Semana 1 : Introducción, Algoritmos y Pseudocódigo

### ****Teoría**** Qué es un algoritmo, un pseudocódigo y cómo representarlos con la herramienta PSeInt.

### Algoritmo

Un algoritmo es el conjunto ordenado y finito de pasos o instrucciones que debemos seguir para realizar una tarea o resolver un problema.

En nuestra vida diaria nos encontramos con muchos algoritmos como cuando seguimos el proceso de matricula en la universidad (cada universidad tiene bien definido los pasos a seguir para poder matricularnos) o por ejemplo cuando queremos tomar un bus del metropolitano por primera vez ( sabemos que tenemos que comprar una tarjeta, luego debemos recargarla para posteriormente el saldo sea validado por un lector y luego recién podemos subir a un bus).

### Pseudocódigo

El pseudocódigo es una manera de representar un algoritmo de tal forma que sea entendible para la lectura humana, es por ello que no existe una sintaxis estándar.

En general un pseudocódigo consta de las siguiente partes:

1. Nombre del algoritmo o proceso.
2. Inicio, indica el comienza del algoritmo.
3. Desarrollo, aquí se detalla el conjunto de pasos a seguir.
4. Fin, indica el final del algoritmo.

### Usemos PSeInt

El PSeInt es un herramienta que nos ayudará a dar nuestros primeros pasos en algoritmos y programación; nos sirve para poder escribir nuestro pseudocódigo, evitando así las dificultades propias de algún lenguaje de programación.

Como habíamos indicado antes, no existe un estándar en la sintaxis en pseudocódigo, por ello en este curso seguiremos la sintaxis propuesta por PSeInt.

* Instalación

La herramienta es gratuita y se puede descargar desde [aquí](http://pseint.sourceforge.net/). La instalación es muy intuitiva, usar la configuración sugerida por el asistente de instalación. Una vez instalado, al abrir un archivo nos aparecerá una ventana emergente preguntándonos que perfil deseamos, en este caso seleccionar**Flexible**

* Sintaxis

La sintaxis que usa PSeInt es la siguiente:

* + **Proceso "Nombre Proceso/Algoritmo":** nombra y da inicio al algoritmo.
  + **Instrucciones:** por ejemplo la instrucción Escribir, la cual nos permite mostrar mensajes en la consola de resultados. Vale indicar que cada instrucción debe terminar con un punto y coma ( ; )
  + **Fin Proceso:** indica el final del algoritmo.

Empezaremos con un ejemplo clásico, haremos un algoritmo que nos muestre el mensaje "Hola Mundo".

Proceso saludo

Escribir "Hola mundo";

FinProceso

* Ejecución

Al realizar nuestro pseudocódigo en PSeInt, este le dará las instrucciones a la computadora y nos permitirá ver los resultados en una ventana consola propia de la herramienta. Para ello sólo debemos de dar click en el ícono verde de la barra de herramientas, para mayor información ver el siguiente video.

### ****Teoría****Variables , Operadores Matemáticos, Lógicos y de Comparación.

### Variable

Es un espacio en la memoria de la computadora, dónde se va almacenar información y posee un nombre identificador asociado. Su valor puede cambiar durante la ejecución del algoritmo, es por ello que recibe el nombre de variable.

* Tipos de Datos

Las variables generalmente (en la mayoría de lenguajes de programación) poseen un **tipo de dato**, el cual nos indica la clase de dato con el que se va a trabajar y los valores que podrá tomar.

Existen 3 tipos de datos considerados como primitivos, los cuales usaremos para hacer nuestros pseudocódigos:

* + **Numérico:** números, tanto enteros como reales.
  + **Lógico o Booleano:** solo puede tomar los valores de verdadero o falso.
  + **Caracter**: caracteres o cadenas de caracteres encerrados entre comillas ('A', "hola mundo"). Se acostumbra poner los caracteres entre comillas simples y las cadenas entre comillas dobles.

PSeInt nos permite definir los siguientes tipos de datos, similar a los ya mencionados.

Proceso DefinicionVariables

Definir var1 Como Entero;

Definir var2 Como Real;

Definir var3 Como Logico;

Definir var4 Como Caracter;

FinProceso

* Asignación

Permite guardar un valor en una variable. Se utiliza el signo igual **(=)** para esta acción.

Proceso AsignacionValores

Definir a Como Entero;

Definir s Como Caracter;

a = 5;

s = "La variable a tiene el valor de 5";

FinProceso

* Lectura

Permite recibir valores ingresados por teclado y guardarlos en variables. Se pueden leer uno o más valores a la vez con una sola instrucción.

Proceso LeerValores

Definir a,b Como Entero;

Leer a,b;

FinProceso

* Escritura

Permite mostrar en pantalla algún dato o si son varios tienen que estar separados por comas **(,)**

Proceso EscribirValores

Definir nombre Como Caracter;

Leer nombre;

Escribir "El nombre ingresado es: ", nombre;

FinProceso

El siguiente video nos ayudará a complementar todo lo aprendido.

### Operadores Matemáticos

Son símbolos que nos permitirán realizar cálculos matemáticos, además de ser casi estándar en todos los lenguajes de programación.

Los principales operadores matemáticos son los siguientes:

|  |  |  |
| --- | --- | --- |
| **Operador** | **Significado** | **Ejemplo** |
| + | Suma | a = b + c |
| - | Resta | a = b - c |
| \* | Multiplicación | a = b \* c |
| / | División | a = b / c |
| ^ | Potenciación | a = base ^ exponente |
| MOD | Módulo (resto de la división entera) | resto = n MOD divisor |

### Operadores de Comparación

Son símbolos que nos permitirán comparar dos valores, si el resultado de la comparación es correcto entonces la expresión es considerada verdadera, caso contrario será falsa.

Los operadores de comparación son los siguientes:

|  |  |  |
| --- | --- | --- |
| **Operador** | **Significado** | **Ejemplo** |
| > | Mayor que | 10 > 8 |
| < | Menor que | 4 < 7 |
| >= | Mayor o igual que | 3 >= 3 |
| <= | Menor o igual que | 2 <= 6 |
| = | Igual a | 'A' = 'A' |
| != | Distinto a | 'A' != 'B' |

### Operadores Lógicos

Estos operadores nos pemitirán obtener un resultado verdadero o falso a partir de un conjunto de expresiones o condiciones.

Los operadores lógicos son los siguientes:

|  |  |  |
| --- | --- | --- |
| **Operador** | **Significado** | **Ejemplo** |
| Y | Conjunción: si ambas expresiones son verdaderas, el resultado es verdadero, caso contrario el resultado es falso. | (10>8) Y (5=6) devuelve falso |
| O | Disyunción: si ambas expresiones son falsas, el resultado es falso, caso contrario el resultado es verdadero. | (10>8) O (5=6) devuelve verdadero |
| NO | Negación: si la expresión es verdadera, el resultado será falso, caso contrario el resultado será verdadero. | NO (3>=3)  devuelve falso |

Ahora veremos un ejemplo usando operadores matemáticos, hallaremos la suma de dos números. El resto de operadores los usaremos cuando veamos la estructura de control condicional.

Proceso Suma

Definir a, b Como Entero;

Escribir "Ingrese valor del primer número: ";

Leer a;

Escribir "Ingrese el valor del segundo número: ";

Leer b;

Escribir "La suma de ", a, " + ", b, " es: ", a+b;

FinProceso

### ****Ejercicios****Uso de variables y operadores

Antes de continuar, reforcemos lo que hemos aprendido en esta primera sección con unos cuántos ejercicios muy simples. En este tipo de problemas no es necesario realizar validaciones, a no ser que explícitamente nos lo indiquen.

### Ejercicio 1

* Realizar un algoritmo que reciba de entrada dos números reales positivos, los cuáles representarán la base y la altura de un triángulo. Luego mostrar un mensaje indicando el área de dicho triángulo.

### Ejercicio 2

* Vasya tiene actualmente un sueldo de **x** soles. Anualmente recibe un incremento del **p**% sobre el sueldo que posea en ese momento. Realizar un algoritmo que reciba dos números reales x, p y luego nos muestre cuál sería el sueldo de Vasya de acá a dos años.

### ****eoría****Estructuras de Control Condicional: simple, doble, anidada y múltiple

### Estructura de Control Condicional

Nos permiten gobernar el flujo de ejecución de la instrucciones según se cumplan ciertas condiciones.

### Condicional Simple

Se encarga de evaluar una condición en caso sea verdadera realiza un conjunto de acciones, en caso contrario no hace nada. La estructura es la siguiente:

Si <condicion> Entonces

<instrucciones>

FinSi

Ahora veremos un ejemplo:

Proceso CondicionalSimple

Definir miEdad, tuEdad Como Entero;

Leer miEdad, tuEdad;

Si ( miEdad > tuEdad ) Entonces

Escribir "Soy mayor que tú";

FinSi

FinProceso

### Condicional Doble

Se encarga de evaluar una condición en caso sea verdadera realiza un conjunto de acciones, en caso contrario realiza otro grupo de acciones. La estructura es la siguiente:

Si <condicion> Entonces

<instrucciones1>

Sino

<instrucciones2>

FinSi

Ahora veremos un ejemplo:

Proceso CondicionalDoble

Definir miEdad, tuEdad Como Entero;

Leer miEdad, tuEdad;

Si ( miEdad > tuEdad ) Entonces

Escribir "Soy mayor que tú.";

Sino

Escribir "Soy menor que tú o tenemos la misma edad."

FinSi

FinProceso

### Condicional Anidada

Este tipo de condicional presenta condiciones a evaluar dentro de otras condiciones (combina la condicional simple y doble), éstas estructuras pueden seguir creciendo según como uno se las plantee. La estructura más básica es la siguiente:

Si <condicion> Entonces

<instrucciones1>

Si <condicion> Entonces

<instrucciones2>

FinSi

Sino

<instrucciones3>

Si <condicion> Entonces

<instrucciones4>

FinSi

FinSi

Ahora veremos un ejemplo:

Proceso CondicionalAnidada

Definir miEdad, tuEdad Como Entero;

Leer miEdad, tuEdad;

Si ( miEdad > tuEdad ) Entonces

Escribir "Soy mayor que tú.";

Sino

Si ( miEdad = tuEdad )Entonces

Escribir "Tenemos la misma edad.";

Sino

Escribir "Soy menor que tú";

FinSi

FinSi

FinProceso

### Condicional Múltiple

Esta estructura a diferencia de las otras, permite eligir una alternativa entre un conjunto de opciones. Aquí tenemos una variable que la podemos denominar **selector**, el cual va a comparar su valor con todas las**opciones** definidas y relizará las instrucciones que se indican en la opción que tenga su mismo valor. Si el valor del selector no coincide con ninguna opción se ejecutarán las instrucciones definidas en **"De Otro Modo"**. Es necesario saber que el selector y la opciones deben tener el mismo tipo de dato.

La estructura es la siguiente:

Segun <selector> Hacer

<opcion1>: <instrucciones1>

<opcion2>: <instrucciones2>

<opcion3>: <instrucciones3>

<...>

De Otro Modo: <instrucciones>

FinSegun

Ahora veremos un ejemplo:

Proceso CondicionalMultiple

Definir opcion Como Entero;

Escribir "(1)Algoritmos (2)Fronted (3)Backend (4)Android (5)Electrónica";

Escribir "Ingrese la opción del curso que desea llevar (1-5)";

Leer opcion;

Segun opcion Hacer

1: Escribir "Ud. seleccionó el curso de Algoritmos."

2: Escribir "Ud. seleccionó el curso de Frontend."

3: Escribir "Ud. seleccionó el curso de Backend."

4: Escribir "Ud. seleccionó el curso de Android."

5: Escribir "Ud. seleccionó el curso de Electrónica."

De Otro Modo: Escribir "Opción inválida."

FinSegun

FinProceso

### ****Ejercicios****uso de estructuras de control condicional

### Ejercicio 1

* Petr acaba de dar un examen de matemática en el colegio y el día de hoy recibirá su nota en calificación numérica (número entero del 0 al 20). Él se encuentra investigando acerca de los algoritmos, por ello desea que usted realize un algoritmo que dada su nota muestre su equivalente en calificación en letras.

La nota del 17 al 20 equivale a "AD", del 14 al 16 equivale a "A", del 11 al 13 equivale a "B", cualquier otra nota equivale a una "C".

### Ejercicio 2

* Un año es bisiesto si es divible entre 4, a menos que sea divisible entre 100. Sin embargo, si un año es divisible entre 100 y además entre 400, entonces también resulta bisiesto. Dado un año yyyy de entrada, se le pide mostrar un mensaje indicando si el año es bisiesto o no.

### ****Reto 1****Primer reto del entrenamiento de Algoritmos.

### Reto 1 de Algoritmos

Ya que cada vez nos familiarizamos más con los concursos de programación, debes saber que el día que hoy se realizó el "Hackspace Programming Contest" que consistía de 5 problemas de algoritmos para resolverlos en 2 horas, obviamente el ganador es quien ha resuelto la mayor cantidad de problemas. En este primer contest organizado sólo han participado 3 personas Jeffrey, Carlos y Josué, se sabe que han resuelto **a**, **b** y **c** problemas respectivamente. Los organizadores desean saber quién ha sido el ganador o si hubo empate, por ello le solicita a usted que realice un algoritmo que en base a la información brindada (a, b y c) muestre un mensaje con el nombre del ganador o de los ganadores.

Nota: Una restricción al problema es que sólo puede usar la teoría brindada hasta el momento (variables, operadores y estructuras de control condicional).

#### Semana 2 : Bucles y C++

### ****Teoría****Estructuras de Control Repetitiva: mientras, repetir-hasta y para.

### Estructura de Control Repetitiva

Llamamos así a los denominados bucles, los cuáles nos permiten repetir un conjunto de instrucciones hasta que una condición asociada al mismo se cumpla o deje de cumplirse.

### Mientras

Realiza un conjunto de instrucciones mientras una condición sea verdadera, esta estructura evalúa primero la condición antes de ejecutar la acciones, es por ello que las repetirá de 0 a más veces. La sintaxis es la siguiente:

Mientras <condicion> Hacer

<instrucciones>

FinMientras

Ahora veremos un ejemplo, hallaremos el factorial de un número n:

Proceso EstRepMientras

Definir n, i, fact Como Entero;

Leer n;

fact = 1;

i = 1;

Mientras (i<=n) Hacer

fact = fact \* i;

i = i + 1;

FinMientras

Escribir "El factorial de ", n, " es: ", fact;

Fin Proceso

### Repetir - hasta

Repite un conjunto de instrucciones hasta que una condición sea verdadera; a diferencia de la estructura anterior, esta realizará al menos una repetición, ya que la condición a evaluar se encuentra luego de la ejecución de las instrucciones. La sintaxis es la siguiente:

Repetir

<instrucciones>

Hasta Que <condicion>

Ahora veremos un ejemplo, validaremos el acceso a un sistema sólo comparanado el password y permitiendo como máximo 3 intentos:

Proceso EstRepetirHasta

Definir pass, clave Como Caracter;

Definir fallas Como Entero;

pass = "123";

fallas = 0;

Repetir

Escribir "Ingrese clave: ";

Leer clave;

Si clave != pass Entonces

fallas = fallas + 1;

FinSi

Hasta Que ( fallas = 3 O clave = pass)

Si ( fallas = 3 ) Entonces

Escribir "Ha superado el límite de intentos";

Sino

Escribir "Bienvenido";

FinSi

Fin Proceso

### Para

Principalmente permite ejecutar un conjunto de instrucciones un número determinado de veces (ya sabemos cuántas veces iterar). Opcionalmente se puede comportar como las estructuras anteriormente mostradas ya que itera sobre una variable desde un valor **"inicial"** definido, incrementándose en cada iteración en un valor llamado **"paso"**, mientras la variable sea menor o igual a un valor **"final"**( el campo "paso" es opcional, por defecto incrementa de 1 en 1). La sintaxis es la siguiente:

Para <variable> = <inicial> Hasta <final> ( Con Paso <paso> ) Hacer

<instrucciones>

FinPara

Ahora veremos un ejemplo, mostraremos todos los números pares desde el 1 hasta n.

Proceso EstRepPara

Definir n, i Como Entero;

Leer n;

Para i = 2 Hasta n Con Paso 2 Hacer

Escribir i;

FinPara

Fin Proceso

Vale destacar que para hacer un bucle podemos usar cualquiera de las estructuras presentadas, se recomienda usar la que más se nos acomode según sea el caso.

### ****Teoría****C++: conceptos y configuración del entorno de trabajo.

### Lenguaje de Programación

Es un lenguaje diseñado para describir un conjunto de acciones que son entendidos y que pueden ser ejecutados por máquinas como las computadoras, cada uno está formado por sus propios símbolos y reglas sintácticas. Existen muchos lenguajes de programación como C++, C#, Java, Ruby, Php, Python, etc.

### Compilador

Se encarga de convertir el código fuente hecho en algún lenguaje de programación a un lenguaje de máquina, para que sea comprendido por este mismo.

### ¿Cuál usaremos?

En nuestro caso usaremos C++, ya que es un lenguaje que está permitido en la mayoría de concursos, cuenta con muy buenas librerías usadas en la programación competitiva y además es mucho más fácil de aprender que otros lenguajes.

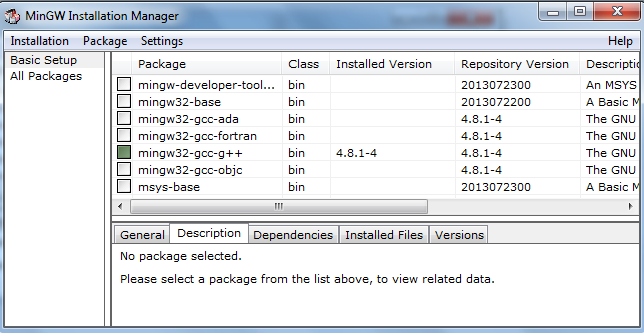
### Configuración del Entorno de Trabajo

1. Instalación de g++

C++ es un lenguaje que necesita de compilación, para ello tenemos disponible libremente el compilador g++, que para Windows podemos descargar la última versión desde [aquí](http://www.mingw.org/), en otros sistemas operativos probablemente ya lo tengan instalado y sólo tendrían que actualizarlo (googlear).

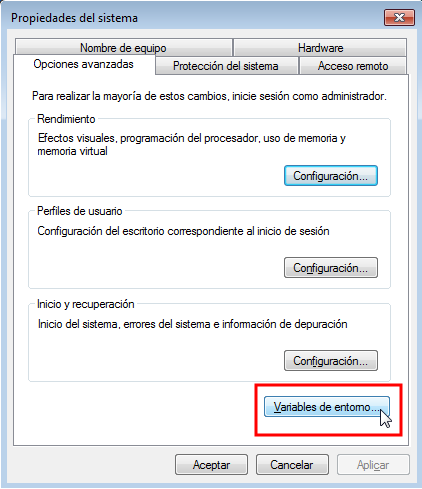
El compilador g++ permitirá a través de nuestro código fuente (.cpp) generar un ejecutable (.exe).

Seguiremos con la instalación en Windows (en otro SO es mucho más fácil), luego de descargar el instalador, utilizando el asistente realizamos la instalación por defecto, posteriormente nos aparecerá una pantalla donde seleccionamos **mingw32-gcc-g++**, luego para terminar damos click en**installation>Apply Changes**.

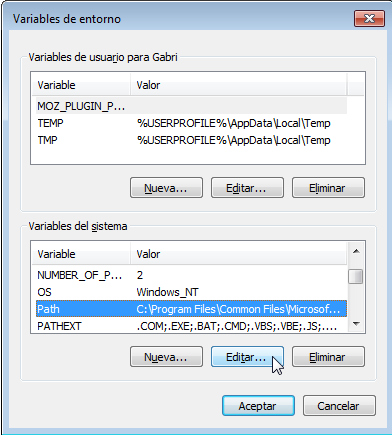


Ahora para que nuestra consola de Windows reconozca el compilador g++ (no es necesario hacer esto en otro SO), es necesario añadir la carpeta **bin** al **path** de Windows. Si no se cambió nada en la instalación, esta carpeta de encuentra en **C:\MinGW\bin.**

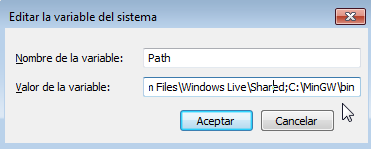
Vamos a **Panel de Control > Sistema > Configuración Avanzada de Sistema > Variables de Entorno**.



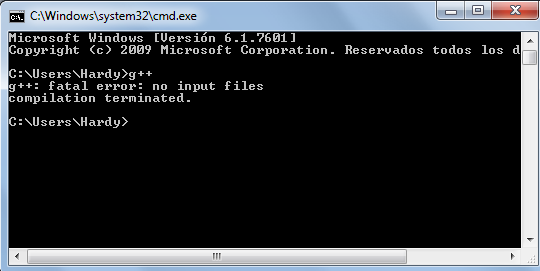
Como ya lo habíamos dicho tenemos que editar la variable de sistema **Path**.



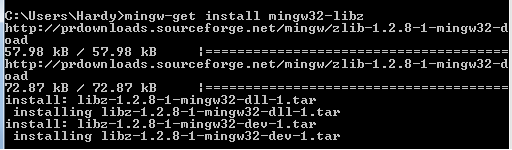
Ahora lo único que tenemos que hacer es agregar nuestra ruta **C:\MinGW\bin** al final del campo **valor de la variable**, previamente se coloca un punto y coma **(;)** para separar.



Para ir verificando que todo vaya bien, podemos entrar a la consola (cmd) y colocar g++ y luego enter. Nos debe aparecer el siguiente mensaje, en caso contario se debe haber hecho mal alguno de los pasos anteriores.



Para terminar con la correcta instalación del g++, debemos colocar la siguiente instrucción en la consola **mingw-get install mingw32-libz** y luego presionar enter, esto es debido a que en el instalador que descargamos no se encuentra una librería.



1. Sublime Text

Actualmente uno de los mejores y más comunes editores de texto es Sublime Text (que ya lo deben estar usando si también siguen el curso de frontend), es por ello que lo usaremos para C++, también pueden usar otros como Gedit, Codeblocks o Dev-C++. Lo pueden descargar desde [aquí](http://www.sublimetext.com/).

1. Prueba del Entorno

Esta prueba es sólo para Windows, para otros SO el entorno ya está configurado o basta con instalar el g++ pero de una manera mucho más sencilla (googlear)

Abrir una nueva hoja en Sublime Text, pegar el siguiente código y guardarlo como test.cpp en la ruta que crea Ud. conveniente.

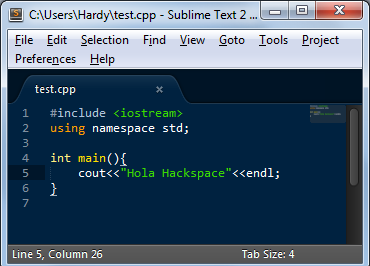
#include <iostream>

using namespace std;

int main(){

cout<<"Hola Hackspace"<<endl;

}



Luego abrir la consola y realizar los siguientes pasos:

* + Ubicarse en la ruta donde guardó el archivo test.cpp
  + Colocar **g++ test.cpp** y luego dar enter.
  + Colocar **a.exe** y luego dar enter.



El resultado que debemos obtener en la consola( ver imagen superior) es: **Hola Hackspace**, casos contrario debe haber cometido un error en algún paso anterior.

### ****Teoría****De un algoritmo a un lenguaje de programación (C++).

Ahora aprenderemos a convertir nuestro algoritmo en código C++, veremos que se nos hará fácil por previamente habernos familiarizado con el PSeInt. Básicamente sólo cambiará la notación, ya que la lógica en sí ya la aprendimos.

### Estructura de un programa en C++

#include <iostream> //las librerías van primero.

using namespace std; //espacio de nombres estándar.

[ declaración de variables ]

int main() //función principal del programa.

{ //inicio del programa

<instrucciones>

return 0;

} //fin del programa

Si no entendiste nada no te preocupes, lo primero que debes saber es que **"//"** significa que a continuación viene un comentario (algo que se omitirá al momento de ejecutarse el programa, se coloca generalmente para que otro que lea el código pueda entenderlo con facilidad). El resto de definiciones viene a continuación:

* Las **librerías** nos indican que tipos de funciones o instrucciones se pueden utilizar en el programa (sí, es algo nuevo para hacer un algoritmo no es necesario). Por ejemplo la de iostream nos permite usar el cin y cout que sirven para la entrada y salida de datos.

Los compiladores C++ nos ofrecen librerías o ficheros que permiten realizar un gran conjunto de tareas, incluyendo entrada/salida de datos, manipulación de cadenas, manejo de memoria, cálculos matemáticos, etc. Poco a poco las iremos conociendo pero pueden ir investigando sobre: iostream, cstdio, string, algorithm y cmath (ver más información al respecto [aquí](http://www.cplusplus.com/reference/)).

* El **espacio de nombres** engloba también un conjunto de funciones o instrucciones, no es suficiente sólo declarar la librería para poder usar una función o instrucción, también se debe declarar el espacio de nombres al que pertenecen. C++ ha puesto todo su estándar en el namespace std, por ello siempre debemos declararlo al inicio del programa.
* Una **función** es una agrupación de instrucciones, que finalmente retorna un valor (más adelante se explicará con mayor detalle) y se puede llamar desde otra función (para reutilizar código). Lo que debemos entender es que todo programa en C++ comienza con una función principal llamada **main()**y son la primeras instrucciones que realizará nuestro programa al ejecutarse (por ahora omitiremos las variables globales) y las llaves que lo acompañan indican el inicio y fin de programa respectivamente. Esta función main debe retornar 0 para indicar que todo es correcto.
* Las **llaves "{ }"** en C++ nos indican el inicio y fin de una función, procedimiento o alguna estructura ( condicional, repetitiva).

### Variables en C++

Los principales tipos de datos que usaremos en C++ son int (entero), double (real), long long (entero largo), char (caracter), bool (lógico) y string (cadena). Puedes ver más al respecto [aquí](http://www.cplusplus.com/doc/tutorial/variables/).

La asignación de valores a una variable es igual a como hemos venido trabajando hasta el momento mediante el signo igual (=). La lectura de una variable por el momento la haremos con la instrucción cin ( cin << ) y la escritura la haremos con cout (cout>>). Más adelante encontrarás ejemplos dónde podrás entender mucho mejor esto.

### Operadores

Veremos como se representan los operadores vistos la semana anterior, ahora en C++.

|  |  |
| --- | --- |
| **Operador** | **Significado** |
| + | Suma |
| - | Resta |
| \* | Multiplicación |
| / | División |
| pow(base, exponente) | Potenciación |
| % | Módulo (resto de la división entera) |
| > | Mayor que |
| < | Menor que |
| >= | Mayor o igual que |
| <= | Menor o igual que |
| == | Igual a |
| != | Distinto a |
| && | Conjunción |
| || | Disyunción |
| ! | Negación |

### Compilación y Ejecución de un Programa en C++.

Para compilar simplemente vamos a la consola e independientemente del sistema operativo, nos ubicamos en la ruta donde se encuentra el archivo .cpp que deseamos compilar y colocamos la siguiente linea:

g++ nombre\_archivo.cpp

Ahora al compilar se nos a generado un archivo a.exe en Windows y a.out en otros sistemas operativos, para ejecutar nuestro programa ahora colocamos en la consola (continuamos en la ruta en la que se encuentra el archivo):

a.exe //en Windows

./ a.out //en las distribuciones de Linux

### Estructuras Condicionales en C++

Como vimos la semana anterior, existen hasta 4 tipo de estructuras condicionales, pero ahora veremos las más interesante (la múltiple casi no se usa en concursos de programación, se las dejamos de tarea). La sintaxis básica es la siguiente, pero como ya lo habíamos comentado esta se puede ir expandiendo.

if( <condicion> ){

<instrucciones1>

}

else{

<instrucciones2>

}

Para que vean que no es nada complicado realizaremos el mismo ejemplo que se hizo en PSeInt, ahora en C++.

#include <iostream>

using namespace std;

int miEdad, tuEdad;

int main(){

cin >> miEdad >> tuEdad;

if( miEdad > tuEdad ){

cout << "Soy mayor que tu.";

}

else{

// para las condicionales si sólo se tiene una instrucción no es necesario colocar las llaves.

if( miEdad == tuEdad) cout << "Tenemos la misma edad.";

else cout << "Soy menor que tu.";

}

}

### Estructuras Repetitivas en C++

En este caso convertiremos la estructura mientras y para en C++, el repetir-hasta tiene una estructura parecida en C++, pero lo veremos después.

La sintaxis de la estructura **mientras** es la siguiente:

while( <condicion> ){

<instrucciones>

}

Realizaremos el mismo ejemplo que se hizo en PSeInt, ahora en C++.

#include <iostream>

using namespace std;

int n, i, fact;

int main(){

cin>>n;

fact = 1;

i = 1;

while( i<=n ){

fact = fact \* i;

i++; // es lo mismo que i = i+1

}

cout << "El factorial de " << n << " es: " << fact << endl; // el endl indica salto de línea

}

La sintaxis de la estructura **para** es la siguiente:

for ( <variable> = <inicial> ; <variable> <= <final> ; i = i + <paso> ){

<instrucciones>

}

Realizaremos el mismo ejemplo que se hizo en PSeInt, ahora en C++.

#include <iostream>

using namespace std;

int n, i;

int main(){

cin>>n;

for( i=2; i<=n; i = i+2 ){

cout << i <<endl;

}

}

### ****Ejemplos****problemas resueltos usando C++.

### Ejemplo 1

Leer un número entero X y luego imprimir los siguientes 6 consecutivos números impares desde X. Incluir X si este es impar.

**Entrada**

La entrada contiene un número entero X ( 1 ≤ X ≤ 1000 )

**Salida**

Imprimir las secuencia de números impares, uno por línea.

**Ejemplos**

|  |  |
| --- | --- |
| **Entrada** | **Salida** |
| 8 | 9 11 13 15 17 19 |

**Solución:**

El problema se resuelve en dos pasos:

* Encontrar el primer número de la secuencia. Para eso hay que verificar si el número que leemos es impar. Si no es impar es claro que para obtener el primer número de la secuencia hay que sumarle uno.
* Ir imprimiendo desde el primer número aumentando 2 en cada iteración.

El código en C++ es el siguiente:

#include <iostream>

using namespace std;

int main() {

int n; //Declaramos la variable para leer el número

int seq; //Declaramos una variable para la secuencia.

cin >> n; //Leemos el número

//Primer paso: Identificar el primer número de la secuencia.

if (n%2 == 0) seq = n + 1;

else seq = n;

//Segundo paso: Imprimir los números, empezando por el primero y avanzando de 2 en 2

for (int i = 0; i < 6; i++) {

cout << seq << endl;

seq += 2;

}

}

### Ejemplo 2

Leer un número entero X y luego imprimir los diez consecutivos números primos desde X. Incluido X si es primo.

**Entrada**

La entrada contiene un número entero X ( 1 ≤ X ≤ 10000 ).

**Salida**

Imprimir los diez números primos uno por línea en orden ascendente.

**Ejemplos**

|  |  |
| --- | --- |
| **Entrada** | **Salida** |
| 1 | 2 3 5 7 11 13 17 19 23 29 |

**Solución:**

Para resolver el problema primero debemos saber hallar si un número es o no primo. Por definición un número primo es aquel número natural mayor que uno que admite únicamente dos divisores: el mismo número y el 1.

Entonces para ver si un número es primo vamos a intentar encontrarle un divisor mayor que 1 y menor que el mismo número. Si no encontramos divisor alguno concluiremos que es primo, caso contrario no lo es.

Luego lo que tenemos que hacer es iterar desde el número que leemos e ir contando la cantidad de primos que vamos encontrando, cuando tenemos los diez terminamos la iteración.

El código en C++ es el siguiente:

#include <iostream>

using namespace std;

int main() {

//Leemos el numero x

int x;

cin >> x;

//Declaramos una variable para contar los primos que vamos encontrando y otra para iterar sobre los posibles primos.

int contador = 0, posiblePrimo;

//Inicializamos posiblePrimo evitando el 1.

if (x <= 1) {

posiblePrimo = 2;

} else {

posiblePrimo = x;

}

//Vamos a buscar primos hasta tener los 10 que necesitamos.

while (contador < 10) {

//Declaramos una variable para guardar el resultado de la verificacion.

//Al inicio asumimos que es primo, por eso iniciamos en true.Si encontramos algun divisor, inmediatamente lo cambiamos a false.

//Y si nunca encontramos un divisor, la variable terminara siendo verdadera, que es justo lo que buscamos.

bool esPrimo = true;

//Empezamos a buscar un divisor desde 2

for (int divisor = 2; divisor < posiblePrimo; divisor++) {

if (posiblePrimo % divisor == 0) {

esPrimo = false;

//Basta que encontremos un divisor para descartar que sea primo, el bucle debe terminar.

break;

}

}

//Si es primo mostramos el número y contamos en la variable contador.

if (esPrimo) {

cout << posiblePrimo << endl;

contador++;

}

posiblePrimo++; //Le sumamos 1 para probar con el siguiente.

}

return 0;

}

### ****Ejercicios****aplicación de todo lo aprendido.

### Ejercicio 1

Denominaremos a un número como cuadrado perfecto si es igual a un número entero positivo elevado al cuadrado. Se le pide a usted dado un número n, indicar cuántos cuadrados perfectos existen menores o iguales a n.

**Entrada**

La entrada contiene un número entero n ( 1 ≤ n ≤ 100000 )

**Salida**

La cantidad de números cuadrados perfectos menores o iguales a n

**Ejemplos**

|  |  |
| --- | --- |
| **Entrada** | **Salida** |
| 5 | 2 |
| 39 | 6 |

### Ejercicio 2

Se define la función f(n) para un número entero positivo.

f(n) = -1 + 2 - 3 + .. + (-1)n n

El reto es calcular f(n).

**Entrada**

Una sola línea que contiene un numero entero n. (1 ≤ n ≤ 10000).

**Salida**

El valor de f(n).

**Ejemplos**

|  |  |
| --- | --- |
| **Entrada** | **Salida** |
| 4 | 2 |
| 5 | -3 |

### ****Reto 2****Segundo reto del entrenamiento de Algoritmos.

### Reto 2 de Algoritmos

Toñito está apunto de ingresar a un edificio de n pisos (considerar que está en el piso 0). Se sabe que cada hora el sube A pisos ( dejando una bandera sólo en el último piso al que llegó) y luego desciende B pisos, según él hace esto para realizar mayor ejercicio y estar mejor de salud. Se recalca que no deja banderas en ningún piso intermedio. El Hackspace desea saber si Toñito pudo dejar una bandera en el piso n, teniendo en cuenta que cuando ya no puede seguir con su rutina ( ya no hay los suficientes pisos en el edificio) baja rápidamente y va rumbo a su casa( ver ejemplos).

**Entrada**

La entrada contiene 3 números enteros n, A y B ( 1 ≤ n, A, B ≤ 10000 ; A>B )

**Salida**

Si existe una bandera en el piso n imprimir "SI", caso contrario mostrar "NO" (sin comillas).

**Ejemplos**

|  |  |
| --- | --- |
| **Entrada** | **Salida** |
| 10 15 7 | NO |
| 5 3 1 | SI |

En el primero caso, el edificio sólo tiene 10 pisos y su rutina le dice que primero tiene que subir 15 pisos, por ello decide irse a su casa.

En el segundo caso, sube hasta el piso 3, baja al piso 2 y luego llega al piso 5 dejando una bandera.

#### Semana 3 : Complejidad de un Algoritmo y más problemas.

### ****Teoría****Análisis de la complejidad de un algoritmo.

Hasta el momento hemos resuelto varios problemas omitiendo un análisis que es necesario. Generalmente cuando intentamos dar solución a un problema se nos viene a la mente varias posibilidades y escogemos la que se nos hace más fácil, pero no estamos analizando si quizá es demasidado lenta, puede que en lugar de segundos tome minutos en ejecutarse.

Ahora, ¿cómo sabemos el tiempo de ejecución de un algoritmo?, fácil es igual al número de operaciones (pasos) que realiza el mismo; por ejemplo: asignación, suma, resta, multiplicación, división, una iteración, evaluar una condición, etc.

Pero si nos damos cuenta, este tiempo varía de acuerdo a como varía el tamaño y orden de los datos de entrada (no es lo mismo leer 100 números a leer 100000 números). Ante esto surge el término **worst-case runtime** (el peor de los casos), el cual representa el tiempo de ejecución más largo que encontremos para una entrada dada según las restricciones previamente definidas de la misma (los problemas detallan las restricciones generalmente para el valor que puede tomar algún dato).

Formalmente, si tenemos una función f(n) , donde n representa el tamaño de la entrada y f(n) nos indica el tiempo de ejecución del algoritmo para esa entrada, podemos definir la notación **O grande**, la cuál nos permitirá encontra otra función g(n) que es cota superior de nuestro función f cuando n tiene al infinito.

Podemos denotarlo de la siguiente manera **f(n) = O(g(n))**.

Por ejemplo, si tenemos una función cuyo worst-case runtime es 2n2, entonces nuestra cota superior puede ser un 3n2, pero podemos omitir la constante y decir que **f(n) = O(n2)**.

En los concursos de programación generalmente se espera que un algoritmo no realice más de 108operaciones, que demoran aproximadamente 1 segundo, es por ello que los límites aproximados en la entrada para las complejidad más comunes son los siguientes:

O(N) : N hasta 108

O(N2) : N hasta 104

O(N3) : N hasta 400.

O(log N) : número muy grande.

O(2N) : N hasta 24.

Para mucho mayor información en el tema, se recomienda leer sólo la sección 1 del siguiente tutorial de Topcoder: ver [aquí](https://www.topcoder.com/community/data-science/data-science-tutorials/computational-complexity-section-1/)

#### Semana 4 : Reto Final

### ****Reto Final****Reto final del entrenamiento de Algoritmos.

### Reto Final de Algoritmos

Dado un número n , se desea saber cúantos números primos existen desde 1 hasta n.

**Entrada**

La entrada contiene un número entero n (1 ≤ n ≤ 100000)

**Salida**

La cantidad de números primos solicitada.

**Ejemplos**

|  |  |
| --- | --- |
| **Entrada** | **Salida** |
| 12 | 5 |
| 2 | 1 |