# Enumeration and generation of all constitutional alkane isomers of methane to icosane using recursive generation and a modified Morgan's algorithm

Andreas Gimpel
*agimpel@student.ethz.ch*

Department of Chemistry and Applied Biosciences, ETH Zürich
Vladimir-Prelog-Weg 1-5/10, 8093 Zürich, Switzerland

an entry to the challenge of Prof. Philippe H. Hünenberger
issued in his lecture "Informatik I for Chemists" on 29.10.2015

**Abstract:** This algorithm employs a simple canonical code for representation of isomers for use in generation. The generation process is recursive. This simple approach was chosen due to the convenience of generation. For large n, however, while providing all possible isomers, it creates many isomers which are identical. This renders the code inefficient for alkanes with higher carbon count.
In order to identify isomers as unique, a modified Morgan's algorithm was used to determine the internal structure of the canonical isomer code. To do so, the isomer code was translated into a connectivity table and transformed into a basic Morgan's code. Using the information about internal bonds supplied by the connectivity table, the Morgan's algorithm was executed a definite amount of times, but the so-called "Extended Connectivities (EC)"-values were not translated back into a ranking structure. Instead, they were used to compare each generated isomer to the previous ones, effectively preventing any superfluous isomers to be accepted.
While the algorithm scales poorly with carbon count, it provides exact results throughout all alkanes from methane to icosane in less than one hour. The Morgan's algorithm has significantly increased the accuracy of the isomers produced by the recursive generation while reducing the amount of rules required for generation. Thus decreasing complexity of the generation algorithm.

## Introduction

The program uses two different code representations of isomers: a modified n-tuple code and a modified code based on Morgan's algorithm. Despite both codes being canonical, two representations were necessary to depict the information needed to enumerate isomers. While the n-tuple code can represent the basic structure of the isomer in a simple and easily reproducible way, it lacks information on the exact internal structure. The Morgan code is able to represent the internal structure, but is not readable by itself, rendering generation a complex task.
To accommodate the use of two representations, three algorithms were necessary in total: the generation of isomers using the n-tuple code, the translation of the n-tuple code into Morgan code, and lastly the examination of the isomer structure with the Morgan code.

## The modified n-tuple code

The representation of isomers with the modified n-tuple code is based on the n-tuple code described in "Enumerating Molecules", by J.L. Faulon, D.P. Visco and D. Roe[1].
The basic representation of an isomer starts with the so-called 'root' of the isomer. This is one of the carbon atoms which has the most bonds (C-H bonds are neglected). From this atom which receives a value equal to its C-C bonds, one continues by following the bond to the next atom directly connected to the root which has the most bonds. This atom receives a value equal to the number of bonds besides the bond that was used to connect to it. In this representation, a branch ends with a atom which has a value of 0. If an end was reached, the next branch with the highest amount of bonds is chosen.

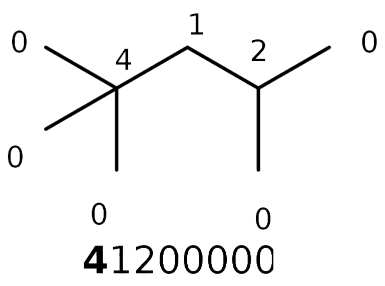


Figure 1: Example of the standard n-tuple code for 2,2,4-trimethylpentane. The numeration starts at the atom with four bonds and continues along the main branch. After its end, the methyl-substituents are enumerated with 0s. This way, the code becomes canonical.

One disadvantage of the n-tuple representation is that by enumerating branches simply by the order of their bonds removes information about the exact bonds in the isomer, because there is no information how multiple branches are connected together. For this reason, the modified n-tuple code does not enumerate branches ordered by the amount of bonds, but rather based on branch connectivity. To do so, every branch will first be completed entirely before the next branch of the root is enumerated.

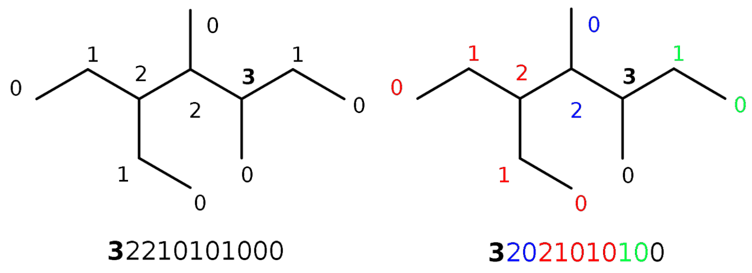


Figure 2: Example of bond information loss of a standard n-tuple code with 3-ethyl-4,5-dimethylheptane as example. *Left*: standard n-tuple code. It is impossible to distinguish between branches of the root, and branches of branches. *Right*: modified n-tuple code. The bond structure of the isomer is preserved. The color codes show how every branch can easily be recognized in the modified n-tuple. Therefore, the modification is necessary to preserve connectivity information which is needed for Morgan's algorithm.

## The code based on a modified Morgan's Algorithm

Morgan's algorithm was created in 1965 by H.L. Morgan and is used for creating canonical numbering of molecules, see "Morgan Algorithmus - Tutorial", by Prof. Dr. J. Gasteiger, Dr. Th. Engel [2].
Normally, this algorithm assigns each atom a value based on the amount of bonds to important atoms (again neglecting C-H bonds). Then, every atom is given the sum of the values of its neighboring atoms as its new value. This is repeated until the amount of unique values does not change. At this point, every atom will be numbered based on their own and their neighbor's values. In this modified version, every atom also receives a value based on its C-C bonds, but the summation of values is not terminated dynamically. Instead, the iteration will be done a definite amount of times for every isomer of a given alkane to ensure that the generated values are comparable. After completion, the values are sorted by value.
This modified algorithm is capable of differentiating between constitutional isomers while declaring identical codes to identical isomers. The modified n-tuple code is not able to differ between all possible identical isomers, which renders the Morgan code crucial in eliminating all superfluous isomer codes prior to enumeration.

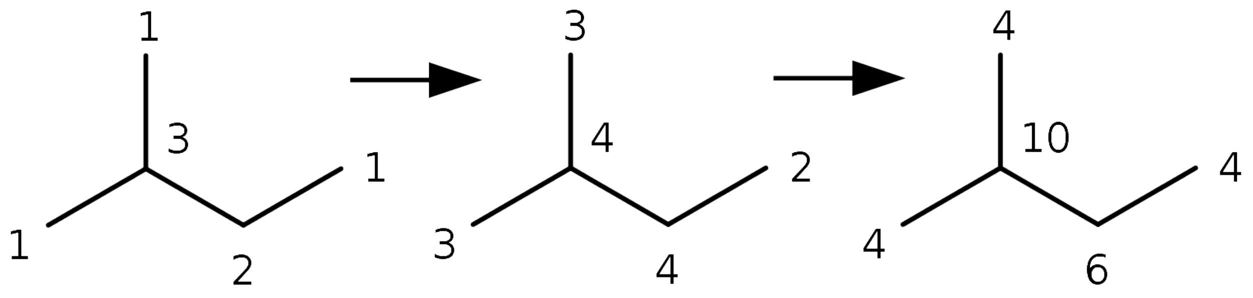


Figure 3: Example of the Morgan's algorithm with 2-methylbutane. Every step is another iteration of the algorithm which systematically assigns new values based on neighboring atoms. It is evident that the assigned values are heavily dependent on the isomer's internal structure. This allows the Morgan's algorithm to differ between constitutional isomers.

## Algorithm A: generation

The generation of isomers is recursive, therefore all isomers have to be saved. Here, a vector *isomers* of the dimensions 22x500000x22 is used for storage. Its structure is as follows, where n is the amount of atoms in the current alkane:

- *isomers[n][0][0]* saves the amount of valid isomers found (x) for this alkane
- *isomers[n][1..x][1..n]* represents the values of the digits of this isomer of this alkane

The provided isomer used to generate all other isomers recursively is methane (code [0]). The implementation loops through every isomer of the previous alkane and each of the digits of these isomers. Under the following conditions, a digit is incremented and a 0 is inserted after it, effectively adding a methyl-substituent to this carbon atom, while all other digits are copied. Because a carbon atom may not have more than 4 bonds, and a non-root atom may not exceed the value of the root, the value a digit of the isomer code may have is limited to [0,1,2,3] (in addition to 4, if the digit resembles the root).
If a isomer is generated, it will be checked for uniqueness by algorithm C after translation by algorithm B. A unique isomer will be kept, while superfluous isomers are discarded by overwriting them in *isomers*.

**Algorithm A: generation (pseudocode)**

```
for every isomer of the previous alkane's isomers {
  for every digit of the isomer's code {
    copy digits up to current digit;

    if the digit's value + 1 is a valid value {
      increment digit value and insert into vector;
      add value 0 to position digit+1;

      copy all digits from digit+1 to digit+2 of new isomer;

      if new isomer is unique {
        increment amount of valid isomers found;
        save this isomer;
        next isomer at this+1 position;
      } else {
        discard this isomer;
        next isomer at this position;
      }
    }
  }
}
```

The poor scaling of the generation algorithm is evident, as the double loop scales with *(isomers of previous alkane)*(n)*. Since the isomer amount scales exponentially, the performance of this implementation is especially poor. For large n, additional rules that limit the possibilities are necessary to retain justifiable efficiency. Still, for n<=20, the implementation is sufficiently fast.

## Algorithm B: translation

In order to translate the modified n-tuple code into a representation that can be used by Morgan's algorithm, the connections in the isomer have to be known. As the n-tuple code was modified to retain this information, the following algorithm uses this information to create a connectivity table in the form of a two-dimensional vector *connections*. This vector is used as follows:

- *connections[digit][0]* saves the amount of connections of the atom at position *digit*
- *connections[digit][1..x]* points to the digits of the atoms this atom is connected to

In the isomer representation, only *forward* connections are represented. This is the reason why all digits besides the root have a value of (number of C-C bonds)-1, as all atoms but the root are connected to a previous atom in a *backwards* fashion. This idea of forward connections is crucial to find the bonds of an atom: every atom represented by a digit in the code is *forward* connected to the atom of the next digit (unless it is 0 and thus an end of the current branch) and therefore, the atom at the next digit is *backwards* connected to the atom at this digit. This procedure thus also allows to identify backwards connections.
In addition, the concept of 'foreign connections' is important to determine which atoms are branches of the active atom, and which are branches of branches. If an atom X has a connection to another atom A as well as atom B (A digit comes before B digit in code), atoms following A in code may not be connected to the atom X, because A continues its branch if A>0. Therefore,

a certain amount of digits in the code following A cannot be connected to X. These atoms are foreign connections, as they come before B, but do not connect to X. From the example, it is

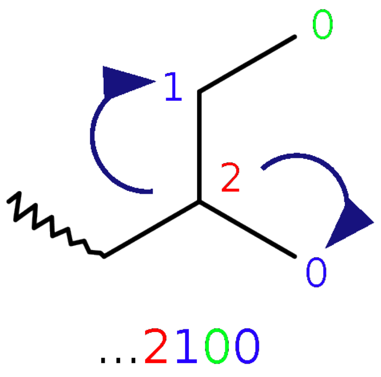


Figure 4: Example of foreign connections. The atom whose connections are searched for is red, and it is connected to the blue digits. The green digit however is placed before the second connection in the code, which means it is a foreign connection and therefore not connected to the red atom.

apparent that every atom has a number of forward connections equal to its value. This is true for directly connected atoms as well as foreign connections. Therefore, after an atom connected to the atom X, a number of foreign connections equal to the connected atom's value follow. Zeros do not add foreign connections and thus reduce the number of foreign connections left until another directly connected atom of atom X is found.
Using this methodology, it is possible to identify all the digits, corresponding to the atoms an atom is directly connected to. This is necessary for the Morgan algorithm. The morgan codes are stored in a comparable way to *isomers*, in the vector *morgans[1..x][1..n]*. This vector is reinitialized for each alkane and thus only contains the current alkane's isomer codes (where x is the isomer number). For the zeroth iteration, the amount of C-C bonds represent the value for each atom, which is done in the translation process also.

## Algorithm B: translation (pseudocode)

```
for every digit in the isomer code {
  if digit is root: copy value to morgans at 1;
  if digit is not root: copy value to morgans at digit and add 1;

  #connections = value of digit;
  #foreign connections = 0;

  if digit has at least one connection {
    connect digit to the following digit;
    connect following digit to digit;
    increment both digit's connection amounts;
    decrement #connections;
    #foreign connections = value of following digit;
  }

  for all digits from (digit+2) to last digit of code AND while #connections>0 {

    if there are no #foreign connections {
      connect original digit to current digit;
      connect current digit to original digit;
      increment both digit's connection amounts;
      decrement #connections;
      #foreign connections += value of current digit;
    }

    if there are #foreign connections {
      ignore current digit;
      #foreign connections += value of (current digit);
      decrement #foreign connections;
    }
  }
}
```

## Algorithm C: examination

An isomer is unique, if there is no identical canonical representation within all existing isomers. To provide the canonical representation that is easy to compare, Morgan's algorithm is used. Since the values of the zeroth iteration are already inserted into *morgans* by algorithm B, the iteration process can be done with *morgans* and the connectivity table. Unlike the Morgan's algorithm, the iteration does not terminate if the amount of unique values is constant. Instead it terminates after (n/3)+1 iterations which is enough to precisely identify an isomer, because at least 1/3 of the entire structure has been taken into account. After the last iteration, each atom has been assigned a specific value at its place in *morgans*.

### Algorithm C: Morgan's Algorithm (pseudocode)

```
initialize temporary array;
#iterations = 0;

while #iterations <= (n/3) {

  for all digits in morgans at current isomer {
    copy value into temporary array;
  }

  for all digits in morgans at current isomer {
    value in morgans at current digit = 0;

    for every atom this digit is connected to {
      morgans at current digit += value of atom in temporary array;
    }
  }
  increment #iterations;
}
```

The values in *morgans[isomer][1..n]* are then sorted by value using insertion sort to provide an easy way to compare them. The comparison is done with the isomer at the last position, so it acts as a sentinel if it is unique. Therefore, there will always be an identical isomer found. If the found duplicate is at the isomer's position in *morgans* (it has found itself), it is unique.

### Algorithm C: comparison (pseudocode)

```
duplicate found? = false;

while there is no duplicate found {

  while the last digit has not been reached and the values are identical {
    check next digit;
  }

  if the last digit has been compared and it is identical {
    duplicate found? = true;
  }
}

if the found isomer is at the last position in morgans {
  return isomer is unique;
} else {
  return isomer is not unique
}
```

The information on the isomer's uniqueness is passed to the generating algorithm, which then either saves this isomer and enumerates it as a valid isomer, or discards it entirely.

## Program structure

In order to combine all three algorithms, the program loops over n from 2 to 20 in the following fashion:

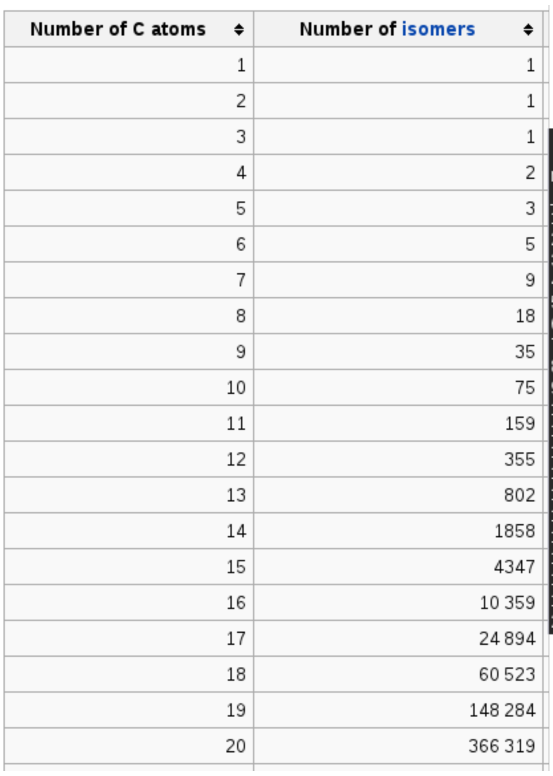
**main structure (pseudocode)**

```
initialize isomer vector;

declare methane;

for n from 2 to 20 {
  generate isomers with n atoms based on previous alkane's isomers {
    check each isomer for uniqueness {
      translate isomer;
      create morgans code;
      compare to existing isomer morgan codes {
        decide on uniqueness;
      }
    }
  }
  print amount of valid isomers for alkane n;
}
```

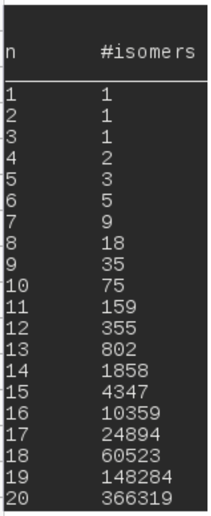
In addition to printing the number of valid isomers, the program may output all isomer codes in files for their (manual) reconstruction as well.

## Results and performance

The program is to be compiled using `g++ -O3 alkane_isomers.cc` when using the g++ compiler. Using this compiler and this optimization flag, the results were completely accurate and did not show any deviation from the reference values at all.
The performance was measured on my home computer, featuring an Intel Core i5-2500K with 4.2GHz and 8GB RAM running openSUSE Leap 42.1 64-bit on an internal hard drive. The time measured using the `time` command was 54min and 1s until completion. This result shows evidently how inefficient and slow the generation of isomers can become for large n. It is worth noting that the results for n<=19 were produced after about 7.5min. This underlines the exponential scaling for n that this algorithm unfortunately has.
The reference machine, a computer in the computer lab HCI D267 featuring an Intel Core



| Number of C atoms | Number of isomers |
|---:|---:|
| 1 | 1 |
| 2 | 1 |
| 3 | 1 |
| 4 | 2 |
| 5 | 3 |
| 6 | 5 |
| 7 | 9 |
| 8 | 18 |
| 9 | 35 |
| 10 | 75 |
| 11 | 159 |
| 12 | 355 |
| 13 | 802 |
| 14 | 1858 |
| 15 | 4347 |
| 16 | 10 359 |
| 17 | 24 894 |
| 18 | 60 523 |
| 19 | 148 284 |
| 20 | 366 319 |



```
n        #isomers
1        1
2        1
3        1
4        2
5        3
6        5
7        9
8        18
9        35
10       75
11       159
12       355
13       802
14       1858
15       4347
16       10359
17       24894
18       60523
19       148284
20       366319
```

Figure 5: *Left*: Reference values for the amount of isomers for alkanes from methane to icosane from Wikipedia [3].
*Right*: Amount of isomers for methane to icosane obtained by running the program. There is no deviation at all. The algorithm was accurate and precise.

i7-4770K with (presumably) 3.4GHz and 8GB RAM running Fedora 22 on (presumably) a network drive, was also tested. The time measured was 49min and 41s until completion. These results, albeit barely, are within the allowed 1h of CPU time that the challenge requires. To sum up, looking up the amount of isomers of a given alkane can be accomplished faster by searching the internet than running the algorithm for n>16. However, you wouldn't learn any C++ doing that and looking it up is too easy anyway.

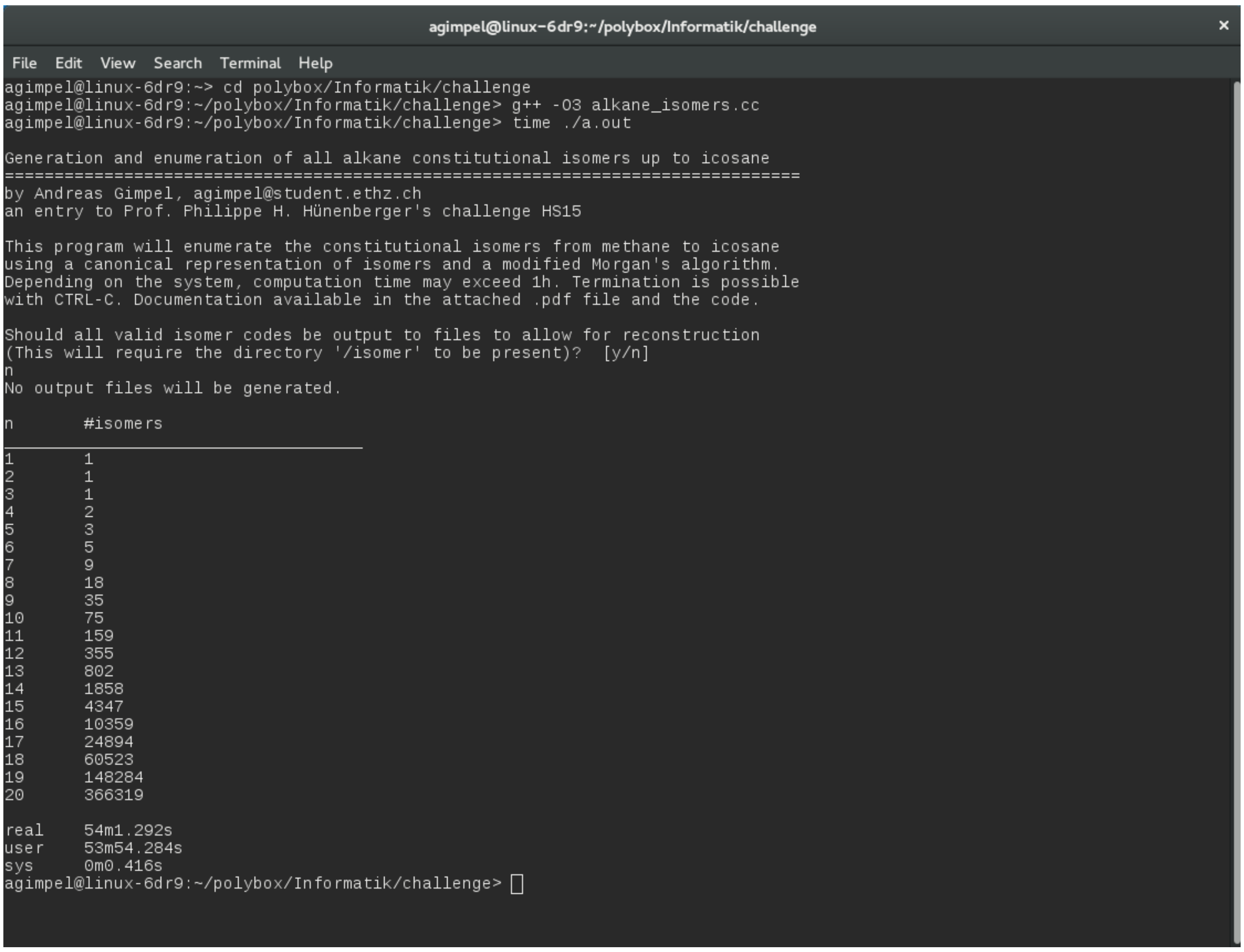
## References

[1] "Enumerating Molecules" by J.L. Faulon, D.P. Visco and D. Roe (p. 48ff)
Sandia National Laboratories, 2004
`http://prod.sandia.gov/techlib/access-control.cgi/2004/040960.pdf`
Note: The generation algorithm proposed was not used. Only the n-tuple code representation was modified and implemented.

[2] "Morgan Algorithmus-Tutorial" by Prof. Dr. J. Gasteiger, Dr. Th. Engel
CCC Univ. Erlangen, 2003
`http://www2.chemie.uni-erlangen.de/projects/vsc/chemoinformatik/erlangen/struktur_rep/code_2d/morgan_2.html`

[3] List of straight-chain alkanes
Wikipedia
`https://en.wikipedia.org/wiki/List_of_straight-chain_alkanes`

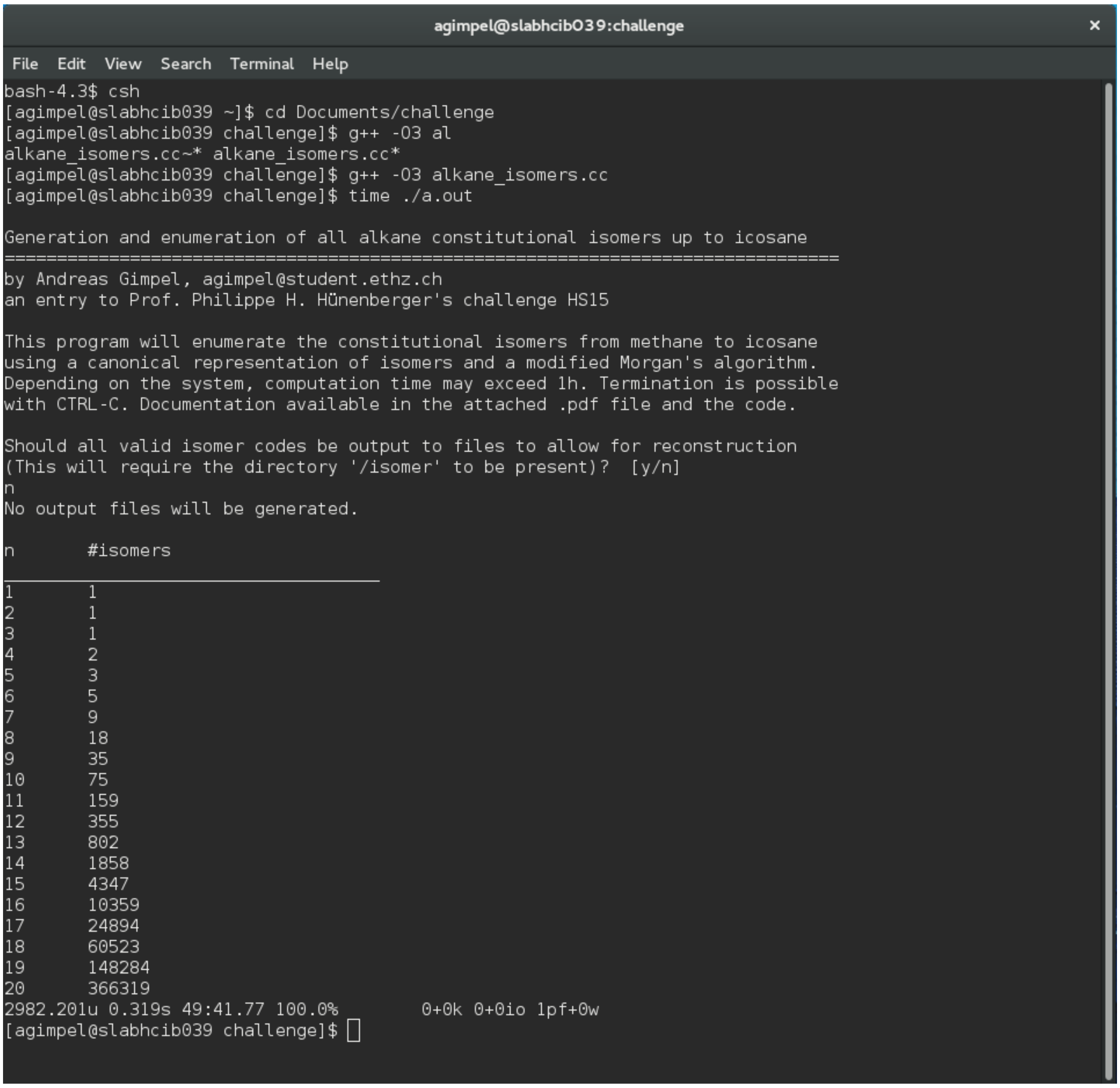
# Appendix


agimpel@linux-6dr9:~/polybox/Informatik/challenge

File Edit View Search Terminal Help

```
agimpel@linux-6dr9:~> cd polybox/Informatik/challenge
agimpel@linux-6dr9:~/polybox/Informatik/challenge> g++ -O3 alkane_isomers.cc
agimpel@linux-6dr9:~/polybox/Informatik/challenge> time ./a.out

Generation and enumeration of all alkane constitutional isomers up to icosane
================================================================================
by Andreas Gimpel, agimpel@student.ethz.ch
an entry to Prof. Philippe H. Hünenberger's challenge HS15

This program will enumerate the constitutional isomers from methane to icosane
using a canonical representation of isomers and a modified Morgan's algorithm.
Depending on the system, computation time may exceed 1h. Termination is possible
with CTRL-C. Documentation available in the attached .pdf file and the code.

Should all valid isomer codes be output to files to allow for reconstruction
(This will require the directory '/isomer' to be present)?  [y/n]
n
No output files will be generated.

n       #isomers
____________________________________
1       1
2       1
3       1
4       2
5       3
6       5
7       9
8       18
9       35
10      75
11      159
12      355
13      802
14      1858
15      4347
16      10359
17      24894
18      60523
19      148284
20      366319

real    54m1.292s
user    53m54.284s
sys     0m0.416s
agimpel@linux-6dr9:~/polybox/Informatik/challenge> 
```

Figure 6: Output on my home computer (Intel Core i5-2500K with 4.2GHz and 8GB RAM running openSUSE Leap 42.1 64-bit on an internal hard drive).



agimpel@slabhcib039:challenge

File Edit View Search Terminal Help

```
bash-4.3$ csh
[agimpel@slabhcib039 ~]$ cd Documents/challenge
[agimpel@slabhcib039 challenge]$ g++ -O3 al
alkane_isomers.cc~* alkane_isomers.cc*
[agimpel@slabhcib039 challenge]$ g++ -O3 alkane_isomers.cc
[agimpel@slabhcib039 challenge]$ time ./a.out

Generation and enumeration of all alkane constitutional isomers up to icosane
================================================================================
by Andreas Gimpel, agimpel@student.ethz.ch
an entry to Prof. Philippe H. Hünenberger's challenge HS15

This program will enumerate the constitutional isomers from methane to icosane
using a canonical representation of isomers and a modified Morgan's algorithm.
Depending on the system, computation time may exceed 1h. Termination is possible
with CTRL-C. Documentation available in the attached .pdf file and the code.

Should all valid isomer codes be output to files to allow for reconstruction
(This will require the directory '/isomer' to be present)?  [y/n]
n
No output files will be generated.

n       #isomers
____________________________________
1       1
2       1
3       1
4       2
5       3
6       5
7       9
8       18
9       35
10      75
11      159
12      355
13      802
14      1858
15      4347
16      10359
17      24894
18      60523
19      148284
20      366319
2982.201u 0.319s 49:41.77 100.0%        0+0k 0+0io 1pf+0w
[agimpel@slabhcib039 challenge]$ 
```

Figure 7: Output on the reference computer (Intel Core i7-4770K with (presumably) 3.4GHz and 8GB RAM running Fedora 22 on (presumably) a network drive).