```
//=========================================================================
//
//          FONTOS információk a PROG2 labor uC AVR-C NAGYZH-król
//
//    1.: a PROG2 LABOR órákon elhangzott, hogy mindenki CSAK a saját,
//          NEPTUN-ban feltüntetett laborfoglalkozás időpontjában írhatja
//          meg a NAGYZH-t. Kérem, hogy ezt vegyék figyelembe!
//
//    2.: a NAGYZH-n csak a mi általunk adott üres papír, a számítógép
//          'asztalán' általunk feltett segédletek (ATmega adatlap, T-bird
//          porttábla) és a WINDOWS-os számológép használható.
//
//    3.: az AVR-C NAGYZH-n 4 db. (első 4) és 1 db. (5.) különálló feladat
//          lesz!
//          A 4 db. különálló feladatok megoldását 4 db. különálló
//          függvényekben kell megírni.
//     Ezért;
//      2 db. függvény helyes megírása és közvetlen meghívása; elégséges,
//      3 db. függvény helyes megírása és közvetlen meghívása; közepes,
//      4 db. függvény helyes megírása és közvetlen meghívása; jó
//     érdemjeggyel honorálható.
//
//     Amennyiben egy hallgató az 5. feladatot is működőképesen megírja
//    (menükezelés klaviatúrával), úgy az érdemjegye eggyel nő, azaz pl.:
//      2 db. függvény helyes megírása és klaviatúra menüs meghívása;
//            közepes,
//      3 db. függvény helyes megírása és klaviatúra menüs meghívása; jó,
//      4 db. függvény helyes megírása és klaviatúra menüs meghívása;jeles.
//    . érdemjeggyel honorálható.
//
//=========================================================================
//
//    AVR-C nagyZH elvi hibajegyzéke:
//
//    Ha az alábbi elvi hibák megtalálhatók az AVR-C programban, az adott
//    feladatrészt vagy akár az egész programot értékelhetetlenné teszik;
//    - ha a FÜGGVÉNY-ek előállításánál nem szerepel külön a DEKLARÁCIÓS és
//      DEFINICIÓS rész,
//    - ha az int main() végén a "return 0;" előtt nincs lezáró végtelen
//      ciklus,
//    - ha a sajátfüggvényben van a végtelen ciklus (nem tér vissza a
//      vezérlés a main()-hez),
//    - ha az ISR (megszakítás) rutinban '_delay_ms()' késleltető
//      használata történt.
//
//    Továbbá;
//    - a globális változók csak ISR (megszakítás) rutin miatt
//      használhatók, függvényben paraméter átadásra viszont a
//      használatuk tilos.
//
//    - a globális változók csak a main() függvényben (változótól vagy
//      függvény visszatérési értékként) VAGY csak az ISR (megszakítás)
//      rutinban kaphatnak értéket.
//
//    Természetesen más hibák miatt is lehetséges értékelhetetlen rész vagy
//    egész program, de, ha a fent felsorolt hibák jelen vannak, akkor a
//    feladatrész vagy az egész feladat biztosan nem értékelhető.
//=========================================================================
```

```
//    AVR-C MINTA NAGYZH
//
//-------------------------------------------------------------------------
//  1. FELADAT:
//    Írjon és hívjon meg egy függvényt a bemenő paraméterek érték szerinti
//    átadásával, amely ciklus segítségével összegzi a decimális [1,25]
//    zárt intervallumban az összes olyan egész értéket, melyek bináris
//    alakjában a szám 2. bitje "1",
//          valamint a 0. bitje "0" értékű.
//
//    A függvény bemenő paraméterei az előzőekben szereplő intervallum
//    határai legyenek!
//    A függvény adja vissza a hívó 'main()' függvénynek a kiszámított
//    eredményt, amelyet a 'main()'-ben binárisan jelenítsen meg a
//    LED-eken! A nulladik bit az LSB.
//
//    MEGJEGYZÉS: ha már két megjelenítendő eredménye van a LED-eken, akkor
//    a két eredményt kb. 5 másodpercenként, felváltva jelenítse meg a
//    LED-eken.
//
//    Az aktuális feladathoz tartozó LED-ek állapotát írja az alábbi
//    ablakba úgy, hogy a világító LED "1"-nek feleljen meg!
//
//-------------------------------------------------------------------------
//  2. FELADAT:
//    Írjon és hívjon meg egy függvényt a bemenő paraméterek érték szerinti
//    átadásával, melyben egy adott szám prímtényezős felbontásában
//    megszámolja, hogy hány darab 3-as szorzója van.
//    A vizsgálandó érték: 270.
//
//    A függvény bemenő paraméterei az előzőekben szereplő vizsgálandó
//    érték és a prím szorzótényező legyenek.
//    A függvény adja vissza a hívó 'main()' függvénynek a kiszámított
//    eredményt, amelyet a 'main()'-ben binárisan jelenítsen meg a
//    LED-eken! A nulladik bit az LSB.
//
//    MEGJEGYZÉS: ha már két megjelenítendő eredménye van a LED-eken, akkor
//    a két eredményt kb. 5 másodpercenként, felváltva jelenítse meg a
//    LED-eken.
//
//    Az aktuális feladathoz tartozó LED-ek állapotát írja az alábbi
//    ablakba úgy, hogy a világító LED "1"-nek feleljen meg!
//
//-------------------------------------------------------------------------
//  3. FELADAT:
//    Írjon és hívjon meg egy függvényt a bemenő paraméterek érték szerinti
//    átadásával, melyben összegzi a 100-nál kisebb Fibonacci számokat.
//
//    A függvény bemenő paramétere az előzőekben szereplő határérték
//    legyen!
//    A függvény adja vissza a hívó 'main()' függvénynek a kiszámított
//    eredményt, amelyet a 'main()'-ben decimális formátumban
//    megszakítással jelenítsen meg a 7-szegmenses kijelzőn!
//
//    MEGJEGYZÉS: ha már két megjelenítendő eredménye van a 7-szegmenses
//    kijelzőn, akkor a két eredményt kb. 5 másodpercenként, felváltva
//    megszakítással jelenítse meg a 7-szegmenses kijelzőn.
//
//    Az aktuális feladathoz tartozó, 7-szegmenses kijelzőn megjelenő
//    értéket írja az alábbi ablakba!
```

```
//-----------------------------------------------------------------------
//  4. FELADAT (csak szemléltető példa gyanánt):
//    Írjon és hívjon meg egy függvényt a bemenő paraméterek érték szerinti
//    átadásával, melyben összegzi a decimális 1234 és decimális 1111
//     értékeket!
//
//    A függvény bemenő paraméterei az előzőekben szereplő két érték
//    legyen!
//    A függvény adja vissza a hívó 'main()' függvénynek a kiszámított
//    eredményt, amelyet a 'main()'-ben decimális formátumban
//    megszakítással jelenítsen meg a 7-szegmenses kijelzőn!
//
//    MEGJEGYZÉS: ha már két megjelenítendő eredménye van a 7-szegmenses
//    kijelzőn, akkor a két eredményt kb. 5 másodpercenként, felváltva
//    megszakítással jelenítse meg a 7-szegmenses kijelzőn.
//
//    Az aktuális feladathoz tartozó, 7-szegmenses kijelzőn megjelenő
//    értéket írja az alábbi ablakba!
//
//-----------------------------------------------------------------------
//  5. FELADAT:
//    Ennek a feladatnak csak akkor állhat neki, ha az előző feladatokból
//    legalább kettő feltehetően helyes eredménye (zöld pipája) van.
//
//    A klaviatúra 1. billentyűjének lenyomása idejére jelenjen meg
//    a LED-eken az 1. feladat eredménye!
//
//    A klaviatúra 2. billentyűjének lenyomása idejére jelenjen meg
//    a LED-eken az 2. feladat eredménye!
//
//    A klaviatúra 3. billentyűjének lenyomása idejére jelenjen meg
//    a 7-szegmenses kijelzőn a 3. feladat eredménye!
//
//    A klaviatúra 4. billentyűjének lenyomása idejére jelenjen meg
//    a 7-szegmenses kijelzőn a 4. feladat eredménye!
//
//    Ha nem nyomunk egy billentyűzetet sem, akkor a LED-ek ne
//    világítsanak, és a 7-szegmenses kijelző mindegyik helyiértékén NULLÁT
//    mutasson!
//
//=======================================================================
//
//    FONTOS MEGJEGYZÉSEK:
//
//    A 'signed' és 'unsigned' típusmódósítók mindenkori használata azért
//    fontos, hogy a program a fejlesztői környezet beállításától
//    függetlenül mindig ugyanazt az eredményt szolgáltassa.
//
//    A megszakítások miatt deklarált globális változók előtti 'volatile'
//    típusmódósítók mindenkori használata azért fontos, hogy a program a
//    fejlesztői környezet (optimalizált/nem optimalizált) beállításától
//    függetlenül jól működjön.
//
//=======================================================================
```

```
// Megoldás az ELÉGSÉGES érdemjegyért; a hallgató úgy dönthet, hogy
// az 1. feladatot és a 2. feladatot oldja meg, akkor az 1. és a 2.
// feladat eredményeit felváltva, kb. 5 másodperces kijelzéssel kell
// megjelenítenie a LED-eken!
// Megoldási példa:

#include <avr/io.h>
#define  F_CPU  8000000UL          //  8 MHz
// #define  F_CPU 16000000UL       // 16 MHz
#include <avr/delay.h>

signed int bit (signed int kf, signed int vf); // 1. feladat
signed int prim(signed int af, signed int bf); // 2. feladat

int main()
{
      DDRB=DDRD=0xF0;
      PORTB=PORTD=0;

      signed int bita=1, bitb=25, bitv=0; // 1. feladat változói
      signed int pra=270, prb=3, prv=0;   // 2. feladat változói

      bitv = bit (bita , bitb);
      prv  = prim(pra  , prb);

      while(1)
      {
            PORTD = bitv;           // 1. feladat eredmény kiíratás a LED-re
            PORTB = bitv << 4;
            _delay_ms(5000);

            PORTD = prv;            // 2. feladat eredmény kiíratás a LED-re
            PORTB = prv << 4;
            _delay_ms(5000);
      }
while(1){};
return 0;
}

// ==== Függvények ================================================

signed int bit (signed int kf, signed int vf)  // 1. feladat függvénye
{
      signed int osszegf=0, i=0;

      for( i=kf ; i<=vf ; i++)
      {
            if( (i & 0b0101)==0b00100 )
            {
                  osszegf=osszegf+i;
            }
      }

return osszegf;
}
```

```
signed int prim( signed int af, signed int bf) // 2. feladat függvénye
{
	signed int dbf=0;

	while ( (af % bf) == 0 )
	{
		af = af / bf;
		dbf++;
	}

return dbf;
}
```

```
// Megoldás az ELÉGSÉGES érdemjegyért; a hallgató úgy dönthet, hogy
// az 1. feladatot és a 3. feladatot oldja meg, akkor az 1. feladat
// eredményét a LED-eken és a 3. feladat eredményét a 7-szegmenses kijelzőn
// kell megjelenítenie!
// Megoldási példa:

#include <avr/io.h>
#define  F_CPU  8000000UL          //  8 MHz
// #define  F_CPU 16000000UL       // 16 MHz
#include <avr/delay.h>
#include <avr/interrupt.h>

signed int bit ( signed int kf, signed int vf);      // 1. feladat
signed int sorozat (signed int hatarf);              // 3. feladat

volatile signed int itszam = 0 , itcounter = 0 ;

int main()
{
      DDRB=DDRD=0xF0;
      DDRA=0xFF;
      PORTA=PORTB=PORTD=0;

      signed int bita=1, bitb=25, bitv=0; // 1. feladat változói
      signed int hatar=100, sorozatv=0;   // 3. feladat változói

      bitv = bit (bita , bitb);
      sorozatv = sorozat (hatar);

      TCCR0=5;
      TIMSK=1;
      sei();

      PORTD  = bitv;          // 1. feladat eredménye a LED-re
      PORTB  = bitv << 4;

      itszam = sorozatv;      // 3. feladat eredménye a 7-szeg kijelzőre

while(1){};
return 0;
}

// ==== Függvények ================================================

signed int bit ( signed int kf, signed int vf)       // 1. feladat
{
      signed int osszegf=0, i=0;

      for( i=kf ; i<=vf ; i++)
      {
            if( (i & 0b0101)==0b00100 )
            {
                  osszegf=osszegf+i;
            }
      }
return osszegf;
}
```

```
signed int sorozat (signed int hatarf)          // 3. feladat
{
      signed int sum=0;
      signed int f0=0, f1=1, fx=0;

      sum = sum + f0 ;

      while( fx < hatarf )
      {
            sum = sum + f1 ;
            fx  = f0 + f1 ;
            f0  = f1 ;
            f1  = fx ;
      }

return sum;
}

// ==== IT0 megszakítás ===================================
ISR(TIMER0_OVF_vect)
{
      itcounter++;

      switch(itcounter)
      {
            case 1: PORTA = 0x80+((itszam/1)%10); break;
            case 2: PORTA = 0x90+((itszam/10)%10); break;
            case 3: PORTA = 0xA0+((itszam/100)%10); break;
            case 4: PORTA = 0xB0+((itszam/1000)%10); itcounter=0; break;
      }
}
```

```
// Megoldás a KÖZEPES érdemjegyért; a hallgató úgy dönthet, hogy
// az 1. feladatot és a 2. feladatot oldja meg, és továbbá megvalósítja a
// KLAVIATÚRA kezelést is úgy, hogy az egyes feladatok eredményei a
// megfelelő klaviatúra billentyűjének a megnyomására jelenjenek meg!
// Megoldási példa:


#include <avr/io.h>
#define  F_CPU  8000000UL          //  8 MHz
// #define  F_CPU 16000000UL       // 16 MHz
#include <avr/delay.h>

signed int bit (signed int kf, signed int vf);       // 1. feladat
signed int prim(signed int af, signed int bf);       // 2. feladat

int main()
{
      DDRB =DDRD=0xF0;
      PORTB=PORTD=0;

      DDRC =0b01111000;
      PORTC=0;
      signed int olvas=0;

      signed int bita=1, bitb=25, bitv=0;      // 1. feladat változói
      signed int pra=270, prb=3, prv=0;        // 2. feladat változói

      bitv = bit (bita , bitb);
      prv  = prim(pra  , prb);

      while(1)
      {
            PORTC = 0b1000;
            _delay_ms(10);
            olvas = PINC;
            olvas = (~olvas) & 0b111;

            switch (olvas)
            {
                  case 1: PORTD=bitv; PORTB=bitv<<4; break; // 1. feladat
                  case 2: PORTD=prv ; PORTB=prv<<4; break;  // 2. feladat
                  default: PORTD=PORTB=0; break;     // különben nincs a
                                                     // LED-eken kijelzés
            }
      }

while(1){};
return 0;
}
```

```
// ==== Függvények ================================================

signed int bit (signed int kf, signed int vf)        // 1. feladat
{
      signed int osszegf=0, i=0;

      for( i=kf ; i<=vf ; i++)
      {
            if( (i & 0b0101)==0b00100 )
            {
                  osszegf=osszegf+i;
            }
      }
return osszegf;
}


signed int prim(signed int af, signed int bf)        // 2. feladat
{
      signed int dbf=0;

      while ( (af % bf) == 0 )
      {
            af = af / bf;
            dbf++;
      }

return dbf;
}
```

```
// Megoldás a KÖZEPES érdemjegyért; a hallgató úgy dönthet, hogy
// az 1. feladatot, a 2. feladatot, és a 3. feladatot oldja meg, akkor
// az 1. és a 2. feladat eredményeit felváltva, kb. 5 másodperces
// kijelzéssel kell megjelenítenie a LED-eken, továbbá a 3. feladat
// eredményét a 7-szegmenses kijelzőn kell megjelenítenie!
// Megoldási példa:


#include <avr/io.h>
#define F_CPU  8000000UL            //  8 MHz
// #define F_CPU 16000000UL         // 16 MHz
#include <avr/delay.h>
#include <avr/interrupt.h>

signed int bit (signed int kf, signed int vf); // 1. feladat
signed int prim(signed int af, signed int bf); // 2. feladat
signed int sorozat (signed int hatarf);

volatile signed int itszam = 0 , itcounter = 0 ;

int main()
{
      DDRB=DDRD=0xF0;
      DDRA=0xFF;
      PORTA=PORTB=PORTD=0;

      signed int bita=1, bitb=25, bitv=0; // 1. feladat változói
      signed int pra=270, prb=3, prv=0;   // 2. feladat változói
      signed int hatar=100, sorozatv=0;   // 3. feladat változói

      bitv = bit (bita , bitb);
      prv  = prim(pra  , prb);
      sorozatv = sorozat (hatar);

      TCCR0=5;
      TIMSK=1;
      sei();

      itszam = sorozatv;      // 3. feladat eredménye a 7-szeg kijelzőre

      while(1)
      {
            PORTD  = bitv;          // 1. feladat eredménye a LED-re
            PORTB  = bitv << 4;
            _delay_ms(5000);

            PORTD  = prv;           // 2. feladat eredménye a LED-re
            PORTB  = prv << 4;
            _delay_ms(5000);
      }
while(1){};
return 0;
}
```

```
// ==== Függvények ================================================

signed int bit (signed int kf, signed int vf)        // 1. feladat
{
      signed int osszegf=0, i=0;

      for( i=kf ; i<=vf ; i++)
      {
            if( (i & 0b0101)==0b00100 )
            {
                  osszegf=osszegf+i;
            }
      }
return osszegf;
}

signed int prim(signed int af, signed int bf)        // 2. feladat
{
      signed int dbf=0;

      while ( (af % bf) == 0 )
      {
            af = af / bf;
            dbf++;
      }
return dbf;
}

signed int sorozat (signed int hatarf)               // 3. feladat
{
      signed int sum=0;
      signed int f0=0, f1=1, fx=0;

      sum = sum + f0 ;

      while( fx < hatarf )
      {
            sum = sum + f1 ;
            fx  = f0 + f1 ;
            f0  = f1 ;
            f1  = fx ;
      }

return sum;
}


// ==== IT0 megszakítás ===================================

ISR(TIMER0_OVF_vect)
{
      itcounter++;

      switch(itcounter)
      {
            case 1: PORTA = 0x80+((itszam/1)%10); break;
            case 2: PORTA = 0x90+((itszam/10)%10); break;
            case 3: PORTA = 0xA0+((itszam/100)%10); break;
            case 4: PORTA = 0xB0+((itszam/1000)%10); itcounter=0; break;
      }
}
```

```
// Megoldás a JÓ érdemjegyért; a hallgató úgy dönthet, hogy
// az 1. feladatot, a 2. feladatot, a 3. feladatot, és a 4. feladatot
// oldja meg, akkor az 1. és a 2. feladat eredményeit felváltva,
// kb. 5 másodperces kijelzéssel kell megjelenítenie a LED-eken,
// továbbá a 3. feladat és 4. feladat eredményeit felváltva,
// kb. 5 másodperces kijelzéssel kell megjelenítenie a 7-szegmenses
// kijelzőn!
// Megoldási példa:


#include <avr/io.h>
#define  F_CPU  8000000UL          //  8 MHz
// #define  F_CPU 16000000UL       // 16 MHz
#include <avr/delay.h>
#include <avr/interrupt.h>

signed int bit (signed int kf, signed int vf);       // 1. feladat
signed int prim(signed int af, signed int bf);       // 2. feladat
signed int sorozat (signed int hatarf);              // 3. feladat
signed int ossze (signed int xf, signed int yf);     // 4. feladat

volatile signed int itszam = 0 , itcounter = 0 ;


int main()
{
      DDRB=DDRD=0xF0;
      DDRA=0xFF;
      PORTA=PORTB=PORTD=0;

      signed int bita=1, bitb=25, bitv=0; // 1. feladat változói
      signed int pra=270, prb=3, prv=0;   // 2. feladat változói
      signed int hatar=100, sorozatv=0;   // 3. feladat változói
      signed int x=1234, y = 1111, zv=0;  // 4. feladat változói

      bitv = bit (bita , bitb);
      prv  = prim(pra  , prb);
      sorozatv = sorozat (hatar);
      zv   = ossze(x , y);

      TCCR0=5;
      TIMSK=1;
      sei();

      while(1)
      {
            PORTD  = bitv;       // 1. feladat eredménye a LED-re
            PORTB  = bitv << 4;
            itszam = sorozatv;   // 3. feladat eredménye a 7-szeg kijelzőre
            _delay_ms(5000);

            PORTD  = prv;        // 2. feladat eredménye a LED-re
            PORTB  = prv << 4;
            itszam = zv;         // 4. feladat eredménye a 7-szeg kijelzőre
            _delay_ms(5000);
      }
while(1){};
return 0;
}
```

```
// ==== Függvények ================================================

signed int bit (signed int kf, signed int vf)        // 1. feladat
{
      signed int osszegf=0, i=0;

      for( i=kf ; i<=vf ; i++)
      {
            if( (i & 0b0101)==0b00100 )
            {
                  osszegf=osszegf+i;
            }
      }
return osszegf;
}


signed int prim(signed int af, signed int bf)        // 2. feladat
{
      signed int dbf=0;
      while ( (af % bf) == 0 )
      {
            af = af / bf;
            dbf++;
      }
return dbf;
}


signed int sorozat (signed int hatarf)          // 3. feladat
{
      signed int sum=0;
      signed int f0=0, f1=1, fx=0;

      sum = sum + f0 ;

      while( fx < hatarf)
      {
            sum = sum + f1 ;
            fx  = f0 + f1 ;
            f0  = f1 ;
            f1  = fx ;
      }

return sum;
}


signed int ossze (signed int xf, signed int yf)      // 4. feladat
{
      signed int sum = 0;
      sum = xf + yf;
return sum;
}
```

```
// ==== IT0 megszakítás ===================================
ISR(TIMER0_OVF_vect)
{
	itcounter++;

	switch(itcounter)
	{
		case 1: PORTA = 0x80+((itszam/1)%10); break;
		case 2: PORTA = 0x90+((itszam/10)%10); break;
		case 3: PORTA = 0xA0+((itszam/100)%10); break;
		case 4: PORTA = 0xB0+((itszam/1000)%10); itcounter=0; break;
	}
}
```

```
// Megoldás a JÓ érdemjegyért; a hallgató úgy dönthet, hogy
// az 1. feladatot, a 2. feladatot, a 3. feladatot oldja meg, és továbbá
// megvalósítja a KLAVIATÚRA kezelést is úgy, hogy az egyes feladatok
// eredményei a megfelelő klaviatúra billentyűjének a megnyomására
// jelenjenek meg!
// Megoldási példa:

#include <avr/io.h>
#define  F_CPU  8000000UL          //  8 MHz
// #define  F_CPU 16000000UL       // 16 MHz
#include <avr/delay.h>
#include <avr/interrupt.h>

signed int bit (signed int kf, signed int vf);       // 1. feladat
signed int prim(signed int af, signed int bf);       // 2. feladat
signed int sorozat (signed int hatarf);              // 3. feladat

volatile signed int itszam = 0 , itcounter = 0 ;

int main()
{
      DDRB =DDRD=0xF0;
      DDRA =0xFF;
      PORTA=PORTB=PORTD=0;

      DDRC =0b01111000;
      PORTC=0;
      signed int olvas=0;

      signed int bita=1, bitb=25, bitv=0;      // 1. feladat változói
      signed int pra=270, prb=3, prv=0;        // 2. feladat változói
      signed int hatar=100, sorozatv=0;        // 3. feladat változói


      bitv = bit (bita , bitb);                // 1. feladat
      prv  = prim(pra  , prb);                 // 2. feladat
      sorozatv = sorozat (hatar);              // 3. feladat

      TCCR0=5;
      TIMSK=1;
      sei();

      while(1)
      {
            PORTC = 0b1000;
            _delay_ms(10);
            olvas = PINC;
            olvas = (~olvas) & 0b111;

            switch (olvas)
            {
                  case 1: PORTD=bitv; PORTB=bitv<<4; break;// 1. feladat
                  case 2: PORTD=prv ; PORTB=prv<<4; break; // 2. feladat
                  case 4: itszam=sorozatv; break;          // 3. feladat
                  default: PORTD=PORTB=itszam=0; break;
            }
      }

while(1){};
return 0;
}
```

```
// ==== Függvények ================================================

signed int bit (signed int kf, signed int vf)        // 1. feladat
{
      signed int osszegf=0, i=0;

      for( i=kf ; i<=vf ; i++)
      {
            if( (i & 0b0101)==0b00100 )
            {
                  osszegf=osszegf+i;
            }
      }
return osszegf;
}


signed int prim(signed int af, signed int bf)        // 2. feladat
{
      signed int dbf=0;

      while ( (af % bf) == 0 )
      {
            af = af / bf;
            dbf++;
      }

return dbf;
}

signed int sorozat (signed int hatarf)               // 3. feladat
{
      signed int sum=0;
      signed int f0=0, f1=1, fx=0;

      sum = sum + f0 ;

      while( fx < hatarf )
      {
            sum = sum + f1 ;
            fx  = f0 + f1 ;
            f0  = f1 ;
            f1  = fx ;
      }
return sum;
}

// ==== IT0 megszakítás ===================================
ISR(TIMER0_OVF_vect)
{
      itcounter++;

      switch(itcounter)
      {
            case 1: PORTA = 0x80+((itszam/1)%10); break;
            case 2: PORTA = 0x90+((itszam/10)%10); break;
            case 3: PORTA = 0xA0+((itszam/100)%10); break;
            case 4: PORTA = 0xB0+((itszam/1000)%10); itcounter=0; break;
      }
}
```

```
// Megoldás a JELES érdemjegyért; a hallgató úgy dönt, hogy
// az 1. feladatot, a 2. feladatot, a 3. feladatot, és a 4. feladatot
// oldja meg, és továbbá megvalósítja a KLAVIATÚRA kezelést is úgy, hogy az
// egyes feladatok eredményei a megfelelő klaviatúra billentyűjének a
// megnyomására jelenjenek meg!
// Megoldási példa:


#include <avr/io.h>
#define F_CPU  8000000UL           //  8 MHz
// #define F_CPU 16000000UL        // 16 MHz
#include <avr/delay.h>
#include <avr/interrupt.h>


signed int bit (signed int kf, signed int vf);       // 1. feladat
signed int prim(signed int af, signed int bf);       // 2. feladat
signed int sorozat (signed int hatarf);              // 3. feladat
signed int ossze (signed int xf, signed int yf);     // 4. feladat


volatile signed int itszam = 0 , itcounter = 0 ;


int main()
{
      DDRB =DDRD=0xF0;
      DDRA =0xFF;
      PORTA=PORTB=PORTD=0;

      DDRC =0b01111000;            // mind a 4 sor
      PORTC=0;
      signed int olvas=0, seged=0, sor=0b10000;  // sor: 2. sor meghajtása

      signed int bita=1, bitb=25, bitv=0; // 1. feladat változói
      signed int pra=270, prb=3, prv=0;   // 2. feladat változói
      signed int hatar=100, sorozatv=0;   // 3. feladat változói
      signed int x=1234, y = 1111, zv=0;  // 4. feladat változói

      bitv = bit (bita , bitb);
      prv  = prim(pra  , prb);
      sorozatv = sorozat (hatar);
      zv   = ossze(x , y);

      TCCR0=5;
      TIMSK=1;
      sei();
```

```
	while(1)
	{
		sor   = sor ^ 0b11000;  // 1. és 2. sort hajtjuk meg felváltva
		PORTC = sor;
		_delay_ms(10);

		olvas = PINC;
		olvas = (~olvas) & 0b111;
		olvas = sor+olvas;

		if ( olvas== 0x09 && seged==0 )
		{ seged=1; PORTD=bitv; PORTB=bitv<<4;}
                                        // 1. feladat eredménye a LED-re

		if ( olvas== 0x0A && seged==0 )
		{ seged=1; PORTD=prv;  PORTB=prv <<4;}
                                        // 2. feladat eredménye a LED-re

		if ( olvas== 0x0C && seged==0 )
		{ seged=1; itszam=sorozatv;}
                                        // 3. feladat eredménye a 7szeg-re

		if ( olvas== 0x11 && seged==0 )
		{ seged=2; itszam=zv;}          // 4. feladat eredménye a 7szeg-re


		if ( ( olvas== 0x08 && seged==1 ) ||
		      // ha előzőleg nyomtunk 1. sort, de most már nem nyomjuk
		      // az első sort!
		     ( olvas== 0x10 && seged==2 ) )
		      // ha előzőleg nyomtunk 2. sort, de most már nem nyomjuk
		      // a második sort!
		{     seged=0;    // csak akkor állítjuk vissza a segédváltozót
		                  // 0-ba, HA a megfelelő sorban a lévő
		                  // klaviatúra billentyűjét engedtük el.
		      PORTD=PORTB=0; itszam=0;      // itt 0-ázzuk a kijelzést
		}
	}

while(1){};
return 0;
}

// ==== Függvények ================================================

signed int bit (signed int kf, signed int vf)        // 1. feladat
{
	signed int osszegf=0, i=0;

	for( i=kf ; i<=vf ; i++)
	{
		if( (i & 0b0101)==0b00100 )
		{
			osszegf=osszegf+i;
		}
	}
return osszegf;
}
```

```
signed int prim(signed int af, signed int bf)        // 2. feladat
{
      signed int dbf=0;
      while ( (af % bf) == 0 )
      {
            af = af / bf;
            dbf++;
      }
return dbf;
}


signed int sorozat (signed int hatarf)               // 3. feladat
{
      signed int sum=0;
      signed int f0=0, f1=1, fx=0;

      sum = sum + f0 ;

      while( fx < hatarf )
      {
            sum = sum + f1 ;
            fx  = f0 + f1 ;
            f0  = f1 ;
            f1  = fx ;
      }
return sum;
}


signed int ossze (signed int xf, signed int yf)      // 4. feladat
{
      signed int sum = 0;
      sum = xf + yf;
return sum;
}

// ==== IT0 megszakítás ===================================
ISR(TIMER0_OVF_vect)
{
      itcounter++;

      switch(itcounter)
      {
            case 1: PORTA = 0x80+((itszam/1)%10); break;
            case 2: PORTA = 0x90+((itszam/10)%10); break;
            case 3: PORTA = 0xA0+((itszam/100)%10); break;
            case 4: PORTA = 0xB0+((itszam/1000)%10); itcounter=0; break;
      }
}

//=========================================================================
// FONTOS MEGJEGYZÉSEK:
//
//  - látható, hogy a megszakítás rutinban nincs és nem is szabad (!)
//    _delay_ms() függvényt használni, de a main() főfüggvényben lehetséges
//     a _delay_ms() függvény használata!
//
//  - ha a programot debug-olni szeretnék, akkor a _delay_ms() függvényt a
//    debug-olás idejére kommentezzék ki!
```

```
//---------------------------------------------------------------------------
//
// TOVÁBBI mintapéldák;
//
//  P1. FELADAT:
//    Írjon és hívjon meg egy függvényt a bemenő paraméterek érték szerinti
//    átadásával, amely ciklus segítségével megszámolja a decimális [1,25]
//    zárt intervallumban az összes olyan egész értéket, melyek bináris
//    alakjában a szám 2. bitje "1",
//          valamint a 0. bitje "0" értékű.
//
//    A függvény bemenő paraméterei az előzőekben szereplő intervallum
//    határai legyenek!
//    A függvény adja vissza a hívó 'main()' függvénynek a kiszámított
//    eredményt, amelyet a 'main()'-ben binárisan jelenítsen meg a
//    LED-eken! A nulladik bit az LSB.
//
//    MEGJEGYZÉS: ha már két megjelenítendő eredménye van a LED-eken, akkor
//    a két eredményt kb. 5 másodpercenként, felváltva jelenítse meg a
//    LED-eken.
//
//    Az aktuális feladathoz tartozó LED-ek állapotát írja az alábbi
//    ablakba úgy, hogy a világító LED "1"-nek feleljen meg!
//
//---------------------------------------------------------------------------
//
//  P2. FELADAT:
//    Írjon és hívjon meg egy függvényt a bemenő paraméterek érték szerinti
//    átadásával, amely a decimális 10-től növekvő egész számok közül
//    összegzi az első 5 darab olyan számot,
//    amely 2. bitje "1", valamint a 0. bitje "0" értékű.
//
//    A függvény bemenő paraméterei az előzőekben szereplő kezdőérték és
//    darabszám legyenek!
//    A függvény adja vissza a hívó 'main()' függvénynek a kiszámított
//    eredményt, amelyet a 'main()'-ben binárisan jelenítsen meg a
//    LED-eken! A nulladik bit az LSB.
//
//    MEGJEGYZÉS: ha már két megjelenítendő eredménye van a LED-eken, akkor
//    a két eredményt kb. 5 másodpercenként, felváltva jelenítse meg a
//    LED-eken.
//
//    Az aktuális feladathoz tartozó LED-ek állapotát írja az alábbi
//    ablakba úgy, hogy a világító LED "1"-nek feleljen meg!
//
//---------------------------------------------------------------------------
```

```
#include <avr/io.h>
#define  F_CPU 8000000UL           //  8 MHz
// #define F_CPU 16000000UL        // 16 MHz
#include <avr/delay.h>

signed int p1 (signed int kf,  signed int vf);       // 1. feladat
signed int p2 (signed int kef, signed int dbf);      // 2. feladat

int main()
{
      DDRB=DDRD=0xF0;
      DDRA=0xFF;
      PORTA=PORTB=PORTD=0;

      signed int bita= 1, bitb= 25, p1v= 0;     // 1. feladat változói
      signed int ke  =10, db  =  5, p2v= 0;     // 2. feladat változói

      p1v = p1 (bita, bitb);
      p2v = p2 (ke  , db);

      while(1)
      {
            PORTD = p1v;            // 1. feladat eredmény kiíratás a LED-re
            PORTB = p1v << 4;
            _delay_ms(5000);

            PORTD = p2v;            // 2. feladat eredmény kiíratás a LED-re
            PORTB = p2v << 4;
            _delay_ms(5000);
      }
while(1){};
return 0;
}


// ==== Függvények ================================================

signed int p1 (signed int kf, signed int vf)   // 1. feladat függvénye
{
      signed int dbf=0, i=0;

      for( i=kf ; i<=vf ; i++)
      {
            if( (i & 0b0101)==0b00100 )
            {
                  dbf++;
            }
      }
return dbf;
}
```

```
signed int p2 (signed int kef, signed int dbf) // 2. feladat függvénye
{
      signed int osszegf=0, i=1, vizs_szamf=0;

      vizs_szamf=kef;

      while ( i <= dbf )
      {
            if( (vizs_szamf & 0b0101)==0b00100 )
            {
                  i++;
                  osszegf = osszegf + vizs_szamf;
            }

            vizs_szamf++;
      }
return osszegf;
}



//=========================================================================
//
// FONTOS MEGJEGYZÉSEK:
//
//    A 32 bites CODEBLOCKS fejlesztőkörnyezetben
//    az 'int' típus mérete: 4 BYTE.
//
//    Az AVR-C fejlesztőkörnyezetben az 'int' típus mérete: 2 BYTE, továbbá
//    az AVR-C fejlesztőkörnyezetben a 'long int' típus mérete: 4 BYTE.
//
//    Tehát, ha a CODEBLOCKS-os PROGRAMOZÁS 1 mintapéldákat át szeretnék
//    tenni AVR-C környezetbe, akkor előtte győződjenek meg a helyes
//    változó méretek használatáról!
//
//=========================================================================
```