UPC - FIB - Pro1 - Dru

Includes:

```
#include <iostream>  //carga los valores del sistema operativo

using namespace std; // utiliza el conjunto de nombres standard
```

.....

function MAIN ese gran desconocido.

```
int main(){
        #code...
        ........
        ........
}
```

Variables:

```
miTipoDeVariable nombreDeMiVariable;
miTipoDeVariable nombreDeMiVariable = valorInicial;

int  Numeros REALES
double Numeros QUEBRADOS
bool 0/1 CIERTO/FALSO
char ['a'-'Z']
string conjunto de chars.
```

---

condicionales (EL IFFFF)

```
if( [evalua una condición CIERTA/FALSA] )  ej: if(3 > 2)  // CIERTO  || ej: if(2 > 3) // FALSO

if(3>2) cout << "Perfecto" << endl;
if(true) cout << "Siempre escribiré esto." << endl;
else cout << "Esto jamás se imprimirá". << endl;


if + else if()

n = 6;
if(n < 0) n *= 3;
```

```
else if(n == 0) n = 1;
else if( n > 0)  n=n*2; // = else (es la única opción restante).
```

BUCLES WHILE & FOR

```
while( [se cumpla el condicionante] ){

        # se ejecutará el codigo n veces en función de la condición del while
}

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

        # se ejecutará n veces desde [0 a algo-1];
}
```

Entradas y salidas de datos standard

```
object data;  // object = int, string, bool, char…. lo que se os ocurra

cin >> data; // $data <- lectura de teclado

cout << data; // $consola <- $data

cout << data << endl; // $consola <- $data + '\n' (Salto de linea)
```

BUCLES CONDICIONALES

```
while(cin >> n){  //mientras haya una entrada...
}

while(cin >> n and n != 0){ // mientras haya una entrada y esta sea diferente de 0.
}
// cuidado, while(n != 0 and cin >> n) no funciona igual !!!

bool trobat = false;
for(int i = 0; i < algo and not trobat; ++i){  // ejecutará una busqueda, si encontramos algo,para
if(algunaCosa[i] == otraCosa) trobat = true;
}
```

BUCLES DENTRO DE BUCLES (Loops inside Loops)
Cuando son prácticos los bucles dentro de bucles?
- Cuando queremos desglosar un problema en fases, como por ejemplo, pintar dibujos.

Veamos un ejemplo para dibujar un triangulo rectangulo de n base y altura:

```
for(int i = 1; i <= n; ++i){
        for(int j = 0 ; j < i; ++j) cout << “X”;
        cout << endl;
}
```

Resultado:

```
x
xx
xxx
xxxx
xxxxx
```

Si nos fijamos atentamente, este bucle inicial tiene un valor de 1 sobre n.
Según la i crece, las x por línea también en un orden de +1 por cada iteración hacia n.
Ahora, hagamos algo más divertido y complicado, el mismo bucle con efecto espejo.

```
for(int i = 1; i < n; ++i){
        int j: // como la J va a ser reciclado, la mantenemos
        for(j = 0; j < (n-i); ++j)cout << “ “;  //forma 1.
        for(; j < n; ++j){ // un for de 1 linea dentro se puede escribir de estas 2 formas.
                cout << X; //forma 2.
        }
        cout << endl;
}
```

Resultado:

```
      x
     xx
   xxx
  xxxx
xxxxx
```

De la misma forma podemos hacer otras figuras simplemente desglosando el problema por partes (blancos y pintados) y de la misma forma, aplicando teoremas inversos se puede trabajar con efectos espejo. (como en el ej. anterior).

FUNCIONES. La elegancia hecha código.

Una función es un trozo de código independiente que en función de la entrada devuelve una salida determinada. por ejemplo, f(x).

f es el nombre de la función, x, la variable que le pasamos.

absolutamente TODAS las funciones devuelven algo, excepto la VOID (vacío).

para declarar una función tenemos 3 posibles parámetros a definir.

Tipo de función: void, int, double, char, string….. (y otros que ya veremos).
Nombre de la función, por ahora debe ser único! se suele recomendar aplicar el criterio LCC (lowerCamelCase).
Variables que recibe (ints, chars, bla bla bla… todo lo que queráis pasar-le para que pueda trabajar).

Ejemplos de funciones:

```
void imprimirDades(string dades){
}

bool esParell(int n){
	return 0;  // todas las funciones devuelven algo en función del algoritmo que tengan.
} // el return 0; es solo un ejemplo !!!!


char devolverPrimeraLetra(string palabra){
	return palabra[0]; //este ejemplo es bastante más realista
}

string leerCadenaDeChars(){ // Si, una función puede no recibir parámetros :P
	char a;
	string aux= “”;
	while(cin >> a){
		aux = aux + a;
	}
	return aux;
}
```

Las funciones pueden tener también parámetros IN/OUT, es decir, variables que permutan en el tiempo.

Veamos algunos ejemplos:

```
/*  includes etc...  */
void verMax(int& a, int b){ //Fijarse en que a se pasa por REFERENCIA ( int & a) & = x ref.
        if( b > a) a = b;
}

void(const vector<int>& v, int& max){
        int aux = v[0];
        for(int i = 1; i < aux.size(); ++i){
                verMax(aux,v[i]);
        }
}
/*  more code + main() here  */
```

Oh, magia, tras pasar por este codigo algo absurdo, veremos que “max” vale el mayor de los ints del vector, y todo el codigo intermedio desaparecerá sin dejar rastro.
Esto es útil cuando hay que desplazarse recursivamente o iterativamente por vectores y estructuras grandes donde es primordial saber en que puntos estamos en cada momento dentro de las funciones sin tener que recurrir a multiples factores de salida (hacer que una función devuelva más de un único item a la vez).

RECURSIVIDAD || “La función” o “un sueño dentro de un sueño dentro de un sueño…”

La recursividad en las funciones trabaja de forma parecida a la inducción.
Tenemos un caso base que nos hará salir del bucle con un return x; (llamdas constantes a la misma función). En el resto de casos, tendremos uno o mas disparadores que lanzaran la misma función con un pequeño cambio dentro.

Pongamos un pequeño ejemplo:
Dados dos números x,y, escribir una línea con todos los enteros que intervengan separados por comas 0<=x<=y.

```
void recursive(int x, int y){
        if(x == y) cout << y << endl;
        else{
                cout << x;
                recursive(x+1,y);
        }
}
int main(){
        int x,y; cin >> x >> y; recursive(x,y);
}
```

LOS VECTORES o listas…. (conjuntos de items).

vector <int> v(n,0); //vector con tamaño n de INT's
vector <int> v(n); // son lo mismo, solo que el 1º se inicializa a ceros TODO

vector<string> v(3,"fuck!"); // a ver si adivinais como se inicializa esto :P

vector<char> v(n) ≈ string v; // un string se puede entender como un vector de chars (+ o -)

Como se accede a un vector llamado v de n posiciones?
0 <= i < v.size();
v[i] = …. o cout << v[i] << endl; // por ejemplo

Así de simple?
- Si el vector es de un único tipo (int, char, string, double), si. En vectores mas complejos hay truco :P

LOS STRUCTS (pseudo-objetos):

un Struct es un conjunto de datos con un nombre propio.
Por ejemplo, definamos el Struct "Estudiante".

```
struct Estudiante{
        int dni;
        string nombre;
        double nota;
}; // fijense que termina en punto y coma !!

//ahora yo puedo crear un "objeto" de tipo Estudiante sin ningún problema.
Estudiante e;
// y claro, puedo rellenarlo, ¿Como?, accediendo a sus ítems mediante el punto.

e.dni = 12345678;
e.nombre = "Dru";
e.nota = 10;
//Y para imprimir datos de un estudiante?

cout << "Soy: " << e.nombre << ", con DNI: " << e.dni << " y nota: " << e.nota << endl;
```

```
// Y que pasa cuando tengo que almacernar varios estudiantes (por ejemplo n) ?

vector<Estudiante> v(n); // :D

for(int i = 0; i < v.size(); ++i){
        cin >> v[i].e.dni >> v[i]e.nombre >> v[i]e.nota;
} // ale, ya tengo todos mis estudiantes guardados :D

// y si quiero imprimirlos ? :S

for(int i = 0; i < v.size(); ++i){
        cout << “Estudiante: “ v[i].nombre << “, con DNI: “ << v[i].dni << “, tiene una nota de: “ <<
v[i].nota << “.” << endl;
}

//Pero, y si me los pide ordenados por nombre?
bool sorter(Estudiant& a, Estudiant& b){
        return a.nombre > b.nombre;
}

// Y por DNI inverso? O.o
bool sorter(Estudiant& a, Estudiant& b){
        return a.dni < b.dni;
}

// Vale vale, pero, y si me piden por nombre, y en caso de coincidencia, por por DNI ascendente
que? eh? ¬¬¡
bool sorter(Estudiant& a Estudiant& b){
        if(a.nombre == b.nombre){ // nombres iguales
                return a.dni > b.dni;
        }else{ // diferente nombre
        retrurn a.nombre < b.nombre;
}


// En cualquiera de los “sorter’s” que useis, es recomendable aplicar la función “sort”.
//Por eso de que, sino no funciona y tal...  ^^¡
sort(v.begin(),v.end(), sorter);


----------
```

LOS TYPEDEF (o como renombrar cosas mu largas …)

typedef int comoMola;

/* Si, ahora usted puede hacer ints o comoMolas tan felizmente:
int a = 3; es lo mismo que: comoMola a = 3;
Absurdo, pero… y si trabajamos con datos mas complejos? */

Por ejemplo, recordais vuestros estudiantes Estudiantes?

//No es lo mismo crear una funcion llamada:
vector<Estudiantes> arreglarVector(const vector<Estudiantes>& v);
//que hacer una:
lista arreglarVector(const lista& v);

y, como hago una “lista” ?

typedef vector<Estudiantes> lista; // y me quedo tan pancho!

- FIN TEMARIO -

Vamos a jugar un poco con los “peudo-objetos” que hemos trabajado hasta el momento:

Igual que yo puedo hacer: int i; i = 3; puedo hacer: lista r; r = leerVector();
donde “leerVector” leerá un conjunto de estudiantes.

```
lista leerVector(){
	int size;
	cin >> size;
	lista v(size);
	for(int i = 0; i < size; ++i){
		cin >> v[i].dni >> v[i].nombre >> v[i].nota;
	}
	return v;
}

//y ahora viene lo divertido

int main(){
	lista miLista = leerVector();
	// ale, ya tenemos un vector de tamaño n directamente incrustado en una variable.
}
```