C++
Biblioteca Estándar de Templates
Standar Template Library

Agustín J. González

ELO320

Características de C++

- El propósito de este material es rápidamente revisar todas características de C++ que ustedes probablemente han encontrado
- Si ustedes han aprendido otro lenguaje, podrán ubicarse rápidamente
- Referencia: Timothy Budd, “Data Structures in C++, using the Standard Template Library”, Addison Wesley 1998. Esta en nuestra biblioteca.

Comentarios y Constantes

- Hay dos tipos de comentarios
  // desde doble slashes hasta el final de la línea
  /*
  Comentarios que se extienden
  varias líneas
  */
- Los comentarios deberían usarse intensamente por documentación.
- **Constantes:**
- hay varios tipos de constantes
  integers -1,12,-37
  enteros octales 014
  enteros hexadecimales 0xFF 0xC
  punto flotante 3.14158, 2.7e14
  caracteres ‘a’ ‘\n’
  string “abc”
- sufijos pueden ser aplicados a constantes
  u por unsigned
  l por long

Variables, Tipos, Valores y Declaraciones

- Una variable es una localización con nombre que almacena un valor de cierto tipo.
- Variables son creadas usando una sentencia de declaración, la cual describe también el tipo asociado.
  int a,b,c; // declara tres enteros
- La declaración puede ser combinada con inicialización:
  double pi = 3.1415926;
- **Tipos Fundamentales de Datos**
- Enteros int
- punto flotante: double, float
- Caracter: char
- Modificadores que pueden ser usados con tipos fundamentales
  signed,, unsigned
  short, long (posiblemente) mas pequeño o mas grande que estándar.

Mas tipos de Datos

- Variables Booleanos (bool)son verdaderas o falsas (true/false)
- Valores enumerados son definidos indicando el rango en forma explícita

  enum month{January, February, March, April, May, June, July,August, September, October,November, December};
  months workingMonth, vacationMonth;
  months summerMonth = August;
- **Variables y Asignaciones**

```
double f, c;          // Temperatura Fahrenheit y Celsius
    c = 43;
    f = (c * 9.0) / 5.0 + 32;
```

- Operadores binarios pueden ser combinados con asignaciones

  i += 5; // equivale a i = i+5;
- Otra anotación compacta es:

  i++

Operadores

Unary Operators	
increment, decrement	i++, ++i, i−−, −−i
negation	− i
bit-wise inverse	∼ i
Arithmetic Operations	
addition, subtraction	a+b a−b
multiplication, division	a*b a/b
remainder after division	a % b
Shift Operations (also stream I/O)	
left shift (also stream output)	a << b
right shift (also stream input)	a >> b
Relational Operations	
less than, less than or equal	< <=
equal, not equal	== !=
greater than, greater than or equal	> >=
Logical Operations	
and	x && y
or	x || y
logical negation	! i
Miscellaneous Operations	
function call	f(a,b,c)
conditional expression	c ? a : b

Stream de I/O

- El operador de corrimiento izquierda y derecha tienen distinto significado cuando son usados con valores de stream.
- El stream mas común es el asociado con la consola de entrada y salida.

```
cout << "the Fahrenheit equivalent of " << c <<
        " is " << f << "\n";
cin >> c; // get a new value of c
cout << "the Fahrenheit equivalent of " << c <<
        " is " << f << "\n";
```

- El Operador >> trabaja or efecto lateral, cambiando la expresión del lado derecho. El resultado puede ser convertido a boolean para probar si la estrada fue exitosa.

```
int sum = 0;
int value;
while (cin >> value) {
        sum += value;
        }
cout << "sum is " << sum << "\n";
```

Punteros

- Los punteros pueden ser usados de las siguientes formas:
- Punteros pueden ser usados con subíndices (tiene sentido si apuntan a arreglos, pero esto no es chequeado)
- Pueden ser des-referenciados, usando el operador *
- Pueden combinar des-referencia con accesos a campos, usando operador ->
- Pueden usar sumas, p+i es la dirección de p[i]

Sentencias condicionales

```
month aMonth;
        ...
if ((aMonth >= June) && (aMonth <= August))
        isSummer = true;
else  // es opcional
        isSummer = false;
```

Sentencia Switch

```
switch (aMonth) {
        case January:
                highTemp = 20;
                lowTemp = 0;
                break;
        case February:
                highTemp = 30;
                lowTemp = 10;
                break;
        ...
        case July:
                highTemp = 120;
                lowTemp = 50;
                break;
        default:
                highTemp = 60;
                lowTemp = 20;
};
```

Bucles de repetición

WHILE

```
c = 0;
while (c <= 100) {
        cout << "Celsius " << c << " is Fahrenheit " <<
                ((9.0 * c) / 5.0 + 32) << "\n";
        c += 10;
        }
```

FOR

```
for (c = 0; c <= 100; c += 10) {
        cout << "Celsius " << c << " is Fahrenheit " <<
                ((9.0 * c) / 5.0 + 32) << "\n";
        }

for (int i = 0; i < 12; i++) {
        cout << "i: " << i << " i squared " << i*i << "\n";
        }
```

Arreglos

- Es una colección de tamaño fijo de valores del mismo tipo.

```
        // declare an array of twelve integer values
int Temperatures[12];
        // now assign all values
Temperatures[0] = 0;
Temperatures[1] = 10;
Temperatures[2] = Temperatures[1] + 15;
...
```

- Pueden ser inicializados

```
string MonthNames[12] = {"January", "February",
        "March", "April", "may", "June",
        "July", "August", "September", "October",
        "November", "December" };
```

- Arreglos Multidimencionales

```
double matrix[10][20];
matrix [i][j] = matrix [i-1][j+1] + 1;
```

Arreglos y Punteros

- Hay una relación muy cercana entre arreglos y punteros
- El nombre del arreglo es tratado como un puntero.
- Punteros pueden usar subíndices, como si fueran arreglos(aun si no lo son)
- MonthNames+3 es legal, es la dirección de MonthNames[3].
- **Arreglos como Argumentos**
- Cuando son usados como argumentos no necesitamos especificar el tamaño.

```
int arraySum (int values[ ], int n)
    // compute sum of array values[0] .. values[n-1]
{
    int result = 0;
    for (int i = 0; i < n; i++) {
            result += values[i]
            }
    return result;
}
```

Estructuras

- Es una colección de campos que no requieren tener el mismo tipo.

```
struct person {
        string name;
        int age;
        enum {male, female} sex;
        };
```

- El acceso es via el operador .

```
person employee;
employee.name = "sam smith";
employee.age++;
if (employee.sex == male)
    ...
```

Funciones

- Una función encapsula un conjunto de acciones.

```
int Fahrenheit(int cTemp)
{
        return (cTemp * 9.0) / 5.0 + 32;
}
```

- Partes:
  Encabezado con tipo retornado, nombre y argumentos.
  Cuerpo con sentencias a ejecutar.
- Valor retornado puede ser void (ningún valor)
- Una función ***prototipo*** es una ***declaración*** pero sin ***definición***.

```
        // prototype for Fahrenheit -- definition occurs later
int Fahrenheit (int);
```

Variables Locales

- Variables al interior de la función existen solo cuando la función es ejecutada, desaparecen cuando la función retorna. La excepción a esta regla son las variables estáticas.
- Si suponemos que la función A llama a B y esta llama a C, podemos imaginar en el stack del proceso usuario:
  Variables locales de C
  Variables locales de B
  Variables locales de A
- Si B fuera recursiva y llamara a si misma una vez antes de llamar a C, tenemos:

  Variables locales de C
  Variables locales de B
  Variables locales de B
  Variables locales de A

El evento Main

- Todo programa debe incluir un procedimiento main, el cual es el punto de partida de la ejecución.

```
# include <iostream>
void main() {
    // program to write table of squares
    cout << "Table of Squares\n";

    for (int i = 0; i < 12; i++) {
        cout << "i: " << i << " i squared " << i*i << "\n";
        }
}
```

Archivos includes

- Muchas estructuras de datos requieren que uno defina un archivo de inclusión antes que podamos usarlas.

Propósito	Nombre
stream input/output	iostream
funciones matemáticas	math.h
números complejos	complex
valores booleanos	bool.h
algoritmos genéricos	algorithm
Abstracción vector	vector
Abstracción lista	list
Abstracción conjunto	set
Abstracción mapa	map