CECS2200-20-p1c6

Cambio de planes, el examen será a partir del martes que viene. En Blackboard, - Pueden hacerlo desde la casa. Desde el martes 14 hasta el jueves 16.
- Tienen 2 horas para terminarlo.
- Lo pueden pausar.
- El examen es aleatorio, cada usuario tendra diferentes preguntas.
- Les daré tres intentos para hacer el examen. Blackboard te pone la nota del ultimo intento. Yo cambio la regla y hago un override y les pondré la nota mayor.
- El examen es multichoice, con cuatro respuestas posibles y seleccionarán la mejor respuesta.
- El examen estará compuesto de 10 preguntas como máximo.
- El material del examen será:

S6: Lunes 6 enero	▪ *Variables (basic types, declaration, initialization, scope rules, casting), constants, literal, keywords* ▪ *Standard library* ▪ *Arithmetic operators (+, -, *, /, %), precedence, and association* ▪ *Arithmetic expressions* ▪ *Math Library*

La clase de hoy No entra en el primer examen!!

Clase de hoy: Variables. Veamos la definición:

Variables

A *variable* is a named storage location in the computer's memory for holding a piece of data.

Declaración de variables: ejemplo

```
1  // Example variable definition o declaration
2  double rate, pay, hours;
3  int h, l, w;
4  char a, b, c;
5  float x, y;
```

Inicialización: ejemplo.

```
// Example variable definition and INITIALIZATION
double rate=0, pay=-1, hours=0;

//También se puede inicializar una variable en otro statement (instrucción)
int precio;



precio =10;
```

Scope Rules:

2.14 Scope

CONCEPT: A variable's scope is the part of the program that has access to the variable.

Every variable has a *scope*. The scope of a variable is the part of the program where it may be used. The rules that define a variable's scope are complex, and we will just introduce the concept here. Later in the book we will cover this topic in more depth.

The first rule of scope is that a variable cannot be used in any part of the program before it is defined. Program 2-19 illustrates this.

Otra regla de scope es que la variable está disponible solamente en el mismo conjunto de llaves {}

```
2  #include <iostream>
3  using namespace std;
4
5  int main(){
6      int value = 100;
7      cout << value;
8
9      if(1){
10         int value2 = 200; //Esta variable solamente vive desde la linea 9
11         //hasta la linea 13
12
13     }//end if
14
15     return 0;
16 }// end main
```

Casting: Te permite cambiar el tipo de variable. Hay varias maneras de hacer esto. Ejemplo

3.4 Explicit Type Conversion

CONCEPT: Type casting allows you to explicitly perform data type conversion.

A *type cast expression* lets you manually promote or demote a value. The general format of a type cast expression is

```
static_cast<DataType>(Value)
```

where *Value* is a variable or literal value that you wish to convert and *DataType* is the data type you wish to convert it to. Here is an example of code that uses a type cast expression:

```
double number = 3.7;
int val;
val = static_cast<int>(number);
```

```
cout << (int) 2.6;          // Displays integer 2

intVal = (int)number;       // Assigns intVal the value of
                            // number, converted to an int

booksPerMonth =             // Converts a copy of the value
  (double)books / months;   // stored in books to a double
                            // before performing the division
                            // operation
```

Constantes: Son valores que no pueden cambiar a lo largo del programa:

Constants are data items whose values cannot change while the program is running.

```
const double INTEREST_RATE = 0.129;
```

When a named constant is defined it must be initialized with a value. It cannot be defined and then later assigned a value with an assignment statement.

```
const double INTEREST_RATE;      // illegal
INTEREST_RATE = 0.129;           // illegal
```

An added advantage of using named constants is that they make programs more self-documenting. If the named constant `INTEREST_RATE` has been correctly defined, the program statement

```
newAmount = balance * 0.129;
```

can be changed to read

```
newAmount = balance * INTEREST_RATE;
```

Otra manera de definir constantes es con define: ejemplo:

```
#define PI 3.14159                  // PI is "defined" to be 3.14159
```

Las constantes viven en su scope.
Sin embargo puedes definir lo que se llaman global constants, y se definen antes del main(). Su scope es global!! o sea, vivien el todos los scopes.
Ejemplo:

```
1  // This program calculates gross pay. It uses global constants.
2  #include <iostream>
3  #include <iomanip>
4  using namespace std;
5
6  // Global constants
7  const double PAY_RATE = 22.55;      // Hourly pay rate
8  const double BASE_HOURS = 40.0;     // Max non-overtime hours
9  const double OT_MULTIPLIER = 1.5;   // Overtime multiplier
10
```

Literal, o String Literal: Son letras dentro de ""

```
cout << "Programming is great fun!";
```

To put it simply, this line displays a message on the screen. You will read more about cout and the << operator later in this chapter. The message "Programming is great fun!" is printed without the quotation marks. In programming terms, the group of characters inside the quotation marks is called a *string literal*, a *string constant*, or simply a *string*.

Keywords:

auto	double ✓	int ✓	struct
break	else ✓	long ✓	switch
case	enum	register	typedef
char ✓	extern	return ✓	union
const ✓	float ✓	short ✓	unsigned ✓
continue	for	signed	void
default	goto	sizeof	volatile
do	if ✓	static	

written in lowercase, each have a special meaning in C++ and can only be used for their intended purposes. As you will see, the programmer is allowed to make up his or her own

names for certain things in a program. Key words, however, are reserved and cannot be used for anything other than their designated purposes. Part of learning a programming