Clase #05 de 27
Funciones II

Abril 26, Jueves

Agenda para esta clase

- Ejercicio: Doble(n)
- Función de Dos Variables & División Entera
  - Más de un parámetro en prototipo
  - Más de un argumento en invocación
  - Más de un parámetro en definicón
  - División cerrada
  - Conversión implícita de tipos
  - Promoción automática de tipos
- Comparación de Flotantes
  - Truncamiento por división entera
  - Representación no precisa de tipos flotantes
  - Comparación con tolerancia
  - Argumentos por defecto
- Operador Condicional y Funciones Partidas: El Operador Ternario
  - Semántica del operador ternario
  - Formateo de expresiones con operador ternario
  - Paréntesis inncesarios.

Ejercicio: Doble(n)

Especificar e Implementar Doble(n)

$$\text{double} : \mathbb{Z} \rightarrow \mathbb{Z} / \text{double}(n) = 2n$$

```
int Double(int);

assert( -14 == Double(-7) );
assert(   0 == Double(0) );
assert(  42 == Double(21) );

int Double(int n){return 2*n;}
```

Función de Dos Variables & División Entera

Promedio de dos números

Promedio de Dos Números

$$\text{avg}: \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{R} / \text{avg}(a, b) = \frac{a+b}{2}$$

```
double Avg(int,int);

assert( 3   == Avg( 2, 4) );
assert( 1   == Avg( 1, 1) );
assert( 0   == Avg( 0, 0) );
assert( 0   == Avg(-1, 1) );
assert( 1.5 == Avg( 1, 2) );

double Avg(int a, int b){
    return (a+b)/2.0;
}
```

- El dominio también puede denotarse como Z^2
- En los prototipos, los tipos de **parámetros** se separan con coma, no es necesario nombrarlos
- En la invocación, los **argumentos** también se separan con coma
- En la definición se deben nombrar los **parámetros**, son independientes de los nombres usados en el prototipo, si es que se usaron
- Cada **parámetro** debe tener su tipo especificado, no es válido escribir Avg(int a,b).
- La división es cerrada entre ints, por eso se divide por un double. La adición da un int, pero se convierte a un double implícitamente, se aplica **promoción de tipo automática**.

Intervalo

20 minutos

Comparación de Flotantes

Problema: Conversión de Fahrenheit a Celsius

$$celsius : \mathbb{R} \rightarrow \mathbb{R} / celsius(f) = \frac{5}{9}(f - 32)$$

```
double Celsius(double);
```

- Pero hay dos sub-problemas a solucionar antes de poder probar e implementar la función:
  - Valor del racioal cinco novenos versus la división entera en C++ de la expresión 5/9
  - Representación no precisa de los tipos flotantes
- Una solución al primer problema es realizar división entre flotantes.
- Para el segundo problema, debemos incorporar la comparación con tolerancia.

Función Están Cerca (AreNear)

```
bool AreNear(double, double, double);
bool AreNear(double, double, double = 0.001);
bool AreNear(double x, double y, double tolerance = 0.001);

assert( AreNear(1.0, 0.999  ));
assert( AreNear(1.0, 0.9, .1));
assert( AreNear(    1.0/3.0, 0.333));
assert( not AreNear(1.0/3.0, 0.33 ));


bool AreNear(double a, double b, double delta){
     return (a-delta) <= b and b <= (a+delta);
}
```

AreNear y Celsius

```
#include <cassert>
#include <iostream>

double Celsius(double);
bool AreNear(double, double, double = 0.001);

int main(){
        assert( 0  == Celsius(32) );
        std::cout << Celsius(64); // 17.7778
        assert( 17.7778 != Celsius(64) );
        assert( 17.777  <  Celsius(64) );
        assert( 17.778  >  Celsius(64) );
        assert( 17.777  <  Celsius(64) and Celsius(64) < 17.778);
        assert( AreNear( 42,  42, 0));
        // 1 == 10*(1/10)
        assert( not AreNear( 1.0, 0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1, 0));
        assert( AreNear(     1.0, 0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1   ));
        assert( AreNear(1.0, 0.999  ));
        assert( AreNear(1.0, 0.9, .1));
        assert( AreNear(    1.0/3.0, 0.333));
        assert( not AreNear(1.0/3.0, 0.33 ));
        assert( AreNear(-35.5556,  Celsius(-32)));
        assert( AreNear(-17.7778,  Celsius(  0)));
        assert( AreNear( 17.7778,  Celsius( 64)));
        assert( AreNear( 17.77777777778,  Celsius( 64), 0.00000000001 ));
}

double Celsius(double f){return 5.0/9.0*(f-32);}

bool AreNear(double a, double b, double delta){
        return (a-delta) <= b and b <= (a+delta);
}
```

Operador Condicional y Funciones Partidas

El Operador Ternario

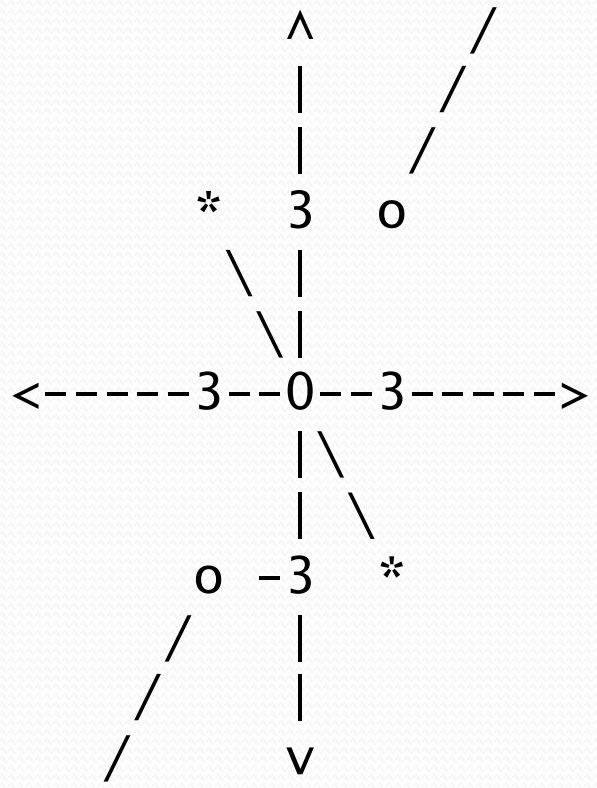
Absoluto

$$\text{abs}: \mathbb{Z} \rightarrow \mathbb{Z} / \text{abs}(x) = \begin{cases} -x & x < 0 \\ x & \text{e.o.c.} \end{cases}$$

```
int Abs(int);

assert( 42 == Abs(-42) );
assert(  0 == Abs(  0) );
assert( 42 == Abs( 42) );

int Abs(int x){return x<0 ? -x : x ;}


// Alternativamente, se puede formatear como
int Abs(int x){ return
     x<0 ? -x :
            x ;}
```

Ejercicio

- Especificar e implementar la función f3 definida por el siguiente gráfico



Términos de la clase #05

Definir cada término con la bibliografía

- Función de Dos Variables & División Entera
  - Más de un parámetro en prototipo
  - Más de un argumento en invocación
  - Más de un parámetro en definicón
  - División cerrada
  - Conversión implícita de tipos
  - Promoción automática de tipos
- Comparación de Flotantes
  - Truncamiento por división entera
  - Representación no precisa de tipos flotantes
  - Comparación con tolerancia
  - Argumentos por defecto
- Operador Condicional y Funciones Partidas: El Operador Ternario
  - Semántica del operador ternario
  - Formateo de expresiones con operador ternario
  - Paréntesis y el operador ternario.

Tareas para la próxima clase

1. Trabajo opcional: Probar, Declarar y Definir las funciónes Celsius y AreNear.
2. Trabajo opcional: Probar, Declarar y Definir la función f3.

¿Consultas?

Fin de la clase