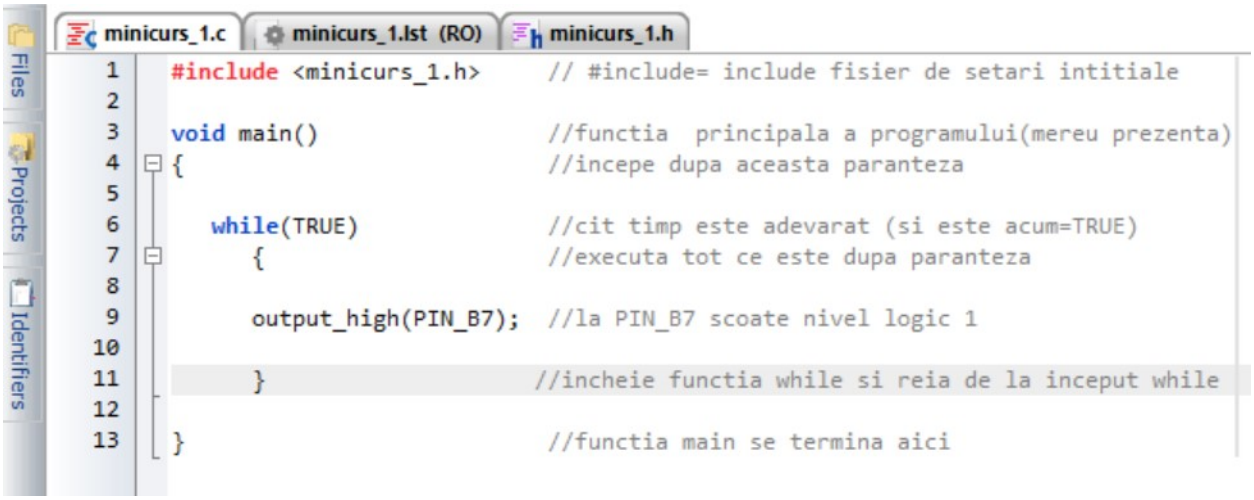
2.Instructiuni de iesire

2.1 Aprindere LED / Output

Stergeti **//TODO: User Code** si scrieti instructiunea : **output_high(PIN_B7)**
Aceasta instructiune trimite la pinul B7 un nivel logic ridicat (*high).*

Daca doriti sa trimiteti un zero logic la iesire utilizati instructiunea: **output_low(PIN_B7).** Aceasta se poate utiliza de exemplu la initializarea unei iesiri la inceputul programului pentru a fi siguri de starea sa. Se poate utiliza ori de cite ori este nevoie sa trecem iesirea in zero logic.

In ambele cazuri in loc de B7 puteti scrie orice pin care poate fi iesire.




```
1   #include <minicurs_1.h>     // #include= include fisier de setari intitiale
2
3   void main()                 //functia  principala a programului(mereu prezenta)
4   {                           //incepe dupa aceasta paranteza
5
6      while(TRUE)              //cit timp este adevarat (si este acum=TRUE)
7         {                     //executa tot ce este dupa paranteza
8
9         output_high(PIN_B7);  //la PIN_B7 scoate nivel logic 1
10
11        }                    //incheie functia while si reia de la inceput while
12
13  }                           //functia main se termina aici
```

Observati ca dupa fiecare instructiune se pune **;** (punct si virgula).
Observati ca punind **//** se pot scrie comentarii (care nu se executa!).

De acum inainte dupa ce veti scrie un program veti apasa **F9 (compile)** si veti transfera fisierul hex generat de acest compilator in microcontroler. Fisierul se gaseste in acelasi folder unde ati salvat si proiectul curent in CCSC.
Veti utiliza programatorul PICKIT2 si instructiunile aferente modulului PIC-AP20.
Verificati aprinderea LED-ului conectat. LED se stinge doar la oprirea alimentarii.

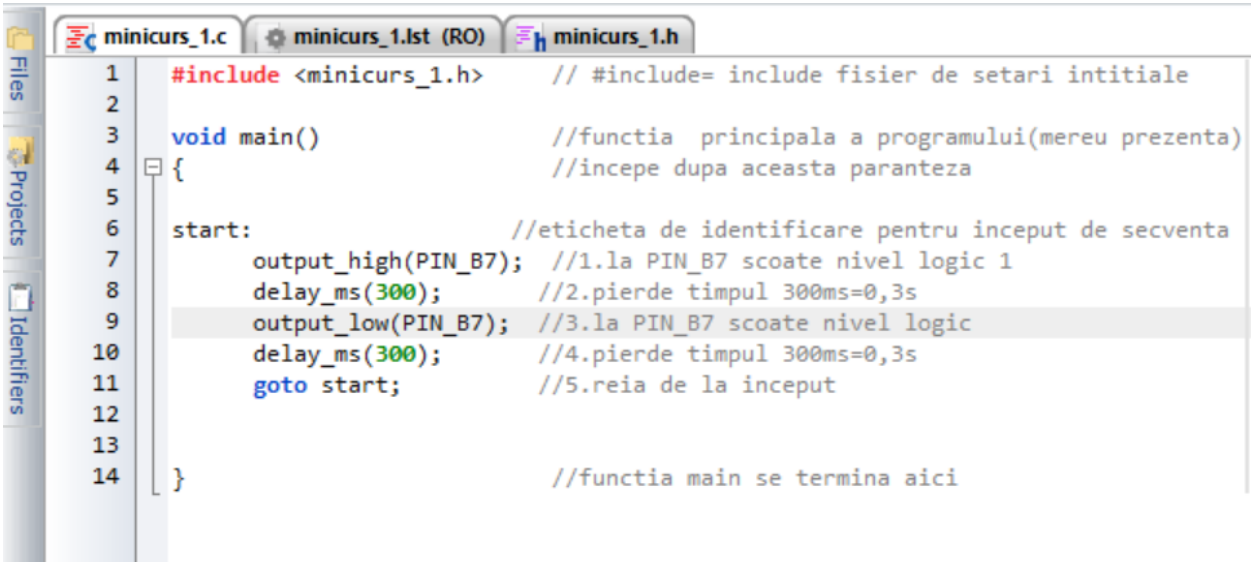
NOTA:
-daca sinteti curiosi sa vedeti cum arata programul in limbaj de asamblare apasati meniul
View si C/ASM List

2.2 Clipire LED / Delay/Goto/Etichete
In acest exemplu vom prezenta functia de intirziere (delay), instructiunea Goto si etichetele.

Daca dorim sa aprindem un LED intermitent atunci ne gindim ca :
1. mai intii il vom aprinde
2. il vom lasa aprins un timp
3. il vom stinge
4. il vom tine stins un timp
5. vom relua de la inceput

Revenim putin la exemplul anterior si la instructiunea **while**. Desi instructiune **while** este foarte recomandata in zilele noastre (!) as dori sa utilizam o metoda mai veche dar care ne arata foarte precis ce se intimpla in cadrul unui program. Nu doresc sa ne intoarcem la metode invechite dar spun doar atit: dupa compilarea programului se utilizeaza aceeasi cantitate de memorie si cu instructiunea **while** si cu metoda preferata in aceasta prezentare (cu instructiunea **goto**).

Instructiunea pentru a intirzia executia (de fapt pentru a pierde timpul) este **delay.**
delay_ms(1000) intirziere 1000mS =1S
delay_uS(1000) intirziere 1000uS=1mS=0,1S
Nu exista parametru pentru minute si ore. Folositi 60.000ms pentru 1 min. Mai tirziu veti putea utiliza intirzierea de mai multe ori la rind pentru a intirzia mai mult (ore).



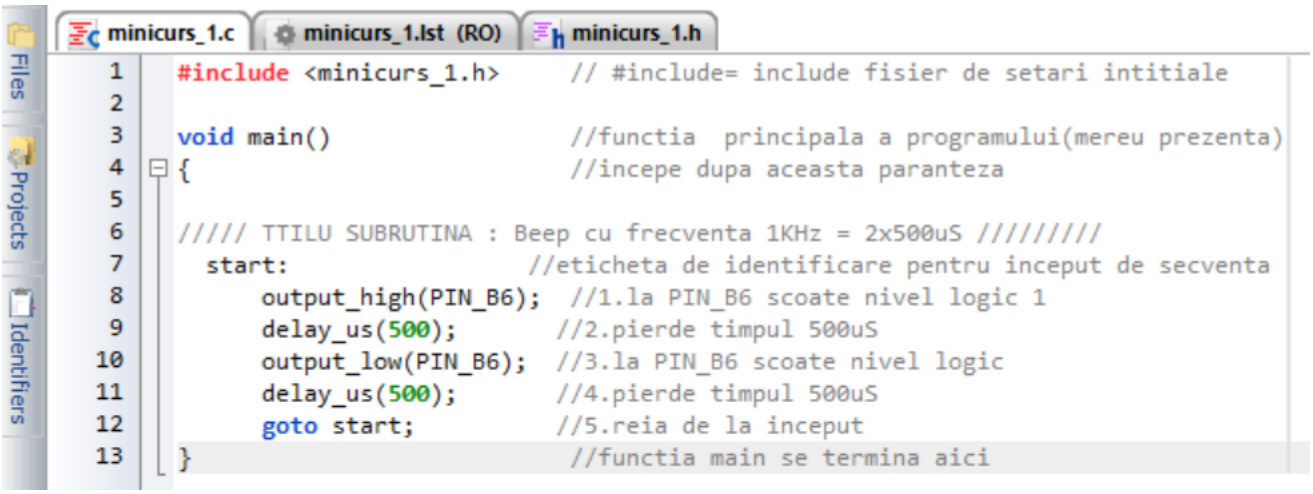
De ce am spus ca este o instructiune care pierde timpul? Fiindca procesorul va executa intirzierea si nimic altceva. Nu va iesi de acolo decit dupa terminarea intirzierii. Uneori aceasta pierdere de timp nu este benefica. Atunci se vor utiliza alte resurse ale procesorului (intreruperi sau alte tehnici)- mai vedem.

Observati ca am sters instructiunea **while** si parantezele si am introdus o eticheta denumita **start** (dupa care am pus doua puncte **:**) iar la sfirsitul programului am scris instructiunea **goto** urmata de eticheta **start**.
In acest fel se intelege mai bine ce face programul. Etichetele nu pot incepe cu cifre. Dati nume sugestive etichetelor pentru a urmari mai usor programul. Programul se va executa intre **start** si **goto start**.

NOTA:
-daca sinteti curiosi sa vedeti cum arata programul in limbaj de asamblare apasati meniul
View si C/ASM List. Observati cit de complicata este in asamblor functia **delay**.

2.3 Aplicatie: Beep

Sunetul este o oscilatie cu o anumita frecventa (audibila, 20-20000Hz). Daca aplicam unui *buzzer* o succesiune de 1 si 0 cu o anumita frecventa acesta va reproduce un sunet. Programul este simplu. Utilizam ce am scris la clipire LED si ajustam putin.




```
1  #include <minicurs_1.h>     // #include= include fisier de setari intitiale
2
3  void main()                 //functia  principala a programului(mereu prezenta)
4  {                           //incepe dupa aceasta paranteza
5
6  ///// TTILU SUBRUTINA : Beep cu frecventa 1KHz = 2x500uS /////////
7    start:                  //eticheta de identificare pentru inceput de secventa
8        output_high(PIN_B6);  //1.la PIN_B6 scoate nivel logic 1
9        delay_us(500);        //2.pierde timpul 500uS
10       output_low(PIN_B6);   //3.la PIN_B6 scoate nivel logic
11       delay_us(500);        //4.pierde timpul 500uS
12       goto start;           //5.reia de la inceput
13 }                           //functia main se termina aici
```

Am schimbat pinul de iesire si durata intirzierii. Aceasta este acum 500uS. Intr-un ciclu avem doua intirzieri de 500uS deci in total 1mS adica 1KHz. Valoarea va fi aproximativa (mai mare) deoarece se mai lungeste cu executia instructiunilor **output** si **goto**. Pentru precizie se pot ajusta intirzierile (499uS, etc). instructiunea **goto** dureaza doua cicluri. Un ciclu la 8MHz are frecventa 8/4=2MHz deci T=1/2MHz=0,5uS durata. Celelalte instructiuni dureaza un ciclu de 0,5uS.

NOTA: sunetul va fi continuu, pina se decupleaza alimentarea.

Modificari:
-Incercati sa aprindeti LED-ul permanent si concomitent sa generati sunet
-Incercati sa faceti LED-ul sa clipeasca si concomitent sa generati sunet

2.5 Iesire pe PORT

In anumite cazuri se prefera scrierea tuturor bitilor unui port.
Instructiunea este (de exemplu pentru PORT B):

`output_B(0b11111111);` fiecare 1 actioneaza iesirea corespunzatoare (primul in dreapta este B0)
daca se scrie 0 atunci iesirea se face 0