Apuntes Pro1 Parcial 1

Declaración de variables:

```
int x;
double x;
float x;
```

La diferència entre Float y Double es que Double tiene más decimales de precisión.Double 64 bits, float 32

```
string paraula;
char lletra;
bool trobat;
```

Condicionales If , else if , else :

```
if ( x < 10 ) {

}
```

En el if sólo entramos si la condición dentro de los paréntesis se cumple. Si lo que queremos es comprobar varias condiciones una tras otro podemos hacer lo siguiente:

```
if ( x < 10) …
else if ( x > 10 ) …
else {...}
```

En este ejemplo mostrado qué se puede observar es el siguiente: Comprobamos primero si X <10, en caso de que lo sea entra y no efectuará ni el else if ni el else, si no lo es irá al else if y así sucesivamente hasta llegar al else. Si la condición de alguno se cumple al efectuar el código suponemos del IF posteriormente ya no comprobará la condición del else if o else e irá hacia la instrucción que haya bajo el else. Esto lo podemos ver más claro en el siguiente ejemplo:

```
if ( x < 10) …
if ( x > 10) …
if ( x == 10) …
```

En este ejemplo como podemos observar hemos sustituido el else por un if y el else if por otro if. ¿Cuál es la diferencia entonces? Realmente estamos haciendo lo mismo?

El motivo de utilizar if y else / else if radica en que ahora si x <10 entrará en el if pero una vez efectuado el código irá al segundo if cuando ya sabemos que x <10 por tanto los otros 2 IFS no se cumple la función entonces estamos revisando condiciones y ejecutando instrucciones completamente innecesarias y qué podemos ahorrar con un else if o else. A clase se profundizará en casos como este.

Entrada y salida de valores:

```
cin >> x;
cout << x;
```

También es posible hacer una entrada dentro de un bucle. Este while terminará justo cuando dejen de entrar valores para el terminal

```
while ( cin >> x ) {
        …
}
```

Bucles while i for:

```
while ( cond a complir per entrar dentro del bucle) {
        ...
}
```

Al while entraremos siempre que la condición de entrada se cumpla. Debe tener cuidado siempre con qué condiciones coloque. Muchas veces os podréis encontrar qué se queda en bucle infinito el while por haber definido mal las condiciones de entrada o qué nunca entre.

Caso típico de bucle infinito:

```
int main() {

        int x = 0;
        while (x < 10) {
                cout << x << endl;
        }

}
```

Pregunta : Porque se queda en bucle infinito?

Resposta : No incrementamos el valor de x y por lo tanto se queda siempre a 0 cumpliéndose infinitamente la condición.

¿Qué podemos hacer para saber que tenemos un bucle infinito?

Una de las formas más utilizadas para depurar es mediante cout. Si introducimos COUTS dentro del bucle mirando si aquel cout sale por pantalla x golpes o qué valor toma la x podremos descubrir que nos hemos dejado.

Declaración de for :

```
for ( type ; cond ; inc / dec ) {
        ...
}
```

Ejemplos:

```
for(int i = 0; i < n ; ++i) {

        …
}

for(int i = 10; i > n ; --i) {
        …
}
```

Para esta asignatura lo qué se hará será qué el for será un bucle que utilizaremos siempre y cuando sólo tengamos una condición para seguir en el bucle y siempre se efectúen todas las iteraciones. Si en cualquier iteración se puede dar que hemos obtenido el resultado deseado utilizaremos el while, de lo contrario el for. Esto lo entenderemos mejor en el siguiente ejemplo. En asignaturas posteriores aprenderéis sentencias qué nos permiten romper el sequenciamento de un for y salir antes. Por ahora seguiremos lo descrito anteriormente.

Supongamos que queremos saber los números son pares del 1 al 10. Sabemos pues que nuestra condición del bucle será para x <= 10 y que los queremos mirar todos. En este caso el bucle for será el que utilizaremos.

```
for (int i = 1 ; i <= 10; ++i) {
        if ( i % 2 == 0) cout << i << endl;
}
```

Ahora supongamos el siguiente ejercicio. Dado un número entre 1 y 10 queremos saber si el número que nos entra por terminal se encuentra entre estos 10 primeros dígitos. En este caso nos encontramos que el número que el usuario puede entrar puede ser del 1 al 10.

```
int num;
cin >> num;
int x = 1;
bool trobat = false;
while ( not trobat ) {
        if ( x == num)  trobat = true;
        else {
                ++x;
        }
}
```

Supongamos que es el 1. ¿Por qué tenemos que recorrer los 9 números restantes cuando ya hemos encontrado el número deseado? Aquí es donde podemos encontrar la diferencia y la respuesta a cuándo utilizar un for o un while. Siempre que el intervalo que utilizamos entre "y" y "n" suponiendo n como valor final nos tendremos que revisar todos SI o SI sin excepción utilizaremos for, por otro lado si lo que estamos mirando puede que se cumpla antes de llegar utilizaremos while.