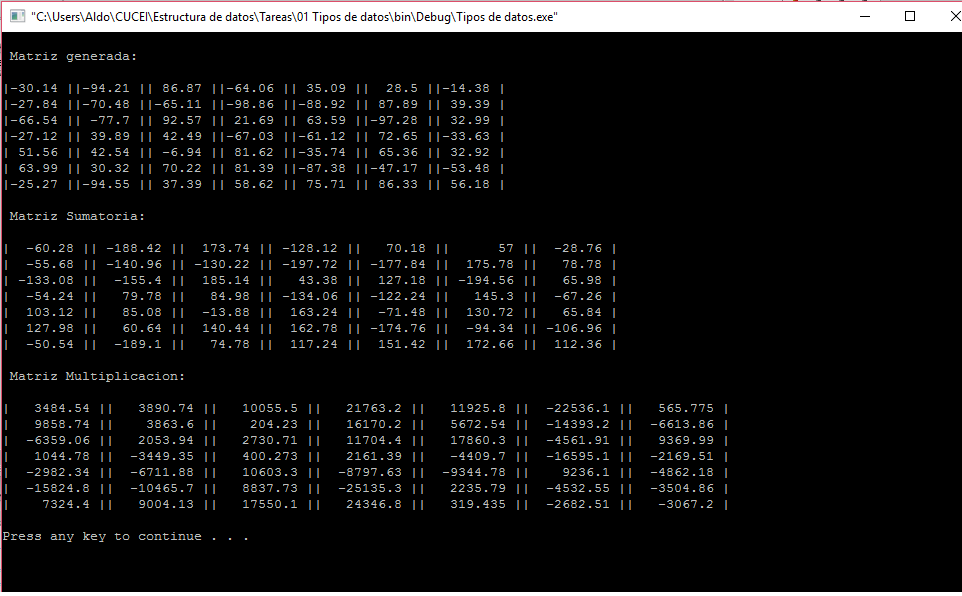


La **opción b)** solicitará un valor entero n para determinar el tamaño de matriz a utilizar, con un mínimo de 3 y un máximo de 10, luego rellenará dos matrices de tamaño n x n con valores aleatorios de tipo real comprendidos en el rango entre -100.00 y 100.00, mostrará el contenido de las matrices en un formato legible y comprensible con dos decimales, además del resultado de la multiplicación de dichas matrices, así como el resultado de la suma de las mismas.

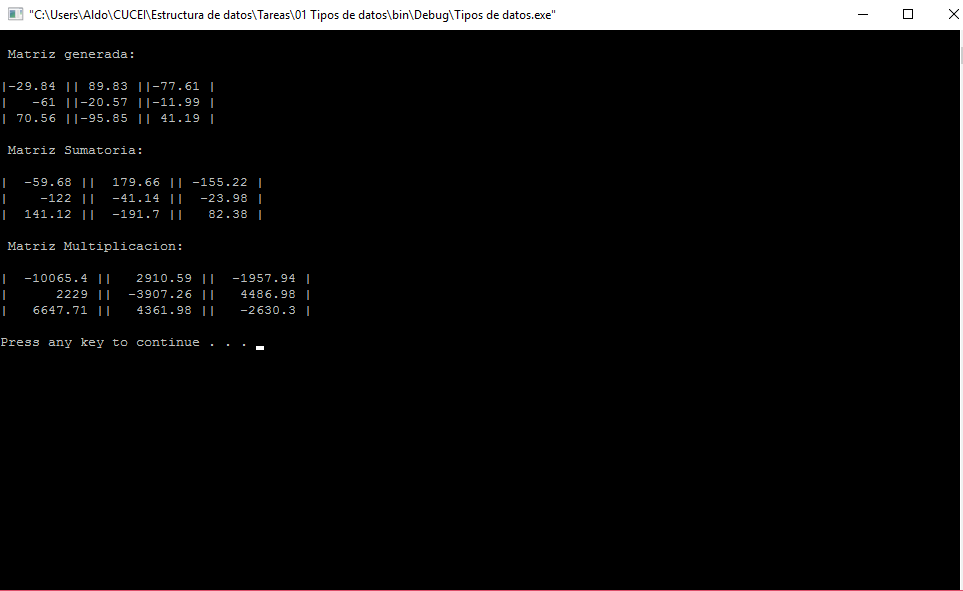
También definida en una clase, la matriz tendrá un atributo el cual será proporcionado por el usuario y definirá el tamaño de la matriz. Los métodos que la integran son 3, un getter y setter para el valor n, y una función que crea las matrices.

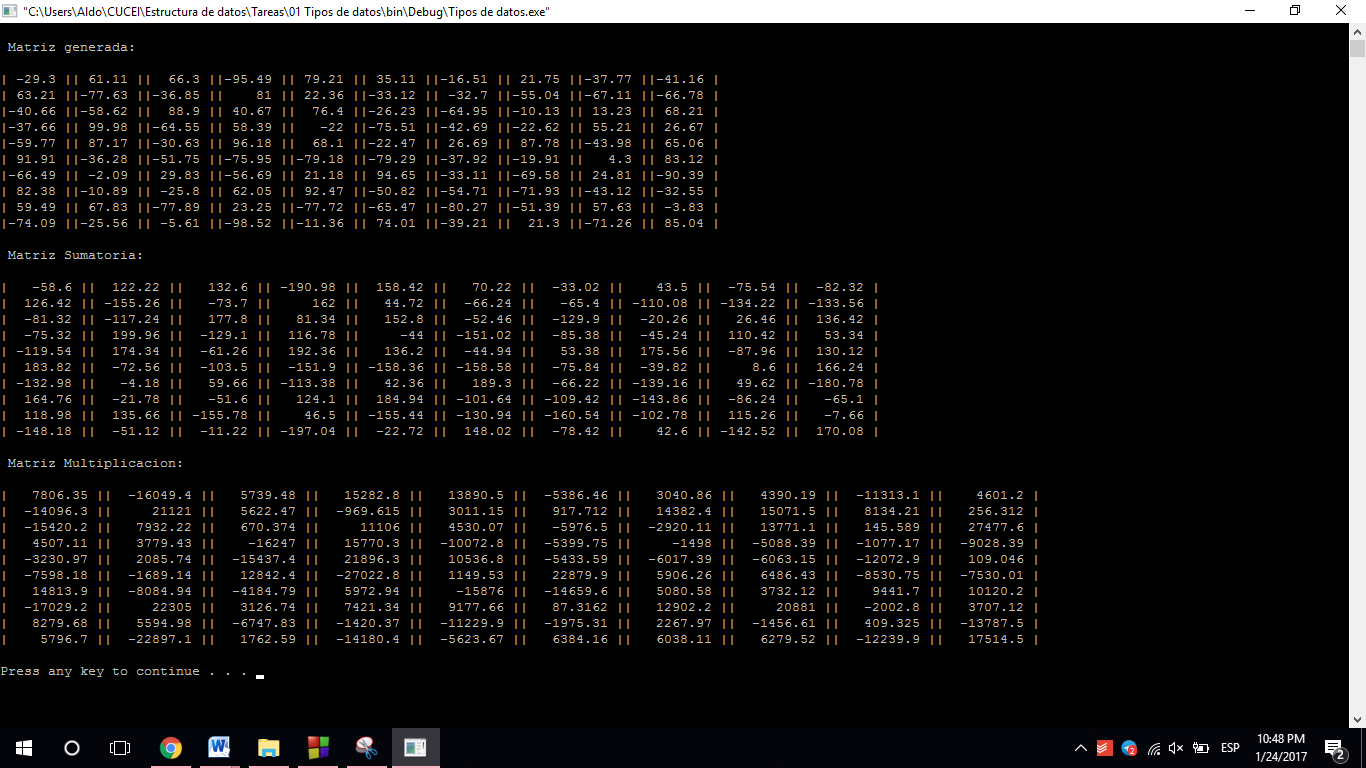


Dentro de la función **setter**, establecemos el límite con una condicional, en donde solo se aceptará un valor dentro del rango 3-10, y todo otro error no sea tomado como válido.

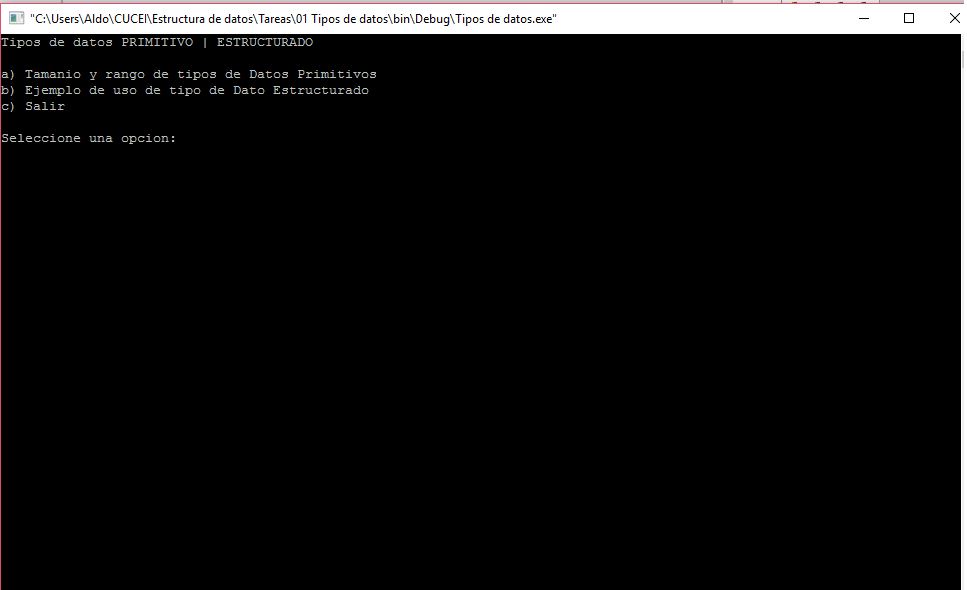


En la **función creadora de matrices**, definimos las matrices y elementos de apoyo para el método. La generación de matrices es con ciclos for, el cual ingresa un valor aleatorio que esta previamente tratado para poder generar número negativos, y limitar a dos decimales el número flotante, después es ingresado por índices en el lugar correspondiente. Las demás matrices son generadas con algoritmos matemáticos, para poder hacer la multiplicación correctamente.





Al **final** en el main, diseñamos el menú cíclico el cual obtiene una variable del tipo carácter, para llevar a cabo las opciones. Esta variable debe estar entre a, b y c (minúsculas y mayúsculas). El menú cicla indefinidamente ya que está dentro de un ciclo while que define la condicional como la opción designada para salir del programa.



**Códigos**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* Program Principal datos primitivos y estructurados**

**Aldo Vargas**

**213495653**

**file: main.cpp\*/**

**#include <iostream>**

**#include "dataTypes.h"**

**#include "matrix.h"**

**using namespace std;**

**int main(){**

**char menuOpt = 's';**

**DataTypes dty;**

**Matrix mtx;**

**while(menuOpt){**

**cout<< "Tipos de datos PRIMITIVO | ESTRUCTURADO " <<endl<<endl;**

**cout<< "a) Tamanio y rango de tipos de Datos Primitivos" <<endl;**

**cout<< "b) Ejemplo de uso de tipo de Dato Estructurado" <<endl;**

**cout<< "c) Salir" <<endl<<endl ;**

**cout<< "Seleccione una opcion: "; cin >> menuOpt;**

**system("cls");**

**if(menuOpt == 'a' || menuOpt == 'A'){**

**cout<< "Tamanio y rango de tipos de Datos Primitivos" <<endl<<endl;**

**for(int i=0; i<8; i++){**

**dty.setValues(i);**

**dty.getValues(i);**

**}**

**system("pause");**

**system("cls");**

**} else if(menuOpt == 'b' || menuOpt == 'B'){**

**cout<< "Ejemplo de uso de tipo de Dato Estructurado" <<endl<<endl;**

**mtx.setN();**

**mtx.createMatrix(mtx.getN());**

**system("pause");**

**system("cls");**

**} else if(menuOpt == 'c' || menuOpt == 'C'){**

**cout << "Saliendo del programa..." <<endl;**

**return 0;**

**}**

**else{**

**cout << "Opcion incorrecta" <<endl;**

**system("pause");**

**system("cls");**

**}**

**}**

**return 0;**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Matrix.h\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* Header Clase Matriz**

**Aldo Vargas**

**213495653**

**file: matrix.h\*/**

**#ifndef MATRIX\_H\_INCLUDED**

**#define MATRIX\_H\_INCLUDED**

**class Matrix{**

**private:**

**int n;**

**public:**

**/\*Imprime las matrices\*/**

**void createMatrix(const int&);**

**/\*Establece n\*/**

**void setN();**

**int getN();**

**};**

**#endif // MATRIX\_H\_INCLUDED**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Matrix.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* Clase Matriz**

**Aldo Vargas**

**213495653**

**file: matrix.cpp\*/**

**#include "matrix.h"**

**#include <iostream>**

**#include <stdlib.h>**

**#include<time.h>**

**#include <iomanip>**

**using namespace std;**

**void Matrix::setN(){**

**system("cls");**

**int x = 0;**

**while(x == 0){**

**cout<<"Ingresa tamanio de Matriz [3 - 10]: "<<endl;**

**cin >> x;**

**if(x>=3 && x<=10){**

**n = x;**

**} else{**

**cout<<"Error de limites"<<endl;**

**x = 0;**

**system("pause");**

**system("cls");**

**}**

**}**

**}**

**int Matrix::getN(){**

**return n;**

**}**

**void Matrix::createMatrix(const int& n){**

**/\*Declaracion de matrices\*/**

**float origMatrix[n][n];**

**float sumaMatrix[n][n];**

**float multMatrix[n][n];**

**/\*Declaracion para aleatorio flotante de -100.00 a 100.00\*/**

**float random;**

**int \_2decimals;**

**srand(time(NULL));**

**system("cls");**

**/\*Generacion de matrices\*/**

**for(int x=0; x<n; x++){**

**for(int y=0; y<n; y++){**

**/\*Generacion aleatorio entre -100.00 hasta 100.00\*/**

**random = ((static\_cast <float> (rand())) /( static\_cast <float> (RAND\_MAX/(200.00))))-100.00;**

**\_2decimals = (random\*100);**

**random = (\_2decimals\*1.0)/100;**

**origMatrix[x][y] = random;**

**sumaMatrix[x][y] = (origMatrix[x][y] + origMatrix[x][y]);**

**}**

**}**

**for (int i = 0; i < n; i++){**

**for (int j = 0; j < n; j++){**

**multMatrix[i][j] = 0;**

**for (int k = 0; k < n; k++){**

**multMatrix[i][j]=multMatrix[i][j]+(origMatrix[i][k]\*origMatrix[k][j]);**

**}**

**}**

**}**

**/\*Impresion de matrices\*/**

**cout<<endl<<" Matriz generada: "<<endl<<endl;**

**for(int x=0; x<n; x++){**

**for(int y=0; y<n; y++){**

**cout<<"|"<<setw(6)<<origMatrix[x][y]<<" |";**

**if(y==n-1){**

**cout<<endl;**

**}**

**}**

**}**

**cout<<endl<<" Matriz Sumatoria: "<<endl<<endl;**

**for(int x=0; x<n; x++){**

**for(int y=0; y<n; y++){**

**cout<<"|"<<setw(8)<<sumaMatrix[x][y]<<" |";**

**if(y==n-1){**

**cout<<" "<<endl;**

**}**

**}**

**}**

**cout<<endl<<" Matriz Multiplicacion: "<<endl<<endl;**

**for(int x=0; x<n; x++){**

**for(int y=0; y<n; y++){**

**cout<<"|"<<setw(10)<<multMatrix[x][y]<<" |";**

**if(y==n-1){**

**cout<<endl;**

**}**

**}**

**}**

**cout<<endl;**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*dataTypes.h\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* Header Tipos de Datos**

**Aldo Vargas**

**213495653**

**file: dataTypes.h\*/**

**#ifndef DATATYPES\_H\_INCLUDED**

**#define DATATYPES\_H\_INCLUDED**

**#include <string>**

**class DataTypes{**

**private:**

**std::string dataType;**

**float minValue, maxValue;**

**int bits;**

**public:**

**/\*Imprime la tabla\*/**

**void getValues(const int&);**

**/\*Calcula la tabla\*/**

**void setValues(const int&);**

**};**

**#endif // DATATYPES\_H\_INCLUDED**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*dataTypes.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**/\* Clase Tipos de Datos**

**Aldo Vargas**

**213495653**

**file: dataTypes.cpp\*/**

**#include "dataTypes.h"**

**#include <iostream>**

**#include <iomanip>**

**#include <math.h>**

**using namespace std;**

**void DataTypes::getValues(const int& s){**

**/\*Imprime la tabla de tipos de datos\*/**

**if(s == 0){**

**cout<<setw(16)<<"Tipo de Dato"<<setw(7)<<"Bits"<<setw(14)<<"Valor Minimo"<<setw(14)<<"Valor Maximo"<<endl;**

**cout<<setw(16)<<dataType<<setw(7)<<bits<<setw(14)<<minValue<<setw(14)<<maxValue<<endl;**

**}**

**else{**

**cout<<setw(16)<<dataType<<setw(7)<<bits<<setw(14)<<minValue<<setw(14)<<maxValue<<endl;**

**}**

**}**

**void DataTypes::setValues(const int& s){**

**/\*Characters\*/**

**if(s == 0){**

**dataType = "Unsigned Char"; //Asigna el tipo de dato**

**bits = sizeof(unsigned char) \* 8; //Obtiene su tamaño en bits**

**maxValue = -1 + (pow(2,bits)); //Calculo de valor maximo**

**minValue = 0; //Dado que es unsigned el min es cero**

**}else if(s == 1){**

**dataType = "Signed Char";**

**bits = sizeof(signed char) \* 8;**

**maxValue = -1 + (pow(2,bits)/2); //El valor maximo es la mitad del rango positivo**

**minValue = 0 - (pow(2,bits)/2); //y la mitad negativa será el minimo**

**}**

**/\*Integers\*/**

**else if(s == 2){**

**dataType = "Unsigned Short";**

**bits = sizeof(unsigned short int) \* 8;**

**maxValue = -1 + (pow(2,bits));**

**minValue = 0;**

**}else if(s == 3){**

**dataType = "Signed Short";**

**bits = sizeof(signed short int) \* 8;**

**maxValue = -1 + (pow(2,bits)/2);**

**minValue = 0 - (pow(2,bits)/2);**

**}**

**/\*Longs\*/**

**else if(s == 4){**

**dataType = "Unsigned Long";**

**bits = sizeof(unsigned long) \* 8;**

**maxValue = -1 + (pow(2,bits));**

**minValue = 0;**

**}else if(s == 5){**

**dataType = "Signed Long";**

**bits = sizeof(signed long) \* 8;**

**maxValue = -1 + (pow(2,bits)/2);**

**minValue = 0 - (pow(2,bits)/2);**

**}**

**/\*Floats\*/**

**else if(s == 6){**

**dataType = "Float";**

**bits = sizeof(float) \* 8;**

**maxValue = -1 + (pow(2,bits));**

**minValue = 0;**

**}else if(s == 7){**

**dataType = "Double";**

**bits = sizeof(double) \* 8;**

**maxValue = -1 + (pow(2,bits));**

**minValue = 0;**

**}**

**}**