Programozás C nyelven (6. ELŐADÁS)

Sapientia EMTE

2020-21



ELJÁRÁSOK: `void`-függvények

- Olvassunk be n számot a billentyűzetről, és írassuk ki a négyzeteiket a képernyőre.

```
int main(){
   int n, i, szam;
   scanf("%i", &n);
   for( i=1 ; i <= n ; ++i){
      scanf("%i", &szam);
      printf("%i\n",szam*szam);
   }
   return 0;
}
```

```
3
7
49
11
121
5
25
_
```

ELJÁRÁSOK: `void`-függvények

```
void be_szamsor_ki_negyzetsor(int);
int main(){
    int n;
    scanf("%i", &n);
    be_szamsor_ki_negyzetsor(n);
    return 0;
}
void be_szamsor_ki_negyzetsor(int db){
    int i, szam;
    for( i=1 ; i <= db ; ++i){
        scanf("%i", &szam);
        printf("%i\n",szam*szam);
    }
}
```

```
3
7
49
11
121
5
25
_
```

ISMÉTLÉS: iteratív vs. rekurzív

PÉLDÁNYOK

1. 2. 3. 4. 5.

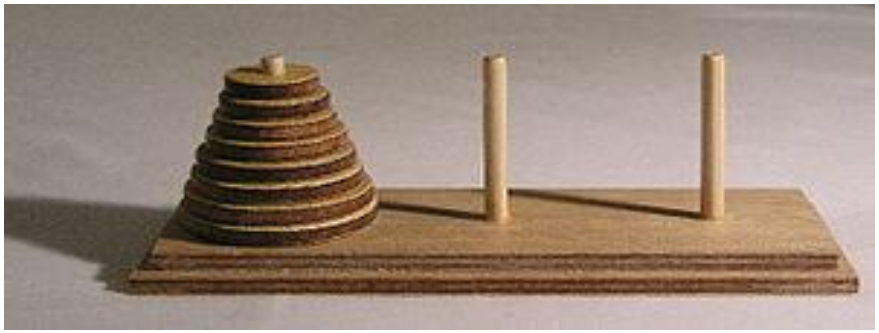
Kiosztott feladatok

ISMÉTLÉS: iteratív vs. rekurzív

- Ha egy függvény azt a feladatot kapja, hogy ismételten végezzen el bizonyos műveleteket, akkor két variáns közül választhat:
  - Ciklus révén felvállalja a sokszori végrehajtást (iteratív módszer);
  - Csak egyszeri végrehajtást vállal fel, és a többit átruházza egy „klónjára” (rekurzív módszer) .
    - átruházza egy „klónjára” ⇔ **meghívja önmagát**

MIÉRT?

Hanoi tornyai: iteratív vs. rekurzív



```
void tohIterative(int num_of_disks, struct Stack *src, struct Stack *aux,
struct Stack *dest){
   int i, total_num_of_moves;
   char s = 'S', d = 'D', a = 'A';
   if (num_of_disks % 2 == 0){
      char temp = d;
      d = a;
      a  = temp;
   }
   total_num_of_moves = pow(
   for (i = num_of_disks; i >= 1; i
      push(src, i);
   for (i = 1; i <= total_num_of_m
      if (i % 3 == 1)
        moveDisksBetweenTwoPoles(src, dest, s, d);
      else if (i % 3 == 2)
        moveDisksBetweenTwoPoles(src, aux, s, a);
      else if (i % 3 == 0)
        moveDisksBetweenTwoPoles(aux, dest, a, d);
   }
}
```

NEHÉZKES

```
void move(int n, int from, int to, int via){
     if (n > 0) {
          move(n - 1, from, via, to);
          Printf("%i >> %i\n", from, to);
          move(n - 1, via, to, from);
     }
}
```

ELEGÁNS

```
void moveDisksBetweenTwoPoles(struct Stack *src,
        struct Stack *dest, char s, char d){
   int pole1TopDisk = pop(src);
   int pole2TopDisk = pop(dest);
   if (pole1TopDisk == INT_MIN){
      push(src, pole2TopDisk);
                                          Disk);
                                          T_MIN){
                                          );
                                          Disk);
                                          e2TopDisk){
                                          Disk);
   }
   else{
      push(dest, pole2TopDisk);
      push(dest, pole1TopDisk);
      moveDisk(s, d, pole1TopDisk);
   }
}
```

Ültess fákat: iteratív

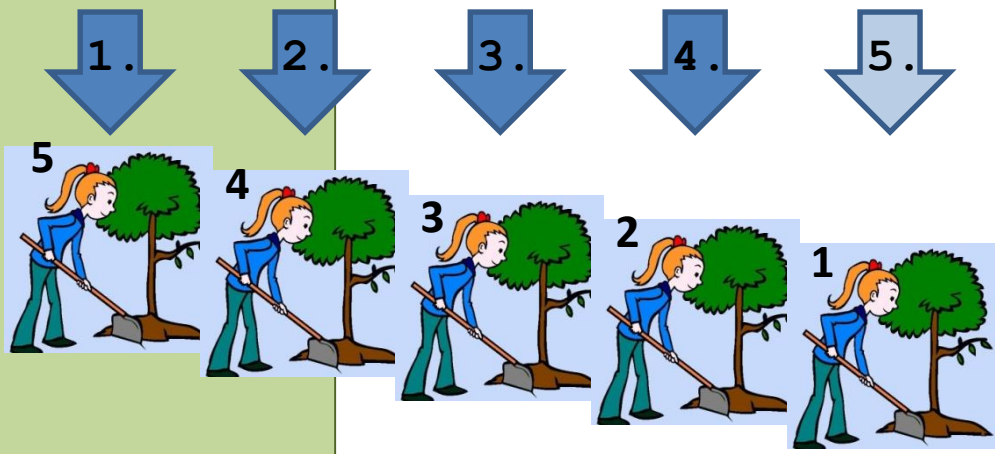
```
void ultess_fakat(int);
int main(){
    int n;
    scanf("%i", &n);
    ultess_fakat(n);
    return 0;
}
void ultess_fakat(int db){
    int i;
    for( i=1 ; i <= db ; ++i){
        <ültess el egy fát>
    }
}
```

Hányszor hívódik meg a függvény

Hány `db/i` –nek történik helyfoglalás

Ültess fákat: rekurzív

```
void ultess_fakat(int);
int main(){
    int n;
    scanf("%i", &n);
    ultess_fakat(n);
    return 0;
}
void ultess_fakat(int db){
    if (db > 1){
        <ültess el egy fát>
        ultess_fakat(db-1);
    }
    else {<ültess el egy fát>}
}
```



Hányszor hívódik meg a függvény

Hány db –nek történik helyfoglalás

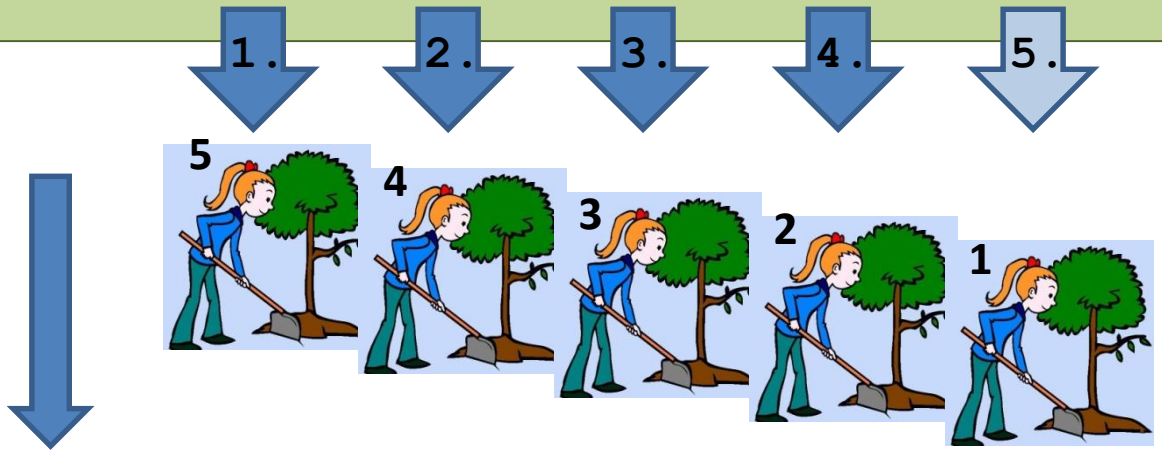
Rekurzív eljárások: SÉMA (1)

```
void R_E(<kapott_feladat_paraméterei>){
    if (<kapott_feladat_NEM_triviális>){
        <old_meg_a_felvállalt_saját_részt>
        R_E(<átruházott_feladat_paraméterei>);
    }
    else {
        <old_meg_a_triviális_feladatot>
    }
}
```

Rekurzív-ág

Triviális-ág

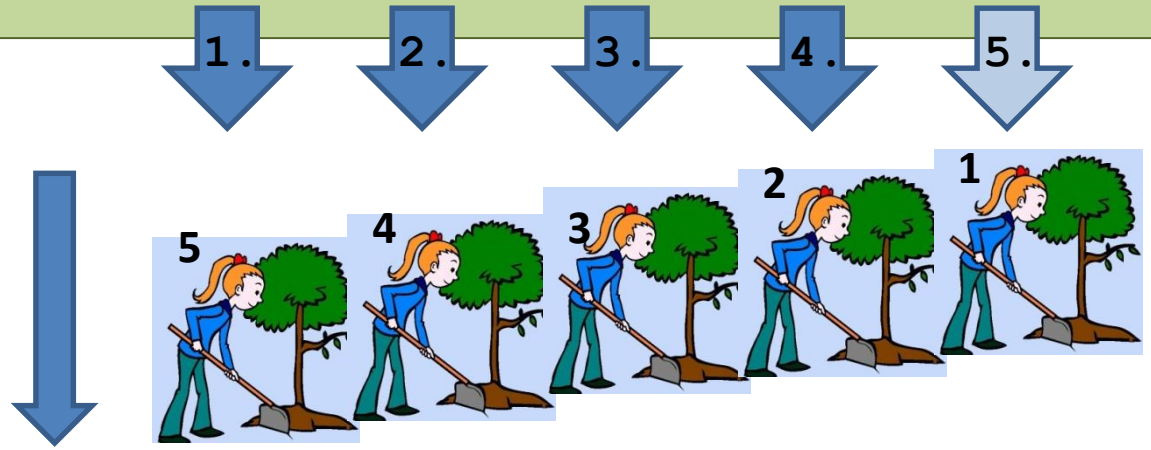
Hasonló, egyszerűbb; konvergál a triviális feladathoz



Melyik példány ültet először fát?

Rekurzív eljárások: SÉMA (2)

```
void R_E(<kapott_feladat_paraméterei>){
    if (<kapott_feladat_NEM_triviális>){
        R_E(<átruházott_feladat_paraméterei>);
        <old_meg_a_felvállalt_saját_részt>
    }
    else {
        <old_meg_a_triviális_feladatot>
    }
}
```



Melyik példány ültet először fát?

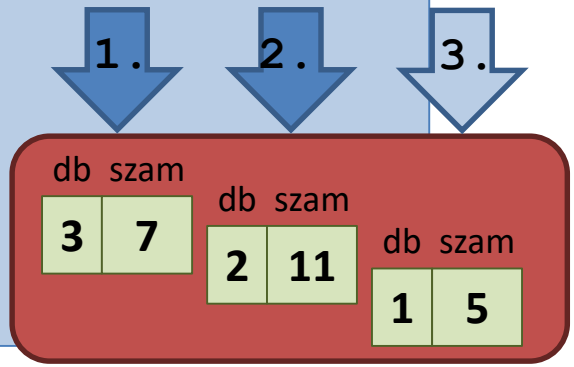
Rekurzív `void`-függvények (1)

```
void REKURZIV_be_szamsor_ki_negyzetsor(int);
int main(){
    int n;
    scanf("%i", &n);
    REKURZIV_be_szamsor_ki_negyzetsor(n);
    return 0;
}
void REKURZIV_be_szamsor_ki_negyzetsor(int db){
    int szam;
    if (db > 1){
        scanf("%i",&szam); printf("%i\n",szam*szam);
        REKURZIV_be_szamsor_ki_negyzetsor(db-1);
    }
    else {
        scanf("%i", &szam);
        printf("%i\n", szam*szam);
    }
}
```

```
3
7
49
11
121
5
25
_
```

Hányszor hívódik meg a függvény

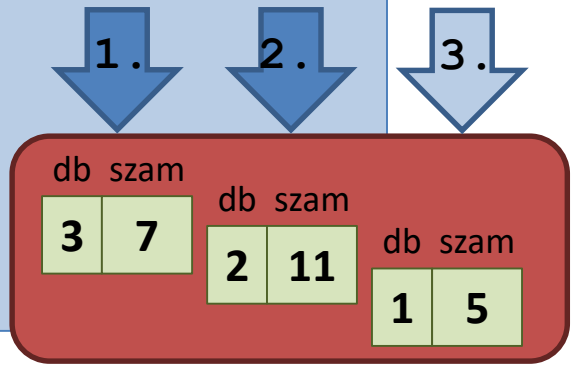
Hány `db`/`szam`–nak történik helyfoglalás



Rekurzív `void`-függvények (2)

```
void REKURZIV_be_szamsor_ki_negyzetsor(int);
int main(){
    int n;
    scanf("%i", &n);
    REKURZIV_be_szamsor_ki_negyzetsor(n);
    return 0;
}
void REKURZIV_be_szamsor_ki_negyzetsor(int db){
    int szam;
    if (db > 1){
        scanf("%i", &szam);
        REKURZIV_be_szamsor_ki_negyzetsor(db-1);
        printf("%i\n", szam*szam);
    }
    else {
        scanf("%i", &szam);
        printf("%i\n", szam*szam);
    }
}
```

```
3
7
11
5
25
121
49
_
```



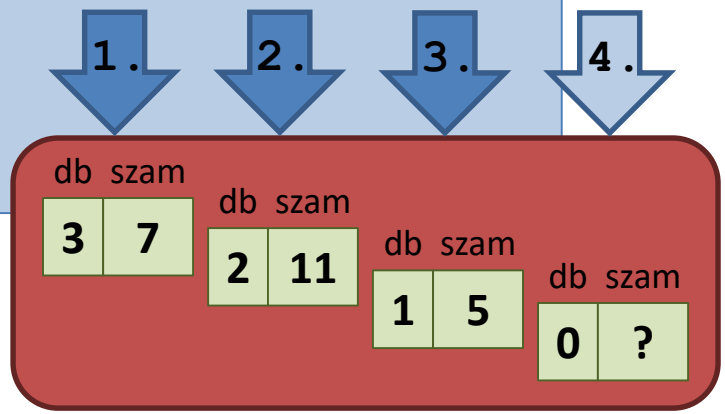
Rekurzív `void`-függvények (3)

```
void REKURZIV_be_szamsor_ki_negyzetsor(int);
int main(){
    int n;
    scanf("%i", &n);
    REKURZIV_be_szamsor_ki_negyzetsor(n);
    return 0;
}
void REKURZIV_be_szamsor_ki_negyzetsor(int db){
    int szam;
    if (db > 0){
        scanf("%i",&szam); printf("%i\n",szam*szam);
        REKURZIV_be_szamsor_ki_negyzetsor(db-1);
    }
    else {;}
}
```

```
3
7
49
11
121
5
25
_
```

Hányszor hívódik meg a függvény

Hány `db`/`szam`–nak történik helyfoglalás



Rekurzív valódi-függvények (1)

Írj rekurzív függvényt, amely visszatéríti a paraméterként kapott természetes szám számjegyei szorzatát!

1. KÉRDÉS:
Hogyan vezethető vissza a feladat hasonló, egyszerűbb részfeladatra?

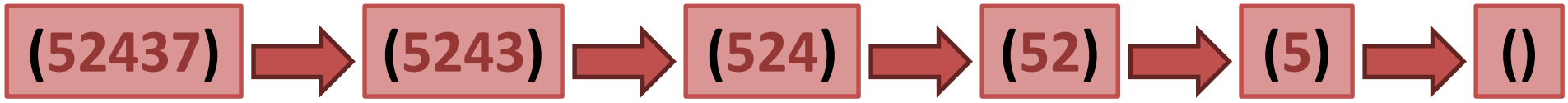
x számjegyszorzata (52437) → **x/10 számjegyszorzata (5243)**

Rekurzív valódi-függvények (2)

Írj rekurzív függvényt, amely visszatéríti a paraméterként kapott természetes szám számjegyei szorzatát!

2. KÉRDÉS:
Mikor tekintem úgy, hogy a feladat triviálissá redukálódott?

Ha elfogyott a szám



Rekurzív valódi-függvények (3)

Írj rekurzív függvényt, amely visszatéríti a paraméterként kapott természetes szám számjegyei szorzatát!

```
int szamjegyszorzat(int x){
  if (x > 0){
    ooo
  }
  else{ ooo }
}
```

Rekurzív-ág

Triviális-ág

Rekurzív valódi-függvények (4)

Írj rekurzív függvényt, amely visszatéríti a paraméterként kapott természetes szám számjegyei szorzatát!

```
int szamjegyszorzat(int x){
  if (x > 0){
    ooo
  }
  else{ return 1; }
}
```

Triviális részfeladat nyilvánvaló megoldásának `return`-elése

Rekurzív valódi-függvények (5)

Írj rekurzív függvényt, amely visszatéríti a paraméterként kapott természetes szám számjegyei szorzatát!

```
int szamjegyszorzat(int x){
  if (x > 0){
    int t = szamjegyszorzat(x/10);
    ooo
  }
  else{ return 1; }
}
```

Rekurzív hívás révén kérem tálcán az átruházott rész megoldását

Rekurzív valódi-függvények (6)

Írj rekurzív függvényt, amely visszatéríti a paraméterként kapott természetes szám számjegyei szorzatát!

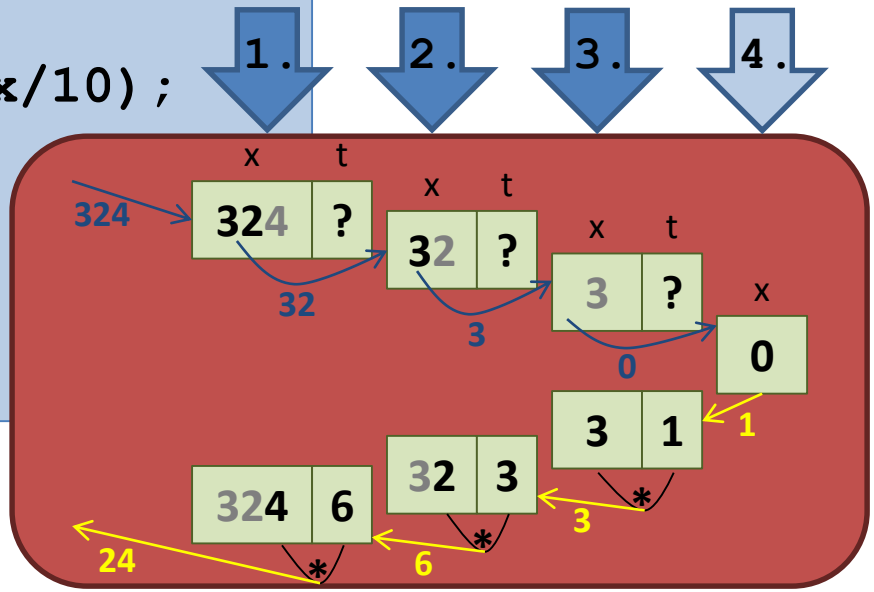
```
int szamjegyszorzat(int x){
  if (x > 0){
    int t = szamjegyszorzat(x/10);
    return t * (x%10);
  }
  else{ return 1; }
}
```

A bevállalt szorzás

Rekurzív valódi-függvények

```
int szamjegyszorzat(int);
int main(){
    int szam;
    scanf("%i", &szam);
    printf("%i", szamjegyszorzat(szam));
    return 0;
}
int szamjegyszorzat(int x){
    if (x > 0){
        int t = szamjegyszorzat(x/10);
        return t * x%10;
    }
    else{ return 1; }
}
```

```
324
24_
```



Összefoglalás (1)

- Minden jelentősebb feladat megoldása részfeladatok ismételt elvégzését feltételezi.
  - Ha ciklus-utasítás révén valósítjuk meg az ismétlést, akkor iteratív implementációról beszélünk.
  - Egy másik lehetőség a rekurzív megvalósítás.
- A rekurzív eljárások/függvények sajátossága, hogy meghívják önmagukat.
  - Olyan, mintha klónoznának magukból további példányokat.
    - Úgy is mondhatnánk, hogy a rekurzivitás mechanizmusa révén, az illető eljárásból/függvényből példányok sora jön létre, amelyek között szétosztódik a megoldandó feladat.

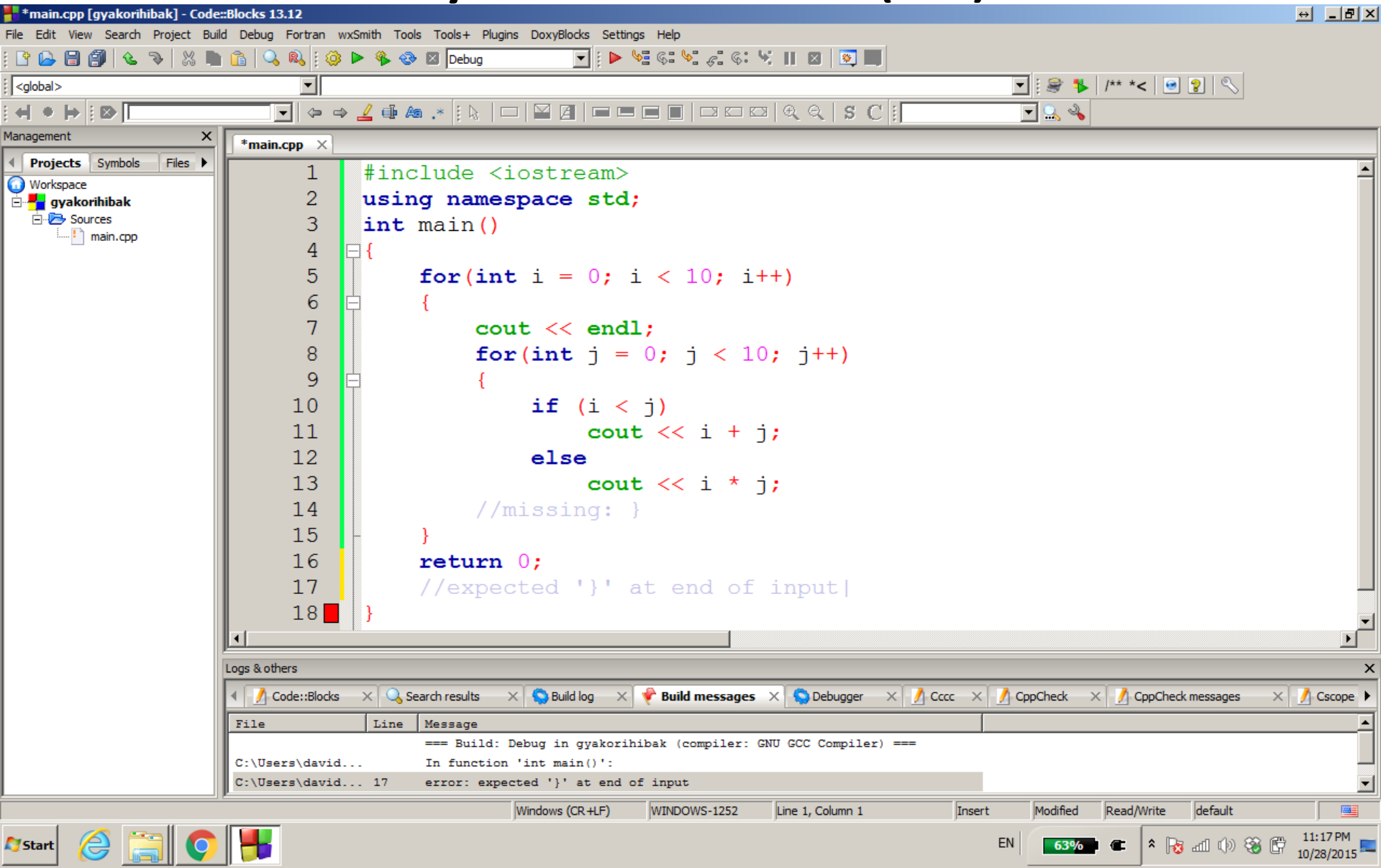
Összefoglalás (2)

- Mindegyik példány bevállal egy adott részt a kapott feladatból, a maradék részfeladatot pedig egy következő példányra ruházza át.
  - Mivel az átruházás egy klónra történik, ezért a kapott feladatnak és az átruházott részfeladatnak hasonlónak kell lennie.
- A rekurzív implementálás kulcskérdése tehát az, hogy miként vezethető vissza a kapott feladat megoldása hasonló, egyre egyszerűbb részfeladatok megoldására.
  - E redukálási folyamat révén végül annyira leegyszerűsödik a feladat, hogy a sorvégi példány egészében be tudja vállalni.

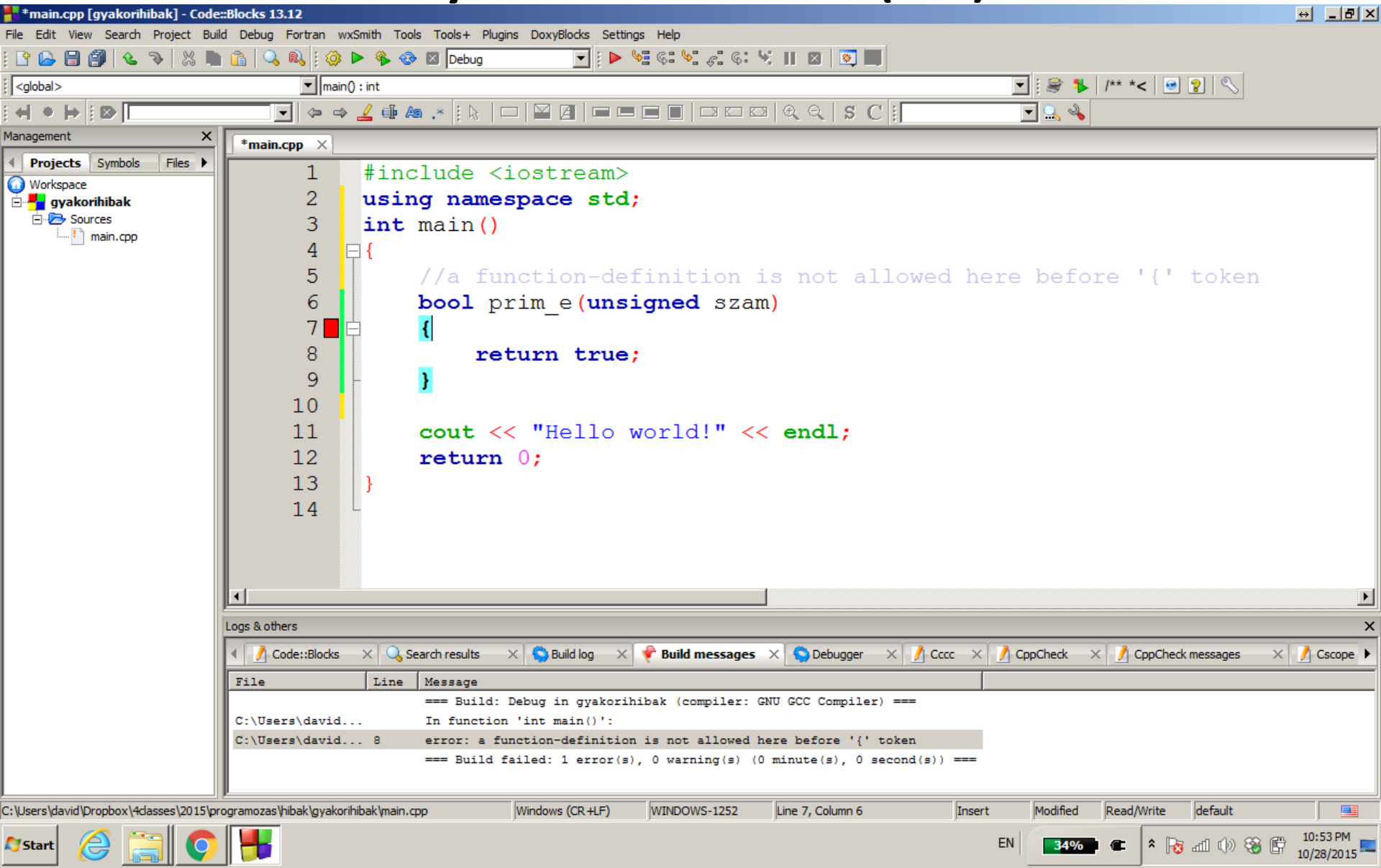
Összefoglalás (3)

- Minden eljárás/függvény-példány hasonló feladatot kap, hogy megoldjon;
- A kapott feladat méretétől függ, hogy a kurrens példány rekurzívan fog viselkedni, vagy bevállalja a teljes megoldást;
  - Rekurzívan viselkedni azt jelenti, hogy önmaga meghívása által, a kapott feladat oroszlánrészét (hasonló, egyszerűbb részfeladatot) átruházza egy következő példányra (a fennmaradt részt bevállalja);
- Egyszerűsített, általános forgatókönyv a kurrens példányra megfogalmazva.

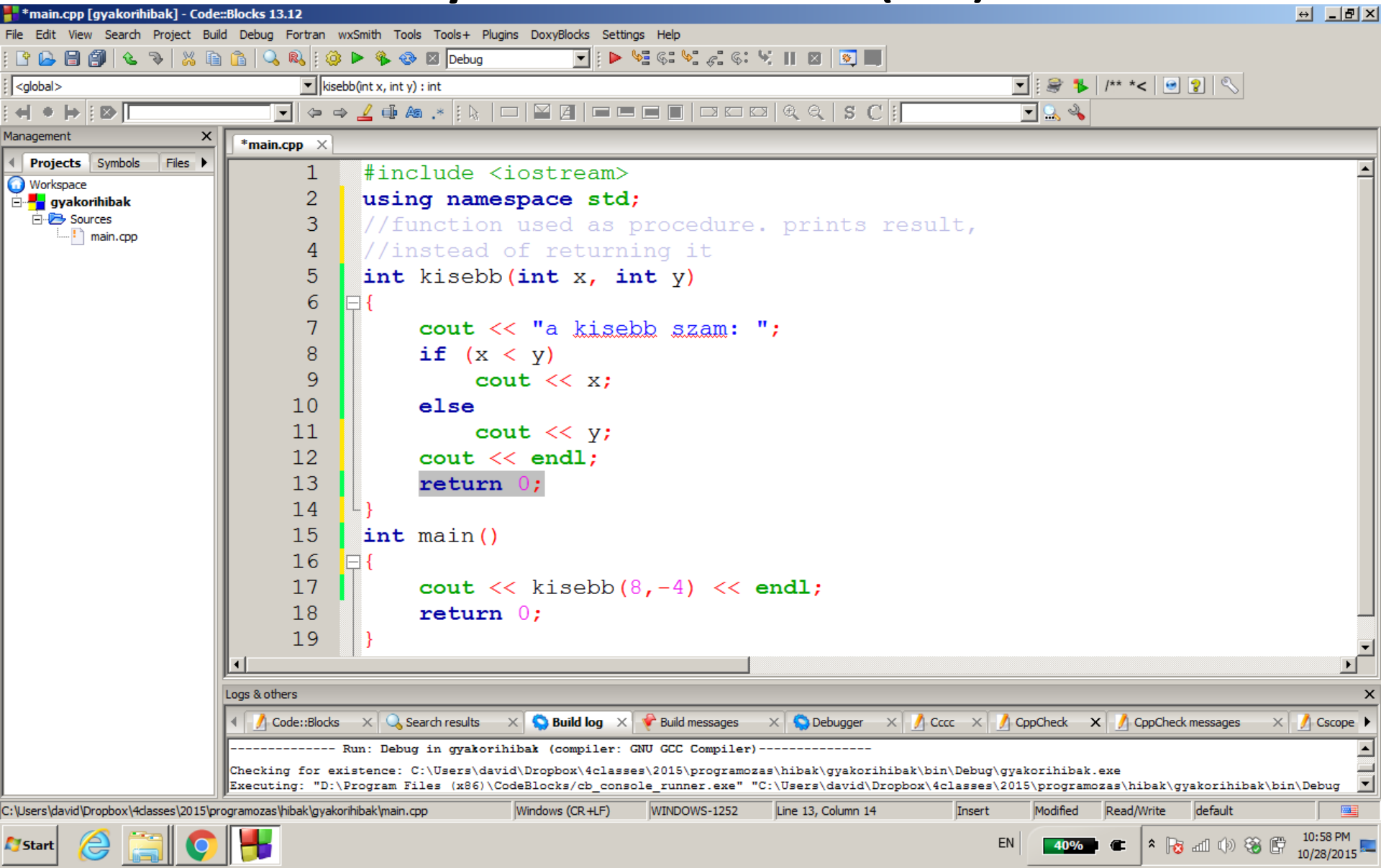
Gyakori hibák (ID)



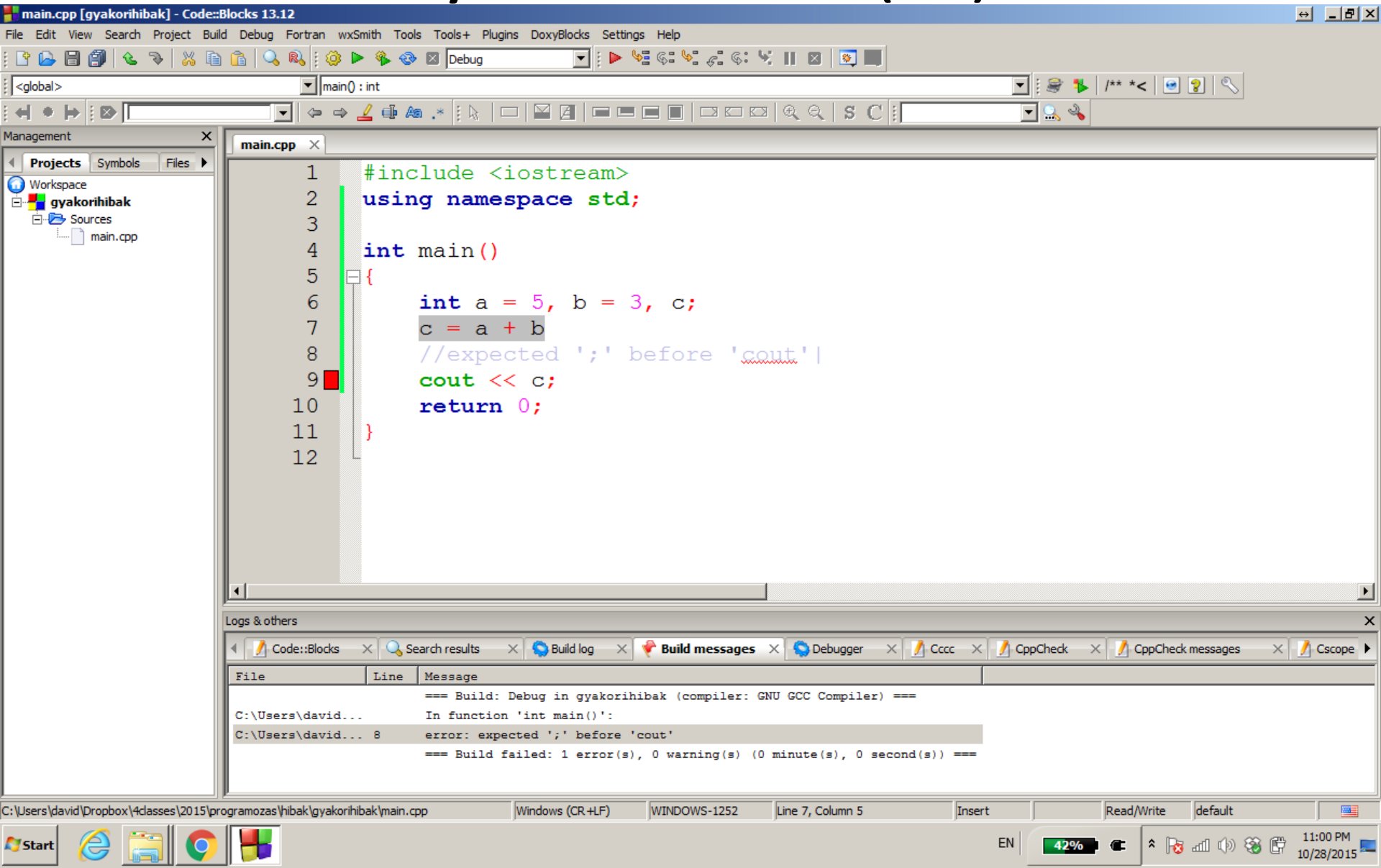
Gyakori hibák (ID)



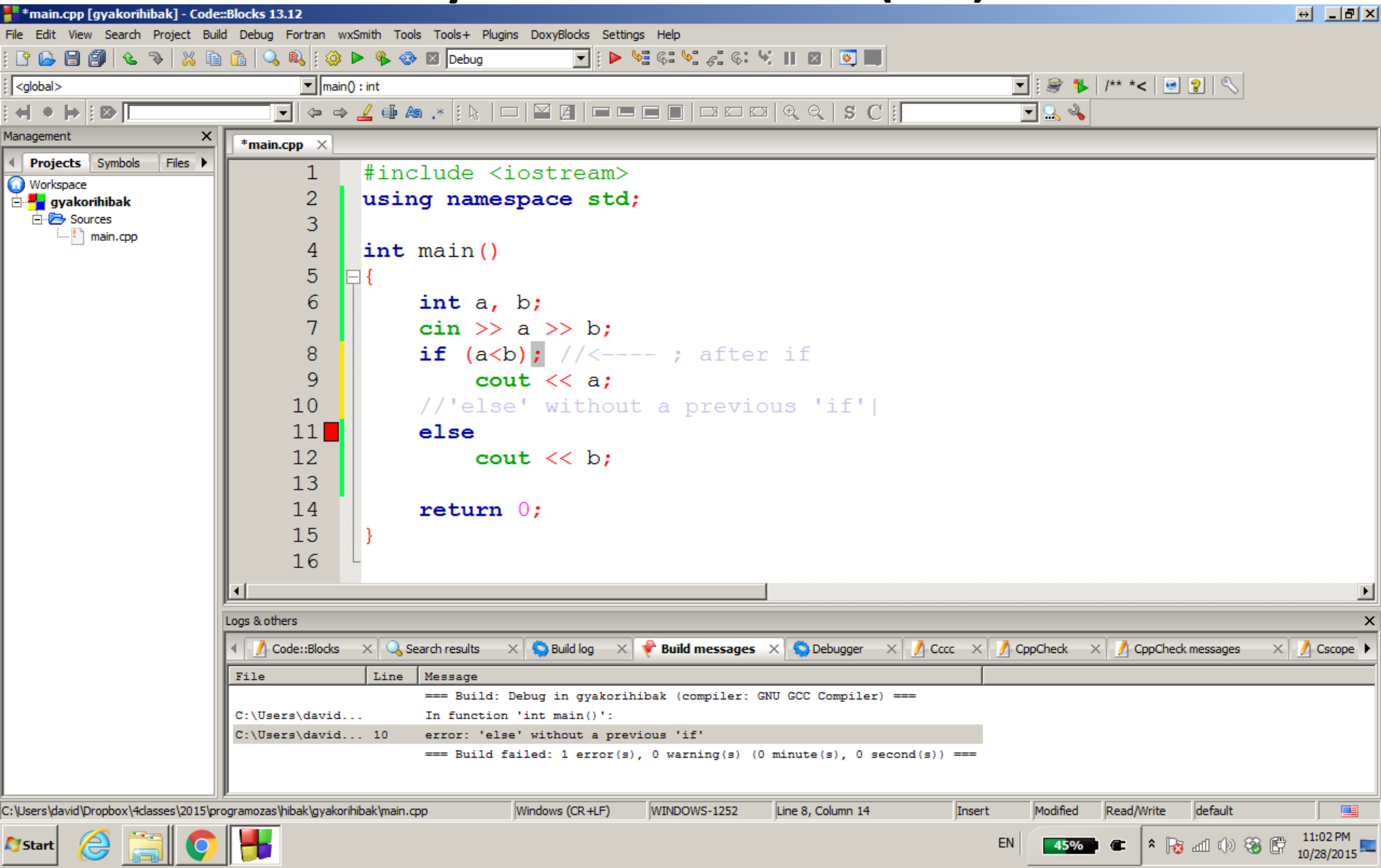
Gyakori hibák (ID)



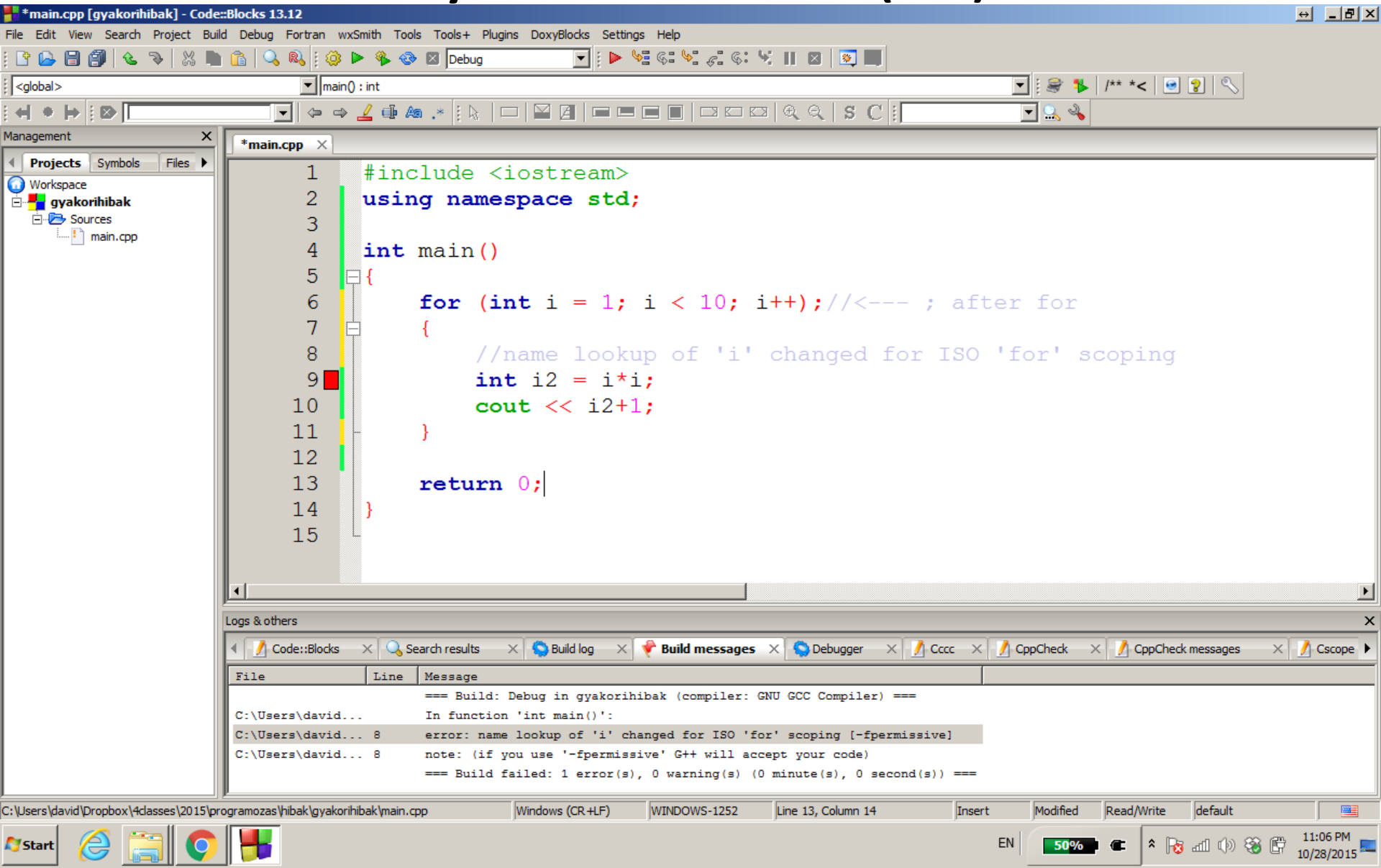
Gyakori hibák (ID)



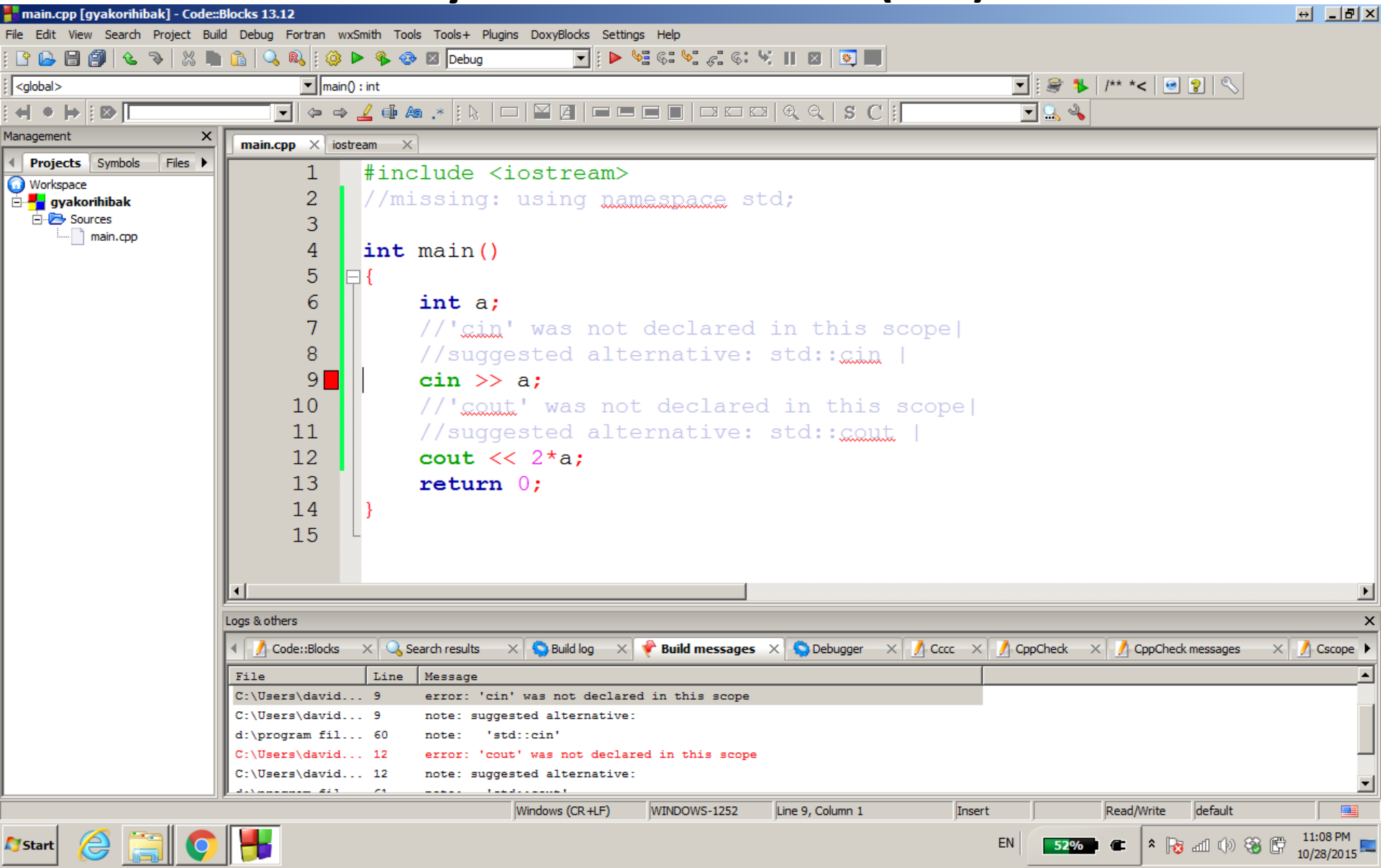
Gyakori hibák (ID)



Gyakori hibák (ID)



Gyakori hibák (ID)



Gyakori hibák (ID)