# Р2 Т01 Коректност: Прекорачење опсега вредности целих бројева (int)

**Р1: Т06 Статички методи: Група задатака 017: Статички методи:**

* [**https://arena.petlja.org/competition/r1-04-metode-01-staticke-metode**](https://arena.petlja.org/competition/r1-04-metode-01-staticke-metode)

## Пример 001: Факторијел 64бит

**Неисправно решење које НЕ пролази све комисијске тест примере:**

// R2 IT T01.01. Korektnost

// https://petlja.org/biblioteka/r/Zbirka/faktorijel\_64bit

// https://arena.petlja.org/competition/r1-04-metode-01-staticke-metode#tab\_129087

using System;

class Faktorijel

{

static int fakt(int n)

{

int p = 1;

for (int i = 2; i <= n; i++)

{

p = p \* i;

}

return p;

}

static void Main()

{

int n = int.Parse(Console.ReadLine());

int c = fakt(n);

Console.WriteLine(c);

}

}

**Исправно решење које пролази све комисијске тест примере:**

// R2 IT T01.01. Korektnost

// https://petlja.org/biblioteka/r/Zbirka/faktorijel\_64bit

// https://arena.petlja.org/competition/r1-04-metode-01-staticke-metode#tab\_129087

using System;

class Faktorijel

{

static **Int64** fakt(int n)

{

**Int64** p = 1;

for (int i = 2; i <= n; i++)

{

p = p \* i;

}

return p;

}

static void Main()

{

int n = int.Parse(Console.ReadLine());

**Int64** c = fakt(n);

Console.WriteLine(c);

}

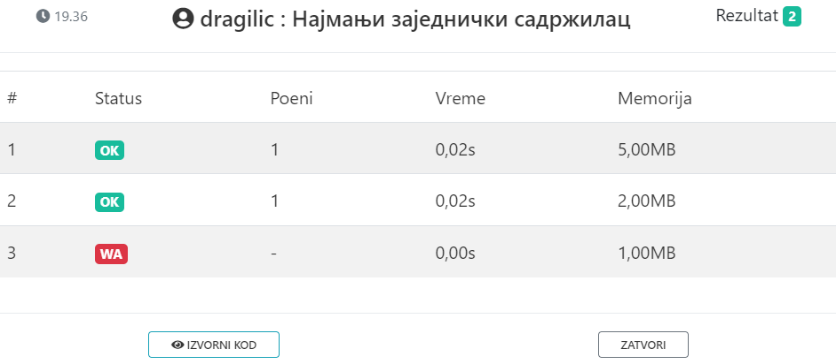
}

## Пример 002: Најмањи заједнички садржилац

**Неисправно решење које НЕ пролази све комисијске тест примере:**

// https://arena.petlja.org/competition/r1-04-metode-01-staticke-metode#tab\_29825

// https://petlja.org/biblioteka/r/Zbirka3/euklid

using System;

class Zadatak

{

static int nzd(int a, int b)

{

while (b > 0)

{

int p = a % b;

a = b;

b = p;

}

return a;

}

static void Main()

{

int n = int.Parse(Console.ReadLine());

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

{

string[] s = Console.ReadLine().Split();

int a = int.Parse(s[0]);

int b = int.Parse(s[1]);

int c = nzd(a, b);

int nzs = (a \* b) / c;

Console.WriteLine(nzs);

}

}

}

Програмски код не пролази трећи тест пример, јер су вероватно вредности улазних променљивих a и b веома велике, односно близу границе типа **int**, па је резултат њиховог производа прекорачио опсег вредности целобројног типа **int**.

**Исправно решење које пролази све комисијске тест примере:**

// https://arena.petlja.org/competition/r1-04-metode-01-staticke-metode#tab\_29825

// https://petlja.org/biblioteka/r/Zbirka3/euklid

using System;

class Zadatak

{

static int nzd(int a, int b)

{

while (b > 0)

{

int p = a % b;

a = b;

b = p;

}

return a;

}

static void Main()

{

int n = int.Parse(Console.ReadLine());

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

{

string[] s = Console.ReadLine().Split();

int a = int.Parse(s[0]);

int b = int.Parse(s[1]);

int c = nzd(a, b);

**Int64 nzs = ((Int64)a \* b) / c;**

Console.WriteLine(nzs);

}

}

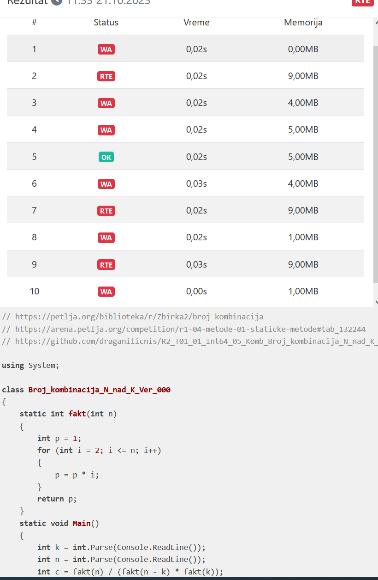
}

## Пример 003: Број комбинација – Верзија 000 – Пролази само 1 тест пример

// https://petlja.org/biblioteka/r/Zbirka2/broj\_kombinacija

// https://arena.petlja.org/competition/r1-04-metode-01-staticke-metode#tab\_132244

// https://github.com/draganilicnis/R2\_T01\_01\_Int64\_05\_Komb\_Broj\_kombinacija\_N\_nad\_K\_i\_Fact/tree/master

using System;

class Broj\_kombinacija\_N\_nad\_K\_Ver\_000

{

static int fakt(int n)

{

int p = 1;

for (int i = 2; i <= n; i++)

{

p = p \* i;

}

return p;

}

static void Main()

{

int k = int.Parse(Console.ReadLine());

int n = int.Parse(Console.ReadLine());

int c = fakt(n) / (fakt(n - k) \* fakt(k));

Console.WriteLine(c);

}

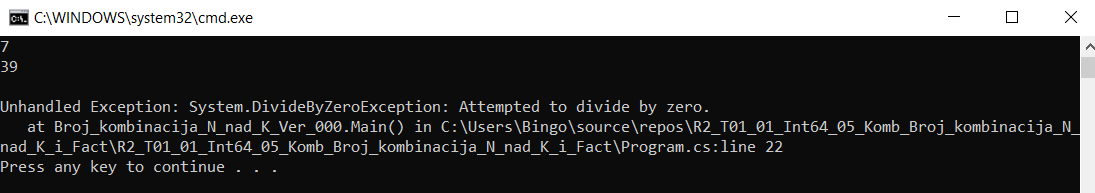
}

За израчунавање броја комбинација користимо уобичајену формулу:

Ова прва (нулта) верзија програмског кода пролази само један комисијски тест пример (тест пример 5), док за све остале пријављује грешку и то или погрешан одговор (WA) или грешку приликом извршавања програма (RTE).

С обзором да још увек немамо доступне комисијске тест примере, најпре је потребно да тестирамо нашу прву верзију програмског кода за тест примере наведене у самом тексту задатка.

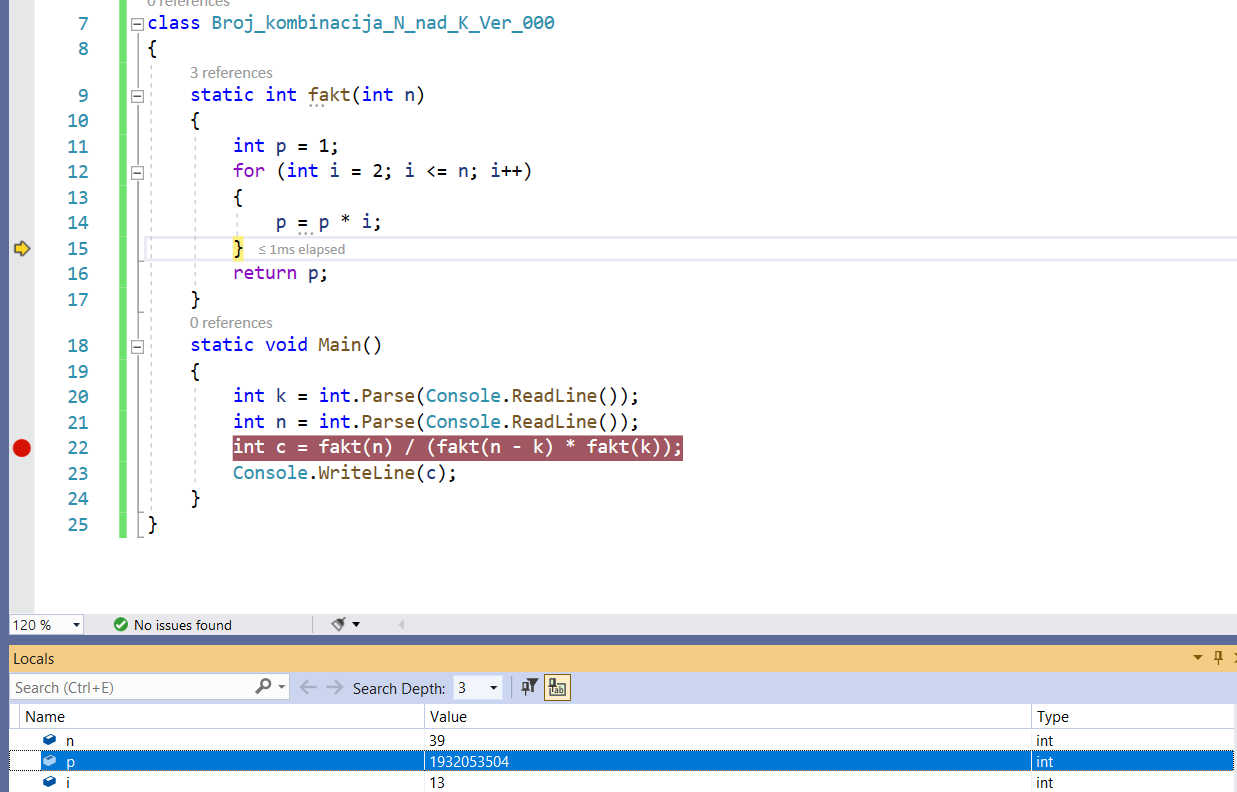
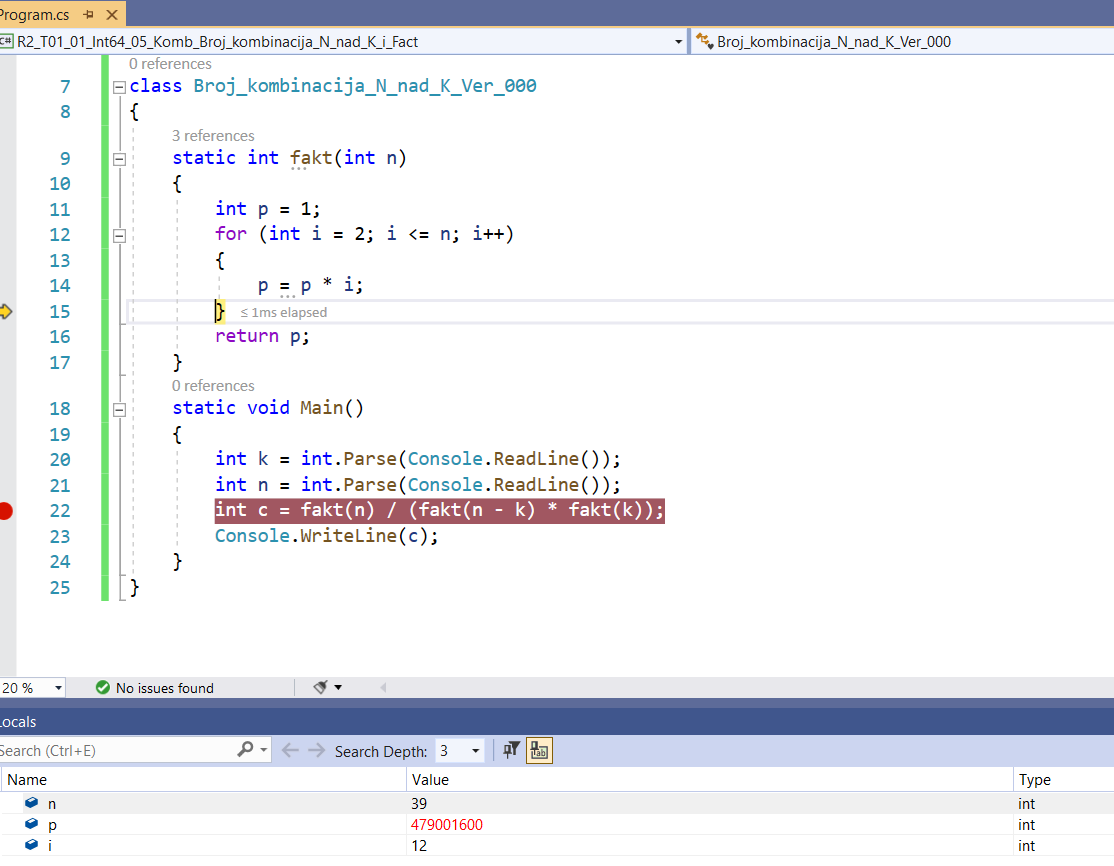
Уочавамо да за први тест пример у којем је k = 3, а n = 5 наша прва верзија програмског кода исписује тачан и очекиван излазни резултат 10, док међутим, за други тест пример у којем је **k = 7, а n = 39**, наша прва верзија програмског кода не само да не исписује тачан резултат већ пријављује грешку приликом извршавања програма (RTE), као на следећој слици:

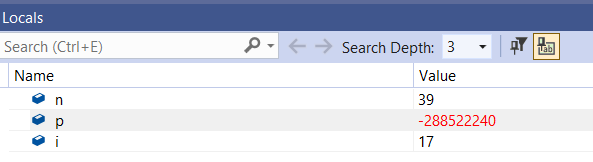
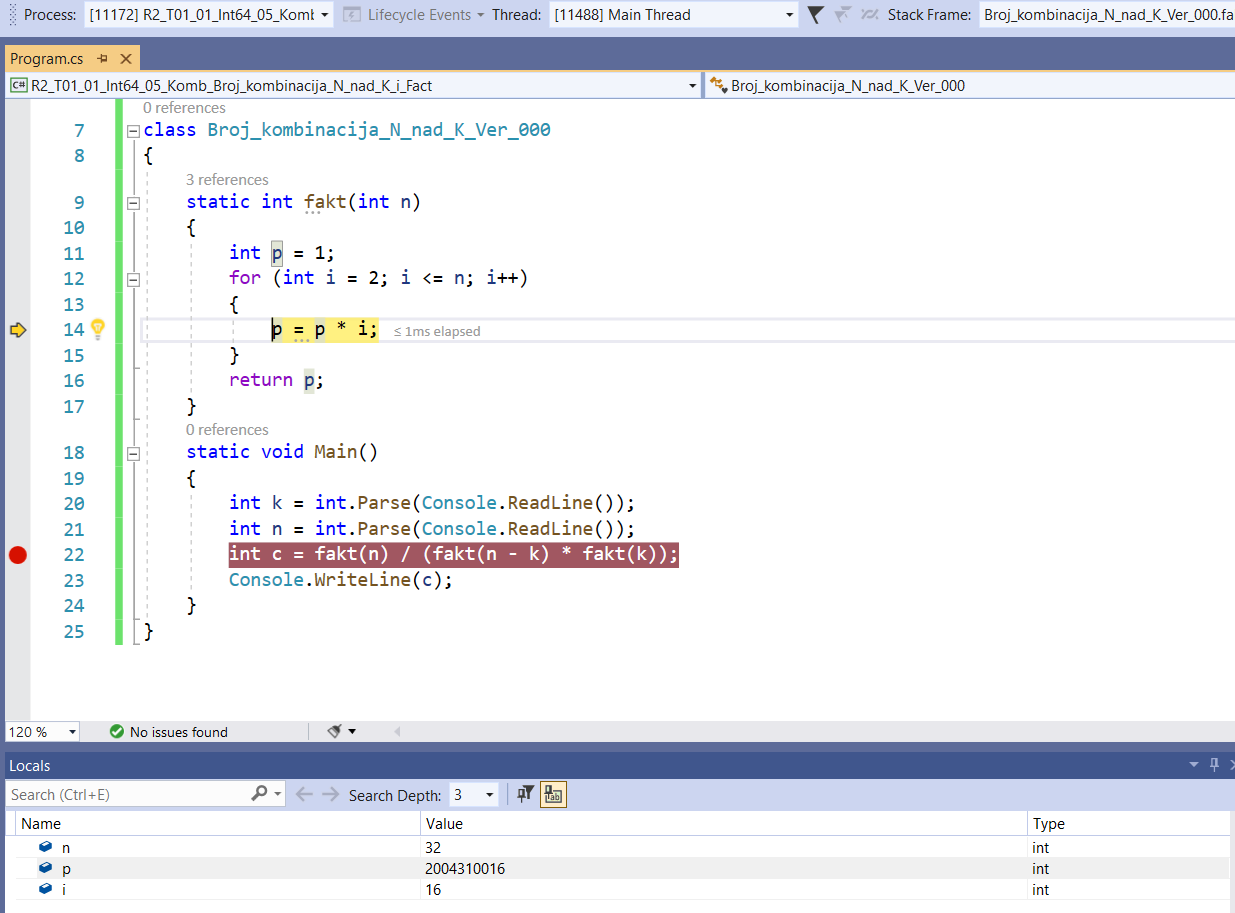


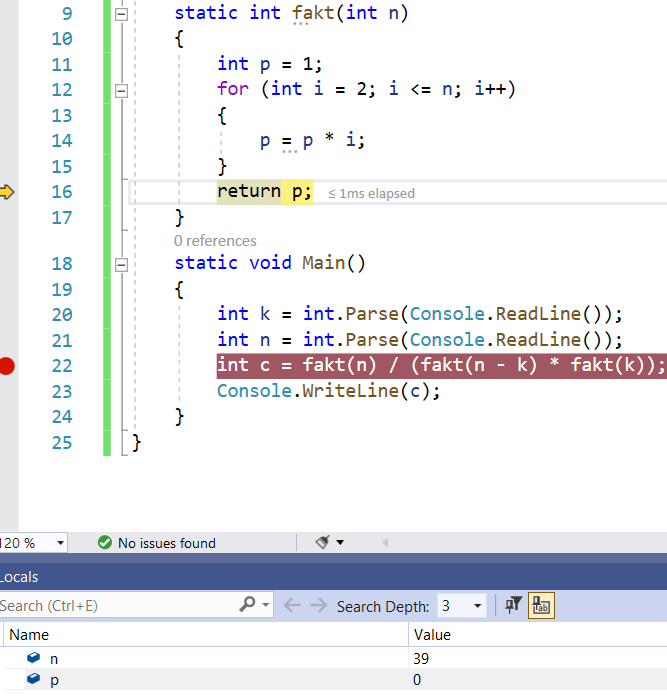
Уколико поставимо прекидну тачку у 22. линији кода у којој три пута позивамо методу и покренемо програм на извршавање корак по корак уочићемо следеће:

Приликом извршавања методе за fakt(n), када је вредност локалне променљиве (бројача петље) **i=12**, тренутна вредност променљиве p је очекивана и тачна **479001600**, односно 479 милиона, као на слици испод. Међутим, већ у следећој итерацији када је вредност бројача петље **i=13**, тренутна вредност променљиве више није очекивана ни тачна, уместо **6227020800** (6 милијарди), што би требало да буде производ 479001600 \* 13 = **6227020800**, програмски код приказује да је тренутна вредност производа **1932053504** (нешто мање од 2 милијарде), као на сликама испод.

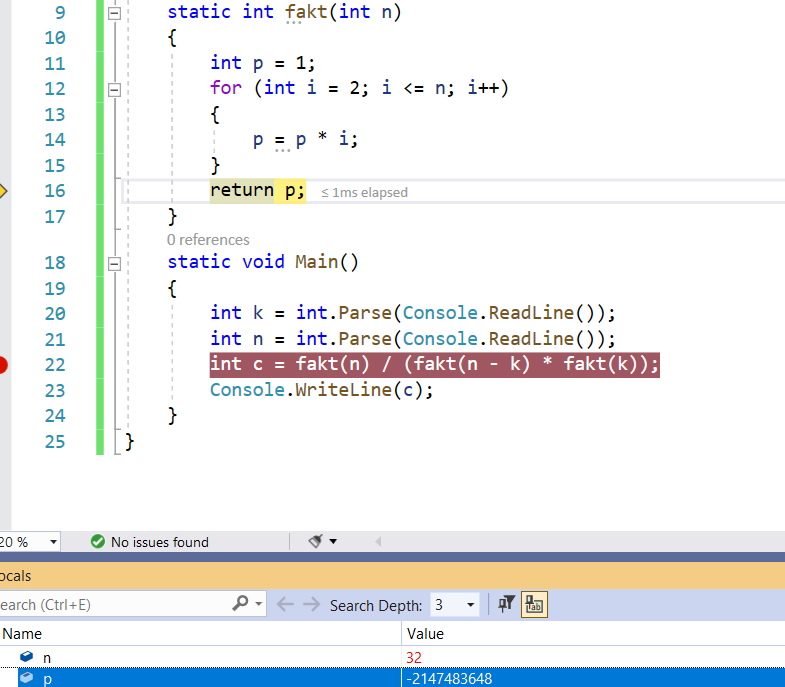
У наредним итерацијама тренутна вредност локалне променљиве p у телу петње у методи (потпрограму) се креће око 2 милијарде за вредноси бројача петље од 14 до 16, док за **i=17**, постаје негативан број, док за **i=34**, добија вредност 0, тако да функција на крају враћа вредност 0:



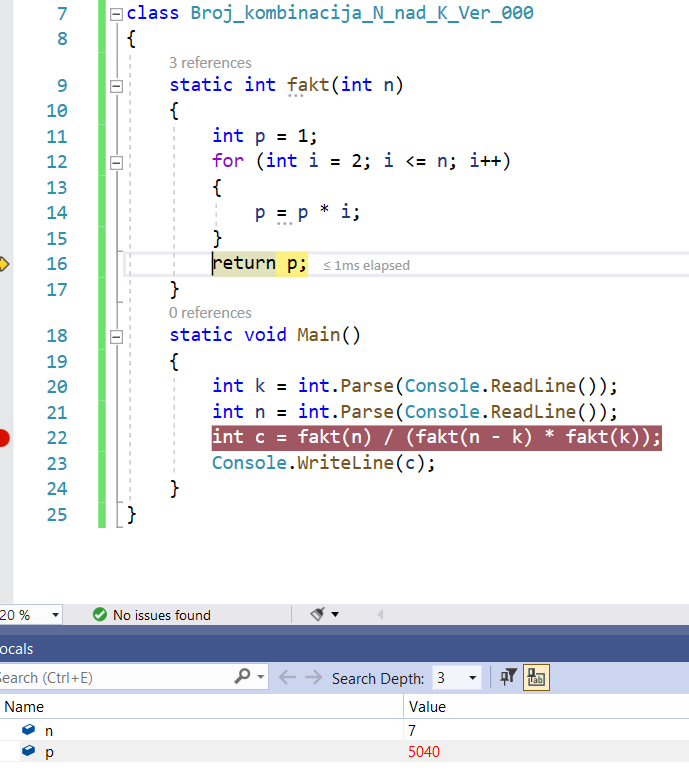




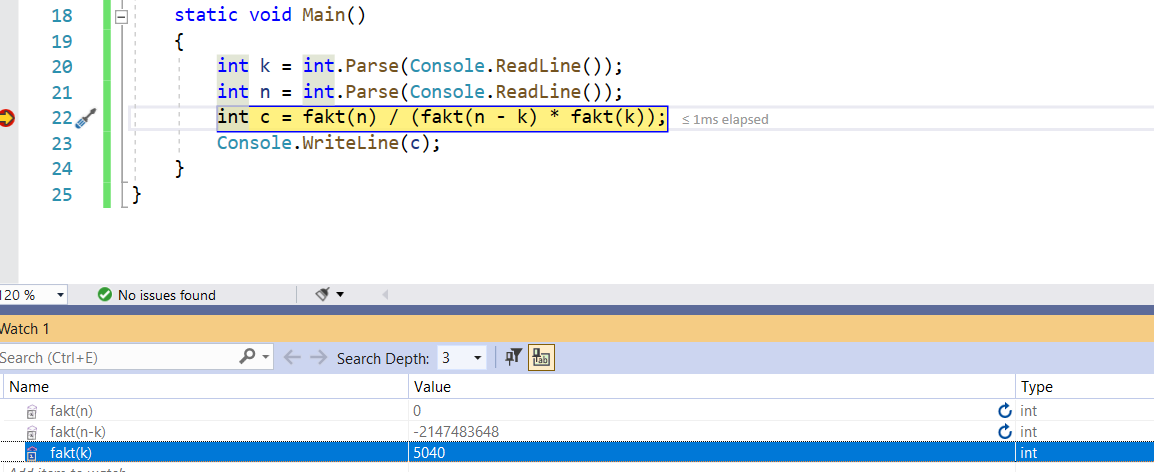
Слично се дешава приликом извршавања методе fakt(n-k), односно fakt(32) која враћа вредност променљиве p = **-2147483648** (минус 2 милијарде):



Само ће последњи трећи позив извршавања методе fakt(k), односно fakt(7) вратити очекивану тачну вредност променљиве p = **-5040**:



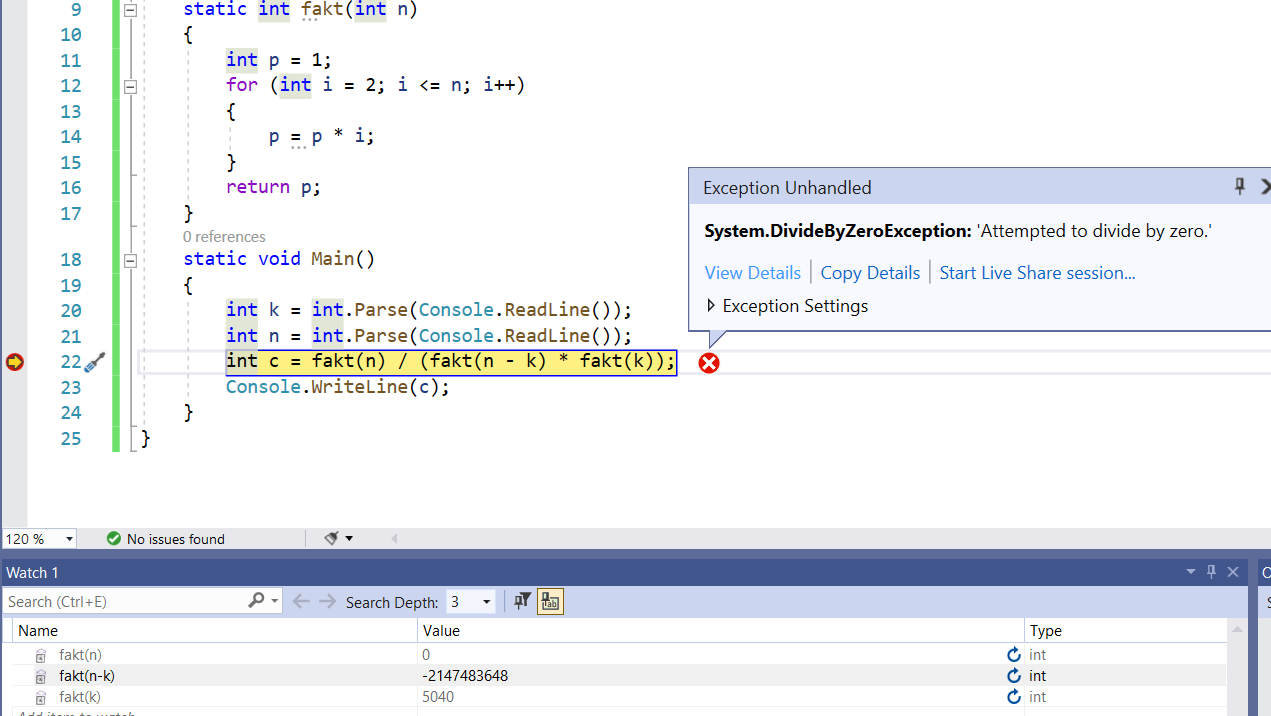
Након повратка у главни програм непосредно пре извршавања математичких операција множења и дељења тренутне вредности су:



Другим речима као да се извршава следећа наредба:

int c = 0 / ((-2147483648)\*5040);

чијим се извршавањем зауставља извршавање програма и приказује грешка приликом дељења нулом:



Можемо да закључимо да наша нулта верзија програмског решења не израчунава тачне резултате за улазне вредности променљивих код којих је аргумент методе (функције) **fakt** такав да је излазна вредност методе прекоричила опсег вредности целих бројева, односно 231 = 2,147,483,648 ≈ 2\*109, односно када је вредност аргумента већа од 11, јер је 13! = 6,227,020,800.

Предлог за следећу верзију:

Уместо целобројног типа **int** треба користити целобројни тип **Int64**.

## Пример 003: Број комбинација – Верзија 001 – Пролазе само 2 тест примера

// https://petlja.org/biblioteka/r/Zbirka2/broj\_kombinacija

// https://arena.petlja.org/competition/r1-04-metode-01-staticke-metode#tab\_132244



using System;

class Broj\_kombinacija\_N\_nad\_K\_Ver\_001

{

static **Int64** fakt(int n)

{

**Int64** p = 1;

for (int i = 2; i <= n; i++)

{

p = p \* i;

}

return p;

}

static void Main()

{

int k = int.Parse(Console.ReadLine());

int n = int.Parse(Console.ReadLine());

**Int64** c = fakt(n) / (fakt(n - k) \* fakt(k));

Console.WriteLine(c);

}

}

Ова нова (прва следећа) верзија програмског кода пролази само два комисијска тест пример (тест примере 3 и 5), док за све остале пријављује грешку, али сада само погрешан одговор (WA).

Примећујемо да више не пријављује грешку приликом извршавања програма (RTE), што је неки мали напредак.

Било би добро да поново поставимо прекидну тачку и тестирамо програмски код. Такође, с обзиром да смо раније закључили да је најкритичнији део прекорачење опсега вредности целих бројева у функцијском потпрограму, можемо да тестирамо програмски код за једноставне вредности када је улазна вредност променљиве k = 1, a у сваком наредном тестирању да повећавамо вредност улазне променљиве n. С обзиром да знамо на је број комбинација у том случају увек једнак n. Врло брзо ћемо закључити да програм израчунава тачне вредности факторијала за вредности променљиве n које нису веће од 20, али да већ за n = 21 постижемо прекорачење, јер је:

**21! = 51,090,942,171,709,400,000 = 5,1 \* 1019, што је веће од 1018, односно 263.**

С обзиром да је у услову задатка наведено да вредност улазне променљиве не сме бити већа од 40, али да може бити већа од 20 постаје јасно да некако морамо да се ослободимо израчунавања вредности које су **веће од 21!**

## Пример 003: Број комбинација – Верзија 002 – Пролазе само 5 тест примера

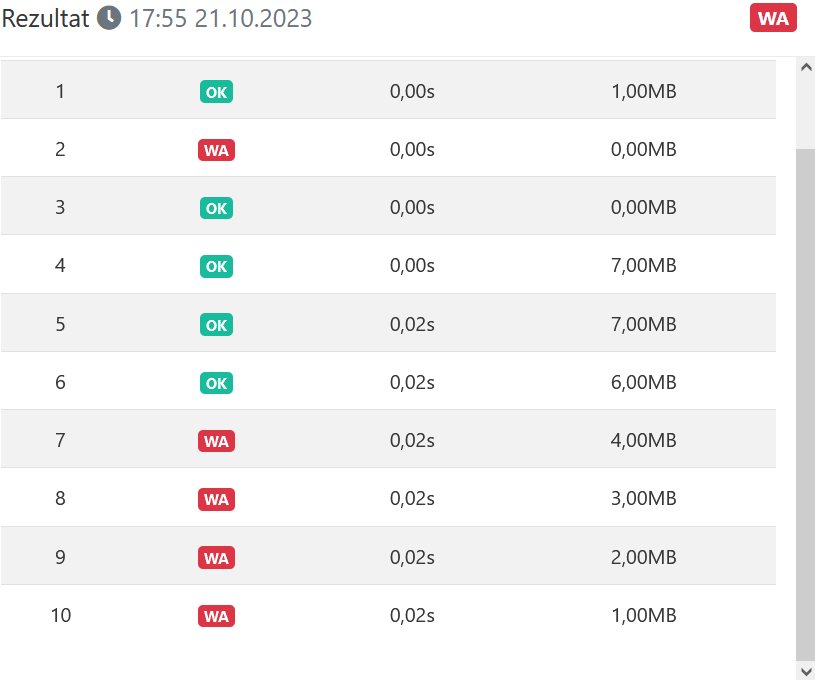
Покушаћемо да уместо уобичајене формуле за израчунавање броја комбинација:

Користимо следећу формулу:

// https://petlja.org/biblioteka/r/Zbirka2/broj\_kombinacija

// https://arena.petlja.org/competition/r1-04-metode-01-staticke-metode#tab\_132244

using System;



class Broj\_kombinacija\_N\_nad\_K\_Ver\_002

{

static Int64 fakt(int n)

{

Int64 p = 1;

for (int i = 2; i <= n; i++)

{

p = p \* i;

}

return p;

}

static void Main()

{

int k = int.Parse(Console.ReadLine());

int n = int.Parse(Console.ReadLine());

**Int64 g = 1;**

**for (int i = 0; i < k; i++)**

**{**

**g = g \* n;**

**n--;**

**}**

**Int64 d = fakt(k);**

**Int64 c = g / d;**

Console.WriteLine(c);

}

}

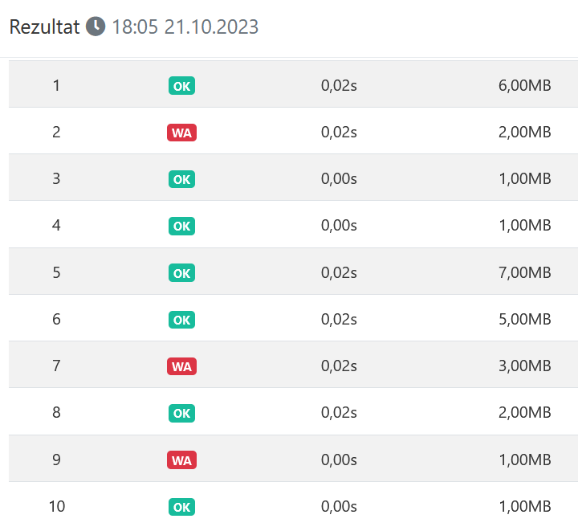
Сада добијамо мало боље резултате, јер нам пролази 5 комисијских тест примера (1, 3, 4, 5 и 6), док за свих осталих 5 пријављује грешку погрешан одговор (WA).

## Пример 003: Број комбинација – Верзија 003 – Пролазе 7 од 10 тест примера

Покушаћемо да искористимо следеће правило:

// https://petlja.org/biblioteka/r/Zbirka2/broj\_kombinacija

// https://arena.petlja.org/competition/r1-04-metode-01-staticke-metode#tab\_132244

using System;

class Broj\_kombinacija\_N\_nad\_K\_Ver\_003

{

static Int64 fakt(int n)

{

Int64 p = 1;

for (int i = 2; i <= n; i++)

{

p = p \* i;

}

return p;

}

static void Main()

{

int k = int.Parse(Console.ReadLine());

int n = int.Parse(Console.ReadLine());

Int64 g = 1;

**if (k > n / 2) k = n - k;**

for (int i = 0; i < k; i++)

{

g = g \* n;

n--;

}

Int64 d = fakt(k);

Int64 c = g / d;

Console.WriteLine(c);

}

}

Добијамо мало боље резултате, јер нам пролази 7 од 10 комисијских тест примера. За 3 тест примера (2, 7 и 9) пријављује грешку погрешан одговор (WA).

## Пример 003: Број комбинација – Верзија 004 – Пролазе свих 10 тест примера

Примећујемо да је бројилац увек већи од имениоца, па је самим тим критичнији за прекорачење ограничења целобројних вредности производа.

Зато ћемо сада покушати да одмах израчунавамо парцијалне вредности количника на основу следеће формуле:

Наравно, морамо одмах да приметимо да ће парцијални количници целих бројева бити реални бројеви, међутим множењем тих реалних бројева на крају ћемо сасвим сигурно добити цео број, као производ. Ипак, за сваки случај извршићемо заокруживање последње вредности производа свих парцијалних количника на цео број:

// https://petlja.org/biblioteka/r/Zbirka2/broj\_kombinacija

// https://arena.petlja.org/competition/r1-04-metode-01-staticke-metode#tab\_132244

// https://github.com/draganilicnis/R2\_T01\_01\_Int64\_05\_Komb\_Broj\_kombinacija\_N\_nad\_K\_i\_Fact/tree/master

using System;

class Broj\_kombinacija\_N\_nad\_K\_Ver\_004\_005

{

**static Int64 Comb(int n, int k)**

**{**

**double p = 1.0;**

**if (k > n / 2) k = n - k;**

**for (int i = 0; i < k; i++)**

**{**

**p = p \* (n - i) / (k - i);**

**}**

**Int64 c = (Int64)Math.Round(p);**

**return c;**

**}**

static void Main()

{

int k = int.Parse(Console.ReadLine());

int n = int.Parse(Console.ReadLine());

**Int64 c = Comb(n, k);**

Console.WriteLine(c);

}

}

Сада наш програмски код пролази свих 10 комисијских тест примера.

Осим оваквог решења можемо размотрити и друге идеје.

## Пример 003: Број комбинација – Верзија 005

Приликом идејног решавања задатака пожељно је користити и друге алате, попут програма за табеларна израчунавања Ексел. Документ се може преузети [овде](https://drive.google.com/file/d/1Q8ZmZcxzYhtDiFxVcp0ELLLI8HOzQTXM/view).

|  |  |  |
| --- | --- | --- |
| N | N! | N! |
| 0 | 1 | 1 |
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 6 | 6 |
| 4 | 24 | 24 |
| 5 | 120 | 120 |
| 6 | 720 | 720 |
| 7 | 5040 | 5040 |
| 8 | 40320 | 40320 |
| 9 | 362880 | 362880 |
| 10 | 3628800 | 3628800 |
| 11 | 39916800 | 39916800 |
| 12 | 479001600 | 479001600 |
| 13 | 6227020800 | 6227020800 |
| 14 | 87178291200 | 87178291200 |
| 15 | 1,30767E+12 | 1307674368000 |
| 16 | 2,09228E+13 | 20922789888000 |
| 17 | 3,55687E+14 | 355687428096000 |
| 18 | 6,40237E+15 | 6402373705728000 |
| 19 | 1,21645E+17 | 121645100408832000 |
| 20 | 2,4329E+18 | 2432902008176640000 |
| 21 | 5,10909E+19 | 51090942171709400000 |
| 22 | 1,124E+21 | 1124000727777610000000 |
| 23 | 2,5852E+22 | 25852016738885000000000 |
| 24 | 6,20448E+23 | 620448401733239000000000 |
| 25 | 1,55112E+25 | 15511210043331000000000000 |
| 26 | 4,03291E+26 | 403291461126606000000000000 |
| 27 | 1,08889E+28 | 10888869450418400000000000000 |
| 28 | 3,04888E+29 | 304888344611714000000000000000 |
| 29 | 8,84176E+30 | 8841761993739700000000000000000 |
| 30 | 2,65253E+32 | 265252859812191000000000000000000 |
| 31 | 8,22284E+33 | 8222838654177920000000000000000000 |
| 32 | 2,63131E+35 | 263130836933694000000000000000000000 |
| 33 | 8,68332E+36 | 8683317618811890000000000000000000000 |
| 34 | 2,95233E+38 | 295232799039604000000000000000000000000 |
| 35 | 1,03331E+40 | 10333147966386100000000000000000000000000 |
| 36 | 3,71993E+41 | 371993326789901000000000000000000000000000 |
| 37 | 1,37638E+43 | 13763753091226300000000000000000000000000000 |
| 38 | 5,23023E+44 | 523022617466601000000000000000000000000000000 |
| 39 | 2,03979E+46 | 20397882081197400000000000000000000000000000000 |
| **40** | **8,15915E+47** | **815915283247898000000000000000000000000000000000** |
| 41 | 3,34525E+49 | 33452526613163800000000000000000000000000000000000 |
| 42 | 1,40501E+51 | 1405006117752880000000000000000000000000000000000000 |
| 43 | 6,04153E+52 | 60415263063373800000000000000000000000000000000000000 |
| 44 | 2,65827E+54 | 2658271574788450000000000000000000000000000000000000000 |
| 45 | 1,19622E+56 | 119622220865480000000000000000000000000000000000000000000 |
| 46 | 5,50262E+57 | 5502622159812090000000000000000000000000000000000000000000 |
| 47 | 2,58623E+59 | 258623241511168000000000000000000000000000000000000000000000 |
| 48 | 1,24139E+61 | 12413915592536100000000000000000000000000000000000000000000000 |
| 49 | 6,08282E+62 | 608281864034268000000000000000000000000000000000000000000000000 |
| 50 | 3,04141E+64 | 30414093201713400000000000000000000000000000000000000000000000000 |
| 51 | 1,55112E+66 | 1551118753287380000000000000000000000000000000000000000000000000000 |
| 52 | 8,06582E+67 | 80658175170943900000000000000000000000000000000000000000000000000000 |
| 53 | 4,27488E+69 | 4274883284060030000000000000000000000000000000000000000000000000000000 |
| 54 | 2,30844E+71 | 230843697339241000000000000000000000000000000000000000000000000000000000 |
| 55 | 1,26964E+73 | 12696403353658300000000000000000000000000000000000000000000000000000000000 |
| 56 | 7,10999E+74 | 710998587804863000000000000000000000000000000000000000000000000000000000000 |
| 57 | 4,05269E+76 | 40526919504877200000000000000000000000000000000000000000000000000000000000000 |
| 58 | 2,35056E+78 | 2350561331282880000000000000000000000000000000000000000000000000000000000000000 |
| 59 | 1,38683E+80 | 138683118545690000000000000000000000000000000000000000000000000000000000000000000 |
| 60 | 8,32099E+81 | 8320987112741390000000000000000000000000000000000000000000000000000000000000000000 |
| 61 | 5,0758E+83 | 507580213877225000000000000000000000000000000000000000000000000000000000000000000000 |
| 62 | 3,147E+85 | 31469973260387900000000000000000000000000000000000000000000000000000000000000000000000 |
| 63 | 1,98261E+87 | 1982608315404440000000000000000000000000000000000000000000000000000000000000000000000000 |
| 64 | 1,26887E+89 | 126886932185884000000000000000000000000000000000000000000000000000000000000000000000000000 |

Из ове табеле видимо да је 40! број са 47 цифара, што значи да је знатно изнад опсега вредности целобројног типа, те да нам је неопходна другачија идеја за решавање задатка.

Такође, можемо веома брзо и лако да направимо табеларни преглед очекиваних вредности, као на слици:



Можемо да почнемо да разматрамо само првих 10 и да покушамо да уочимо нека правила:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **N \ K** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **0** | **1** |  |  |  |  |  |  |  |  |  |  |
| **1** | 1 | 1 |  |  |  |  |  |  |  |  |  |
| **2** | 1 | **2** | 1 |  |  |  |  |  |  |  |  |
| **3** | 1 | 3 | 3 | 1 |  |  |  |  |  |  |  |
| **4** | 1 | 4 | **6** | 4 | 1 |  |  |  |  |  |  |
| **5** | 1 | 5 | 10 | 10 | 5 | 1 |  |  |  |  |  |
| **6** | 1 | 6 | 15 | **20** | 15 | 6 | 1 |  |  |  |  |
| **7** | 1 | 7 | 21 | 35 | 35 | 21 | 7 | 1 |  |  |  |
| **8** | 1 | 8 | 28 | 56 | **70** | 56 | 28 | 8 | 1 |  |  |
| **9** | 1 | 9 | 36 | 84 | 126 | 126 | 84 | 36 | 9 | 1 |  |
| **10** | 1 | 10 | 45 | 120 | 210 | **252** | 210 | 120 | 45 | 10 | 1 |

Ако мало боље визуелно распоредимо резулате добићемо следећу табелу:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **N** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **0** |  |  |  |  |  |  | **1** |  |  |  |  |  |  |
| **1** |  |  |  |  |  | **1** |  | **1** |  |  |  |  |  |
| **2** |  |  |  |  | **1** |  | **2** |  | **1** |  |  |  |  |
| **3** |  |  |  | **1** |  | **3** |  | **3** |  | **1** |  |  |  |
| **4** |  |  | **1** |  | **4** |  | **6** |  | **4** |  | **1** |  |  |
| **5** |  | **1** |  | **5** |  | **10** |  | **10** |  | **5** |  | **1** |  |
| **6** | **1** |  | **6** |  | **15** |  | **20** |  | **15** |  | **6** |  | **1** |

Односно:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **N** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **0** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **1** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **4** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **5** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **6** |  |  |  |  |  |  |  |  |  |  |  |  |  |

Након мало времена можемо да уочимо следећа правила:

Можемо да уведемо матрицу димензија 41\*41 у којој ћемо да чувамо вредности комбинација:

// https://petlja.org/biblioteka/r/Zbirka2/broj\_kombinacija

// https://arena.petlja.org/competition/r1-04-metode-01-staticke-metode#tab\_132244

// https://github.com/draganilicnis/R2\_T01\_01\_Int64\_05\_Komb\_Broj\_kombinacija\_N\_nad\_K\_i\_Fact/tree/master

using System;

class Broj\_kombinacija\_N\_nad\_K\_Ver\_006

{

static Int64 Comb(int n, int k)

{

Int64[,] C = new Int64[n+1, n+1];

C[0, 0] = 1;

for (int i = 1; i <= n; i++)

{

C[i, 0] = 1;

C[i, i] = 1;

for (int j = 1; j <= k; j++)

C[i, j] = C[i - 1, j - 1] + C[i - 1, j];

}

Int64 c = C[n, k];

return c;

}

static void Main()

{

int k = int.Parse(Console.ReadLine());

int n = int.Parse(Console.ReadLine());

Int64 c = Comb(n, k);

Console.WriteLine(c);

}

}

На крају треба рећи да се овај задатак може решити и коришћењем рекурзије о којој ће бити речи касније.

## Препорука

Обавезно урадити (макар само за ова три задатка) следеће (у развојном окружењу):

* Извршити програм корак по корак (тастер **F11**) и пажљиво пратити вредности аргумената (стварних) и параметара метода (формалних аргумената), посебну пажњу обратити на прозор **Autos** и **Local**, као и на програмски стек (**Call Stack**).
* <https://petlja.org/kurs/11171/28/1374>