Apuntes PRO1 Ing. Informática UPC-FIB 2020-2021

Abstract: Apuntes sobre el temario impartido en la asignatura de Programación 1 (PRO1) del grado de Ingeniería Informática de la Universitat Politècnica de Catalunya - Facultat d'Informàtica de Barcelona (UPC-FIB).

Initial Release: November 24th, 2020. (1.0.0)

Current Version: 1.0.0 //Initial Release, added unit 1 to 8.3.1

Current Developer(s): Diego Carballido (Github, Instagram, Mail Address)

> Este documento ha sido redactado con el fin de ayudar a más estudiantes de la asignatura de Programación I, o simplemente a personas interesadas en aprender los conceptos básicos de la programación. Por lo tanto, este documento es open-source, y todos los usuarios son libres de copiarlo y modificar su contenido, si lo creen conveniente, para añadir más información.

Apuntes PRO1 Ing. Informática UPC-FIB 2020-2021

1.- Introducción del documento

Se crea este documento **Markdown** con el fin de tener la mayor cantidad de apuntes e información sobre la asignatura de Programación 1 (**PRO1**) del grado de Ingeniería Informática de la Universitat Politècnica de Catalunya - Facultat d'Informàtica de Barcelona (UPC-FIB). Dicha asignatura, utiliza fundamentalmente el lenguaje de programación **C++**, utilizando los ficheros con extensión **.cc**.

A lo largo de este documento, se detallarán los conceptos impartidos durante la asignatura, exponiendo diversos ejemplos, y casos prácticos, incluyendo fragmentos de código C++, pues por este motivo se redacta este documento en formato **.md**, ya que es más sencillo para incluir fragmentos de código del lenguaje de programación que se desee.

2.- Cómo poder visualizar este documento correctamente?

Este documento con formato Markdown (.md) estará colgado en un repositorio dedicado expresamente para esta asignatura. Dicho repositorio se encontrará en **Github**, en el perfil de Diego Carballido; la plataforma de Github, por si misma, es capaz de interpretar los documentos .md; pero si el usuario lo considera, dispone de distintos programas para la visualización y edición de estos documentos, algunos de ellos son:

- **Typora**: Este es el editor de Markdown más recomendado, siendo este gratuito, con una gran cantidad de elementos personalizables, y unos temas de visualización identicos a los que se utilizan en plataformas como Github o Gitlab. Además el estilo de este editor te permite estar 100% concentrado en el documento. Cabe mencionar que este editor está disponible para macOS, Windows y Linux. Puedes encontrarlo aquí.
- **WriteMonkey**: Otra de las alternativas como editor de Markdown es WriteMonkey, con una UX/UI similar al programa anterior, este tambien está recomendado para la edición y visualización de documentos Markdown en local. Por desgracia este software solo está disponible para ordenadores con SO de Windows. Puedes encontrarlo aquí.

3.- Editores de código para programar en local

Para programar los ejercicios y/o programas de C++ de la asignatura, es recomendable utilizar un editor de código instalado en el equipo que se use habitualmente. El desarrollador utiliza de forma recurrente el editor de código **Sublime Text 3**, compatible con los SO de macOS, Windows y Linux, siendo este gratuito, altamente personalizable, y con identificación por colores del código, en función del lenguaje de programación en el que se esté trabajando, más de 30 distintos. Puedes encontrar más información aquí.

4.- Compilación de código .cc

Una vez desarrollado todo el código necesario para cumplir con los requerimientos solicitados, es necesario testear el código para ver si cumple con las expectativas. Para realizar esta acción es necesario compilar el código. Normalmente se utiliza el compilador de GNU para C++, conocido como **g++**, aunque en el momento de realizar la entrega en el **Jutge**, o durante el examen, se utiliza el compilador **p1++**, el cual es más restrictivo que el anterior por motivos de evaluación de conocimientos.

Para compilar el código deseado, primeramente debemos guardar el fichero con la extensión .cc, una vez guardado, desde el terminal del equipo, debemos dirigirnos al directorio donde se encuentra almacenado el fichero guardado. Una vez localizados en la misma dirección que el fichero ejecutamos el siguiente comando:

```
g++ nombre_fichero.cc -o nombre_fichero.exe
```

Una vez hecho esto, si el programa ha sido correctamente desarrollado, es decir, sin ningún tipo de error lógico o sintáctico propio del lenguaje de programación, será encapsulado en un fichero ejecutable (**Atención**: que el fichero se compile correctamente no significa que cumpla con el propósito requerido por el enunciado del ejercicio.). Una vez se ha creado el fichero .exe, se ejecuta mediante:

```
./nombre_fichero.exe
```

5.- Datatypes de C++

Aunque prácticamente todos los datatypes de C++ son comunes para todos los lenguajes de programación, vamos a hacer un breve resumen de los distintos tipos de variables disponibles y como se declaran:

- **String**: También conocido como cadena de caracteres, este es el tipo de variable dónde podremos almacenar conjuntos de caracteres, principalmente los que podemos introducir mediante el teclado, aunque mas adelante, veremos que también podemos almacenar otro tipo de caracteres.

  ```
  string test_string = "Hello World";
  ```

- **Char**: A diferencia del anterior, este tipo de variable solo puede almacenar un solo carácter, resultando en error de compilación si intentamos asignarle un valor de tipo String. Se puede observar que para asignar un valor de una variable tipo Char, solo encomillamos el valor con comillas simples (' '), en cambio, en tipo String utilizamos comillas dobles (" ").

  ```
  char test_char = 'H';
  ```

- **Int**: Este tipo de variable permite guardar valores numéricos, la diferencia fundamental es que Int guarda números con los que posteriormente se puede operar matemáticamente, modificando el propio valor almacenado en la variable, en cambio, este mismo valor, almacenado en una variable String o Char, no es utilizable mas que para mostrarlo por pantalla. Para la asignación de variables de tipo numéricas, no es necesario el uso de comillas.

  ```
  int test_int = 1234;
  ```

- **Double**: Este tipo de variable tambien permite almacenar valores numéricos, pero a diferencia del tipo Int, este permite guardar valores numéricos decimales. Cabe mencionar que estos valores decimales se separan mediante '.', es decir, notación británica.

```
double test_double = 1234.5678;
```

- **Boolean**: Este datatype solo puede almacenar uno de los dos únicos valores disponibles, que son el `true` y el `false`, cuya función principal es la de establecer estados de tipo interruptor, es decir, con dos únicos estados. Este tipo de variables se utilizan mayoritariamente para permitir o no el paso por ciertas partes del código, en función de un estado determinado previamente.

  ```
  bool pass = true;
  ```

A pesar de que existen más tipos de variable, como por ejemplo el tipo Float, para la asignatura de PRO1 solo necesitamos entender estos 4 tipos de variable.

Como hemos visto en los ejemplos, esta es la forma de inicializar/declarar variables y asignarle un valor determinado, pero hay otras formas de hacerlo, por ejemplo si queremos declarar varias variables del mismo tipo:

```
int num1, num2, num3;
```

O bien si queremos asignarle un valor a alguno de estos:

```
int num1, num2, num3 = 4;
double num4, num5, num6 = 2.34;
string str1, str2 = "Hello World";
```

6.- Instrucciones Input/Output

6.1.- Instrucción `cin` para input

Para poder recoger los valores que queramos asignar de forma manual a las variables, debemos incluir una instrucción que nos permita escribir en el terminal cuando ejecutemos el programa:

```
cin >> test;
```

Con esta instrucción, estamos permitiendo escribir en el terminal el valor que queramos asignarle a la variable test, atención, debemos asegurarnos que el valor que introduzcamos por el terminal para asignar en la variable, se corresponda con el tipo de variable de esta. Por tanto, resultará en error si ingresamos caracteres como 'ABCD...' si la variable está declarada como tipo Int o Double.

Si queremos asignar valores a varias variables seguidas, escribiremos lo siguiente:

```
cin >> test1 >> test2 >> test3 >> testN;
```

Aquí podemos asignar valores a tantas variables como hayamos declarado anteriormente.

6.2.- Instrucción `cout` para output

Para permitir que el programa pueda escribir por pantalla utilizamos el siguiente comando:

```
cout << "Hello World!" << endl;
```

Con esta linea conseguimos imprimir por el terminal la frase "Hello World!", y seguidamente realizamos un salto de linea con el comando `endl` (End Line).

En el primer ejemplo hemos impreso un string directamente redactado, pero también podemos imprimir el valor asignado a cualquier variable declarada en el programa, por ejemplo:

```
string test = "Hello World!";
cout << test << endl;

//OUTPUT>> Hello World!
```

Al igual que con el `cin`, con `cout` también podemos encadenar valores para imprimir:

```
string test = "Hello World!";
cout << test << ' ' << "My name is Paco!" << ' ' << "How R U ?" endl;

//OUTPUT>> Hello World! My name is Paco! How R U ?
```

7.- Estructúra básica de un programa en C++

Para estructurar un programa de C++ de forma correcta, debemos tener en cuenta varios aspectos importantes. En primer lugar, en la parte superior del fichero, debemos informar de las librerias que estamos utilizando; siempre utilizaremos la librería cuyo propósito es incluir instrucciones básicas como `cin`, `cout`, `endl`, etc. Además de esta librería, podremos utilizar diversas de estas según el propósito del programa, por ejemplo, la librería , utilizada para llamar a la constante `M_PI` (número π) o las funciones de `sin()`, `cos()`, etc. También debemos declarar el formato de escritura que utilizamos; siempre utilizaremos el `std`.

```
#include<iostream>  //esto si es obligatorio
#include<cmath> //esto no es 100% obligatorio, solo si es necesario para el programa
using namespace std;  //esto si es obligatorio
```

Una vez hechas las inclusiones mencionadas, debemos crear una función `main` donde escribiremos todo el código del programa. Así pues, un ejemplo básico con todo lo aprendido sería:

```
#include<iostream>
using namespace std;

int main(){
  string test_str = "Hello World";
  int age = 20;

  cout << string << ' ' << "My name is Paco! And I am " << age << " years old!" <<
endl;
}

//OUTPUT>> Hello World! My name is Paco! And I am 20 years old!
```

8.- Instrucciones de condición e iteración

8.1.- Instrucción condicional `if`

Una de las instrucciones más utilizadas es la del condicional `if`, esta instrucción nos permite realizar una comparación entre dos argumentos, un ejemplo es el de la comparación entre los valores asignados de dos variables, o el de la comparación de un valor fijo con el del valor asignado a una variable.

Dentro de la definición de la condición del bucle, utilizaremos un conjunto de operadores comparación entre los dos argumentos, para determinar si una condición es correcta (se cumple) o no. Los operadores de comparación son:

- Menor que: `<`
- Mayor que: `>`
- Menor o igual: `<=`
- Mayor o igual: `>=`
- Igual (en cuanto al contenido): `==`
- Distinto: `!=`

```
int a = 5;
if(a > 3){
  cout << "mayor" << endl; //Imprime esta linea, pues cumple la condición.
}
```

En este caso, hemos comparado si el valor de la variable es mayor que un valor fijado (3), al ser esta condición cierta, el programa ejecuta el código dentro del condicional. En caso de que la condición fuera falsa, el programa obviaría el contenido del condicional y seguiría ejecutando el resto del programa, si existiese.

Otro de los componentes del condicional `if` es el `else`, este componente permite dar una alternativa de código dentro del condicional, si este no cumple la condición requerida. Un ejemplo es:

```
int a = 10;
if(a < 5){
  cout << "Hello" << endl; //No imprime esta linea, pues no cumple la condición.
}else{
  cout << "Bye" << endl;  //Al no entrar en el condicional, se imprime esta
condición.
}
```

En los condicionales `if` es posible estructurar condiciones más complejas, a partir de varias otras condiciones, conjugándolas entre ellas para crear una sola. Esta situación se consigue con los operadores lógicos `and`, `or` y `not` (normalmente se representan como `&&`, `||` y `!` respectivamente, pero en la asignatura de PRO1 no se utiliza la notación simbólica, sinó la escrita).

- And: `and` // `&&` escrito entre dos condiciones, devuelve el booleano `true` siempre y cuando se cumplan las dos condiciones.
- Or: `or` // `||` escrito entre dos condiciones, devuelve el booleano `true` siempre y cuando se cumpla una de las dos condiciones.
- Not: `not` // `!` Cambia el estado de la condición, si esta es originalmente `true` con este operador pasará a ser `false`.

```
int a = 10;
if(a > 5 and a != 11){
  cout << "Hello" << endl; //Imprime esta linea, pues cumple la condición.
}else{
  cout << "Bye" << endl;  //Al entrar en el condicional, no se imprime esta
condición.
}
```

```
int a = 10;
if(a > 5 or a == 10){
  cout << "Hello" << endl; //Imprime esta linea, pues cumple la condición.
}else{
  cout << "Bye" << endl;  //Al entrar en el condicional, no se imprime esta
condición.
}
```

8.2.- Instrucción iterativa definida: Bucle `for`

Otra estructura básica de la programación son los bucles, una repetición de un fragmento de código, en este caso, los bucles definidos, son bucles con un fin definido, es decir, que el código a ejecutar de forma repetitiva, tiene un fin ya marcado. En el caso de los bucles definidos, utilizamos la instrucción `for`, cuya sintaxis es la siguiente:

```
for (int i = 0; i < valor_de_fin; ++i){
  /* fragmento de código que queremos repetir */
}
```

Los argumentos del bucle `for` se componen de una variable que funciona como contador (normalmente se le llama 'i', 'j' o 'k' aunque puede tener el nombre que más convenga para el programa), a la que le determinamos un valor inicial.

El segundo argumento es el condicional de paso o de continuidad, este siempre compara el valor actual de la variable de contador, con el valor que se determina para finalizar la iteración; esta comparación se constituye mediante operadores lógicos; dicho argumento puede ser un valor fijado o una variable.

Finalmente el último argumento es el sumador +1 de la variable de contador, cuya función es aumentar en 1 el valor de la variable de contador, cada vez que se comienza una iteración.

Un ejemplo de bucle definido, es el siguiente, que imprime los números del 0 al 9, utilizando el valor de la variable de contador, pues contiene precisamente los valores que queremos:

```
for (int i = 0; i < 10; ++i){
  cout << i;  //En este caso, utilizamos la variable de contador para imprimir los
números
}

//OUTPUT>> 0123456789
```

8.3.- Instrucción iterativa indefinida: Bucle `while`

El otro tipo de bucles que existen son los bucles indefinidos, estos bucles se llaman así dado que no contienen un valor prefijado en el argumento del bucle que les diga cuando finaliza la iteración del propio bucle, sinó que dependen de una condición (igual que un `if`) que les indica cuándo finalizar las iteraciones. La sintaxis de los bucles `while` es la siguiente:

```
while(/* Condición(es) que indican el fin del bucle */){
  /* fragmento de código que queremos repetir */
}
```

Una de las ventajas que tiene el bucle `while` es que siempre podrá ejecutar la misma lógica de un programa que se haga con bucles `for` (aunque no sea eficiente en caso de que tengamos una condición de fin).

8.3.1.-Uso de inputs para definir fin de iteraciones

Tal y como hemos dicho anteriormente, por defecto, los bucles `while` son utilizados para realizar iteraciones de forma indefinida. Pero una de las excepciones utilizadas para definir el fin de ciclos de iteración del bucle, es el uso de algún tipo de condicional de comparación a través de los inputs realizados a alguna de las variables del programa, estos pueden ser Strings concretos, caracteres de marcación (e.g. `.`, `;`, `#`, etc.), cualquier valor que consideremos.

```
//INPUT>> "Pepe Paco Encarna Asunción Milagros Prudencio"

string nombres;
int count = 0;
while(cin >> nombres and nombres != "Milagros") { //Se asigna el valor a la variable
y se compara
  cout << "El/Ella es: " << nombres << '. ';  //Imprime el valor actual asignado a
la variable
  ++count;
}
cout << endl;
cout << "En total son " << count << endl;

//OUTPUT>> El/Ella es: Pepe. El/Ella es: Paco. El/Ella es: Encarna. El/Ella es:
Asunción.
//         En total son 4
```

En este ejemplo, hemos creado un programa que irá recogiendo distintos nombres escritos por el usuario, estos se irán almacenando de forma individual en la variable `nombres` declarada antes del bucle. Una vez asignado el valor, este tiene la condición de no entrar en el bucle si el string almacenado en la variable coincide con el string `"Milagros"`.

Así pues, siempre y cuando el nombre introducido no sea el de Milagros, el bucle imprimirá una frase donde presenta el nombre que se encuentra almacenado y suma +1 a un contador previamente declarado e inicializado a 0 antes del bucle. Una vez finalizadas las instrucciones dentro del bucle, se vuelve a repetir la acción de escribir un nombre, cuyo valor será sobreescrito en la variable `nombres`, borrando el valor anterior.

En el caso de que el string con el nombre coincida con la condición, este impedirá entrar en el bucle y pasará a las siguientes instrucciones; en este caso, imprimirá el número total de nombres que han pasado por el bucle, que está almacenado en la variable count.