Apunts Pro1 Parcial 1

Declaració de variables:

```
int x;
double x; // 64 bits
float x;  // l'utilitzareu poc a Pro1 però no està malament saber-lo 32 bits
```

La diferència entre Float i Double és qué Double té més decimals de precisió.

```
string paraula;
char lletra;
bool trobat; // condició boleana
```

Condicionals If , else if , else :

```
if ( x < 10 ) {

}
```

En el if sols entrem si la condició dins dels parèntesis es compleix. Si el que volem és comprovar varies condicions una rere altre podem fer el següent:

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

En aquest exemple mostrat el qué es pot observar és el següent: Comprovem primer si X < 10 , en cas de que ho sigui entra i no efectuarà ni el else if ni el else, si no ho és anirà al else if i així successivament fins arribar al else. Si la condició d'algun es compleix al efectuar el codi suposem del IF posteriorment ja no comprovarà la condició del else if o else i anirà cap a la instrucció que hi hagi sota el else. Això ho podem veure més clar en el següent exemple:

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

En aquest exemple com podem observar hem substituït el else per un if i el else if per un altre if. Quina és la diferència doncs? Realment estem fent el mateix?

El motiu de utilitzar if i else / else if rau en què ara si x < 10 entrarà al if però un cop efectuat el codi anirà al segon if quan ja sabem que x < 10 per tant els altres 2 ifs no es compleix la condició llavors, estem revisant condicions i executant instruccions completament innecessàries i què podem estalviar amb un else if o else. A clase es profunditzarà en casos com aquest.

Entrada i sortida de valors:

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

També és possible fer una entrada dins de un bucle. Aquest while acabarà just quan deixin de entrar valors per el terminal

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

Bucles while i for:

```
while ( cond a complir per entrar dins el bucle) {
        ...
}
```

Al while entrarem sempre que la condició d'entrada es compleixi. Heu de vigilar sempre amb quines condicions poseu. Molts cops us podreu trobar qué es queda en bucle infinit el while per haver definit malament les condicions d'entrada o qué mai entri.

Cas típic de bucle infinit:

```
int main() {

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

}
```

Pregunta : Per què es queda en bucle infinit?

Resposta : No incrementem el valor de x i per tant es queda sempre a 0 complint-se infinitament la condició.

Que podem fer per saber què tenim un bucle infinit?

Una de les formes més emprades per debugar és mitjançant cout. Si introduïm couts dins el bucle mirant si aquell cout surt per pantalla x cops o quin valor pren la x podrem descobrir que ens hem deixat.

Declaració de for :

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

Exemples :

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

        …
}

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

El for serà un bucle que utilitzarem sempre i quan sols tinguem una condició per seguir en el bucle i sempre s'efectuin totes les iteracions. Si en qualsevol iteració es pot donar que hem obtingut el resultat desitjat utilitzarem el while , altrament el for. Això ho entendrem millor en el següent exemple.

Suposem que volem saber quins números son parells del 1 al 10. Sabem doncs que la nostra condició del bucle serà per x <= 10 i que els volem mirar tots. En aquest cas el bucle for serà el que utilitzarem.

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

Ara suposem el següent exercici. Donat un número entre 1 i 10 volem saber si el número que ens entra per terminal es troba entre aquests 10 primers dígits. En aquest cas ens trobem que el número que el usuari pot entrar pot ser del 1 fins al 10.

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

Suposem que és el 1. Per què recorre els 9 números restants quan ja hem trobat el número desitjat? Aquí és on podem trobar la diferència i la resposta a quan utilitzar un for o un while. Sempre que el interval qué utilitzem entre " i " i " n " suposant n com a valor final els haurem de revisar tots SI o SI sense excepció utilitzarem for, per altre banda si el que estem mirant pot ser que es compleixi abans d'arribar utilitzarem while.

Nota: Més endavant a EDA o Pro2 apendreu sentències que permeten sortir d'un FOR sense acabar-lo. A pro1 suposarem aquestes normes per aprendre els conceptes bàsics i quan queda millor emprar FOR o while a nivell visual.