

Vorbereiding Programmeerwedstrijden

najaar 2025

<https://liacs.leidenuniv.nl/~vlietrvan1/vbpw/>

Rudy van Vliet

kamer BM.2.03, Gorlaeus, tel. 071-527 2876

rvvliet(at)liacs(dot)nl

college 1, 2 september 2025

Introductie

Strings

Waarom dit vak?

- Verschillende typen programmeeropgaven onderscheiden
- Zorgvuldiger en kritischer leren programmeren
- Nieuwe algoritmes leren
- Beter presteren bij programmeerwedstrijden
- Fun!

Vierde keer

Programmeerwedstrijd

- LKP, BAPC, NWERC, EUC / ICPC (World Finals)
- 5 uur
- team van 3
- 1 computer
- $\pm$ 12 opgaven
- score op basis van
 - aantal opgaven **helemaal** 'goed'
 - tijdstip van goede oplossingen (+ straf tijd)
- standard input / output

BAPC 2024																final standings
RANK	TEAM		SCORE	A	B	C	D	E	F	G	H	I	J	K	L	M
1		Students Segfault go BRRRR Delft University of Technology	13 1468	16 1 try	107 3 tries	220 1 try	270 1 try	47 1 try	96 1 try	21 1 try	145 1 try	36 1 try	9 1 try	124 1 try	258 1 try	79 1 try
2		Team Biem Utrecht University	10 1202	65 2 tries	204 1 try	118 1 try		46 2 tries	86 1 try	38 1 try		68 1 try	12 1 try	255 7 tries		150 1 try
3		They cheated ^^^^^^^^^^^^^^^^^^^^^ Delft University of Technology	10 1773	105 2 tries	284 3 tries			28 2 tries	109 2 tries	60 1 try	153 1 try	58 1 try	4 1 try	298 15 tries	2 tries	234 4 tries
4		cmake Delft University of Technology	9 1018	61 1 try	244 2 tries			83 2 tries	184 1 try	93 1 try	2 tries	107 1 try	8 1 try	78 1 try	1 try	120 1 try
5		W2 Delft University of Technology	9 1113	51 1 try	75 1 try	9 tries		98 3 tries	198 3 tries	38 1 try		93 1 try	6 1 try	134 1 try		280 4 tries
6		S.L.O.N. Delft University of Technology	9 1234	42 1 try	232 1 try			67 1 try	117 2 tries	147 4 tries	12 tries	127 3 tries	8 1 try	200 1 try		174 1 try
7		Friday volleyball Universiteit van Amsterdam	9 1254	220 1 try	184 1 try	152 1 try		136 1 try	65 2 tries	85 2 tries		31 1 try	9 1 try	292 3 tries		
8		(J°□°)' (°_°)' (■_■--) Vrije Universiteit	8 1309	16 1 try	5 tries			133 2 tries	216 3 tries	123 2 tries		191 1 try	11 1 try	204 3 tries		295 1 try
9		ADA Refactor Universiteit van Amsterdam	7 828	23 1 try	225 3 tries			87 3 tries		40 1 try	5 tries	77 1 try	12 1 try	8 tries		244 3 tries
10		Rubber Duckie Debuggers Eindhoven University of Technology	7 884	61 1 try	246 1 try	4 tries		89 2 tries	182 1 try	108 1 try		126 3 tries	12 1 try	3 tries		
11		A random triennial Rijksuniversiteit Groningen	6 639	32 1 try		3 tries		114 3 tries	216 3 tries	50 1 try		140 1 try	7 1 try	5 tries		
12		BinaryChurch Université Catholique de Louvain	6 695	96 1 try	8 tries			92 2 tries	182 3 tries	63 1 try		191 1 try	11 1 try	4 tries		
13		Wormholes Leiden University	6 727	48 1 try				96 2 tries	173 1 try	70 1 try		273 3 tries	7 1 try	6 tries		
14		FC Puddlebee Eindhoven University of Technology	6 969	31 1 try				254 7 tries	220 1 try	112 4 tries		168 1 try	4 1 try			
15		De Wasberen Radboud University	6 1010	176 2 tries				108 3 tries	222 1 try	81 1 try		266 5 tries	17 1 try		6 tries	
16		Backedge boys Universiteit van Amsterdam	6 1029	153 1 try			1 try	248 2 tries	171 1 try	112 2 tries		272 2 tries	13 1 try		2 tries	
17		The Eastern Block Eindhoven University of Technology	6 1102	55 1 try	259 6 tries			283 2 tries		39 2 tries		299 2 tries	7 1 try			
18		Central Processing Unit Rijksuniversiteit Groningen	5 643	52 2 tries				227 2 tries	243 3 tries	28 1 try	1 try		13 1 try			

Toetsing

- programmeerwedstrijd, 4 uur
- datum: 20-24 oktober 2025
- individueel
- vier of vijf opgaven
- 2.5 punt per opgave
- tweede poging: 2.0 punt
- derde of latere poging: 1.5 punt
- aftrek: -0.25 als niet binnen een uur
- als niet goed: percentage van 1.5 punt (algoritmisch)
- DOMjudge (vanuit LIACS)

Toetsing

- huiswerkopgaven (4 weken: 1 a 2 per week)
- individueel
- 2.5 punt per opgave (genormeerd naar totaal 10)
- tweede poging: 2.0 punt
- derde of latere poging: 1.5 punt
- als niet goed: percentage van 1.5 punt
- DOMjudge (vanuit LIACS)

Toetsing

- 80% * programmeerwedstrijd + 20% * huiswerkopgaven
- bonus bij LKP2025 (zaterdag 27 september) / BAPC2025 (zaterdag 25 oktober)
 - 0.5 punt als bij bovenste kwart
 - 0.25 punt als bij tweede kwart
- eindcijfer ≤ 10

Gehaald als

- cijfer inclusief bonus ≥ 5.5
- én cijfer programmeerwedstrijd ≥ 5
- 2 EC extracurriculair

College / boek

- hoorcollege: dinsdag, 11.00–12.45 (BW.0.05 / DM.1.09)
- werkcollege (practicum):
donderdag, 09.00–10.45 (DM.0.17)
- zeven weken
- Steven S. Skiena & Miguel A. Revilla
Programming Challenges – The programming contest training manual
- hoofdstukken 3, 6, 7, 8, 9, 10, 13 (deels, en meer)

Online judge

<https://onlinejudge.org>

- register and confirm
- Remember me
- Browse problems (from book)
- submit / submission ID
- Where's Waldorf? (3.8.2)

DOMjudge

`http://domjudge.liacs.nl/public` (vanuit LIACS)

- Register now (username, team name, password)
- If necessary, ask me to add you to this course's contests
- Choose contest
- Submit
- Hello World! (VbPw2025-0)

Website

- <https://liacs.leidenuniv.nl/~vlietrvan1/vbpw/>
- slides, behandelde stof

Brightspace

- cijfers, email

2.3. Read problem statement carefully

Flying Safely

Source: BAPC2013

2.3. Read problem statement carefully

- extract essential information
- in particular, input/output specification
- sample input/output, but . . .
- estimate required efficiency (maximum input size)

3.1. Character Codes

ASCII

- numbers
- 0, ..., 127
- 48, ..., 57 $\approx$ '0', ..., '9'
- 65, ..., 90 $\approx$ 'A', ..., 'Z'
- 97, ..., 122 $\approx$ 'a', ..., 'z'
- 1 byte in C/C++

3.1. Character Codes

Advantages of Sequential placement
(perhaps used with Algoritmiëk 2025, assignment 2)

- iterate through letters: from 'a' to 'z'
- determine rank of letter: 'C' - 'A'
- convert upper case to lower case v.v.: 'C' - 'A' + 'a'
- char x is uppercase, if and only if ...

Alphabetical order: "aa" < "AB"

3.2. Representing Strings

- null terminated char array
- class string with member functions (like size)
- linked list of char's

Choice of Representation

- amount of space
- constraints on string represented
- constant-time access to i 'th character
- efficient check if i 'th character is within string
- efficient deletion / insertion
- maximum length (un)specified

3.3. Corporate Renamings

Replace exact matchings of corporate names in text

Input:

4

"Anderson Consulting" to "Accenture"

"Enron" to "Dynegy"

"DEC" to "Compaq"

"TWA" to "American"

5

Anderson Accounting begat Anderson Consulting, which offered advice to Enron before it DECLARED bankruptcy, which made Anderson Consulting quite happy it changed its name in the first place!

4

"Anderson Consulting" to "Accenture"

"Enron" to "Dynegy"

"DEC" to "Compaq"

"TWA" to "American"

5

Anderson Accounting begat Anderson Consulting, which
offered advice to Enron before it DECLARED bankruptcy,
which made Anderson
Consulting quite happy it changed its name
in the first place!

Output:

Anderson Accounting begat Accenture, which
offered advice to Dynegy before it CompaqLARED bankruptcy,
which made Anderson
Consulting quite happy it changed its name
in the first place!

Specification details

- at most 100 corporate changes
- at most 1000 characters on line of text

Representation. . .

Input:

4

"Anderson Consulting" to "Accenture"

"Enron" to "Dynegy"

"DEC" to "Compaq"

"TWA" to "American"

5

Anderson Accounting begat Anderson Consulting, which offered advice to Enron before it DECLARED bankruptcy, which made Anderson Consulting quite happy it changed its name in the first place!

Representation - Char array

```
const int MAXLEN = 1000;    // longest possible string
const int MAXCHANGES = 100; // maximum number of name changes

typedef char mystring[MAXLEN+1];

mystring mergers[MAXCHANGES][2]; // store before/after company names
```

Representation - String

```
const int MAXCHANGES = 100;    // maximum number of name changes  
string mergers[MAXCHANGES][2]; // store before/after company names
```

Read changes. . .

4

"Anderson Consulting" to "Accenture"

"Enron" to "Dynegy"

"DEC" to "Compaq"

"TWA" to "American"

5

Anderson Accounting begat Anderson Consulting, which offered advice to Enron before it DECLARED bankruptcy, which made Anderson Consulting quite happy it changed its name in the first place!

Read changes - Char array

```
void read_changes (int &nmergers)
{ int j;  // counter

    scanf ("%d\n", &nmergers);
    for (j=0;j<nmergers;j++)
    { read_quoted_string (mergers[j][0]);
      read_quoted_string (mergers[j][1]);
    }

}  // read_changes
```

Read changes - Char array

Without checks...

```
void read_quoted_string (char *s)
{
    int i = 0;  // counter
    char c;  // latest character

    while ((c=getchar()) != '\"');
    while ((c=getchar()) != '\"')
    { s[i] = c;
      i ++;
    }
    s[i] = '\\0';
}  // read_quoted_string
```

Read changes - String

```
void read_changes (int &nmergers)
{ ...    // declarations j, mergerline, endpos

    cin >> nmergers;
    getline (cin, mergerline);    // to get to the line following nmergers

    for (j=0;j<nmergers;j++)
    { getline (cin, mergerline);
      endpos = read_quoted_string (mergers[j][0], mergerline, 0);
      if (endpos!=string::npos)
          endpos = read_quoted_string (mergers[j][1], mergerline, endpos+1);

      // error message, if applicable
    }

} // read_changes
```

Read changes - String

```
size_t read_quoted_string (string &name, string mergerline,
                           size_t beginpos)
{ size_t endpos;

  beginpos = mergerline.find_first_of ("\\", beginpos);
  if (beginpos!=string::npos)
  { endpos = mergerline.find_first_of ("\\", beginpos+1);
    if (endpos!=string::npos)
    { name = mergerline.substr (beginpos+1, endpos-beginpos-1);
      // beginpos+1, because we do not store the quotes themselves
    }
  }
  else
    endpos = string::npos;

  return endpos;
} // read_quoted_string
```

3.4. Searching for patterns...

4

"Anderson Consulting" to "Accenture"

"Enron" to "Dynegy"

"DEC" to "Compaq"

"TWA" to "American"

5

Anderson Accounting begat Anderson Consulting, which offered advice to Enron before it DECLARED bankruptcy, which made Anderson Consulting quite happy it changed its name in the first place!

Searching for patterns - Char array

```
int findmatch (char *p, char *t)
{ ...    // declarations i, j, plen, tlen

    plen = strlen (p);
    tlen = strlen (t);

    for (i=0;i<=(tlen-plen);i++)
    { j = 0;
      while ((j<plen) && (t[i+j]==p[j]))
        j++;

      if (j==plen)
        return i;
    } // for i

    return -1;
} // findmatch
```

Searching for patterns - String

```
beginpos = s.find (mergers[j][0]))
```

No KMP, either

No need for KMP, anyway

3.5. Manipulating strings. . .

- computing length
- copying string
- reversing string
- replacing substring. . .

Replacing substring - Char array

```
void replace_x_with_y (char *s, int pos, int xlen, char *y)
{ ... // declarations i, slen, ylen

    slen = strlen (s);    ylen = strlen (y);

    if (xlen>=ylen) // shift suffix to the left
    { for (i=pos+xlen;i<=slen;i++) // including EOS
        s[i+(ylen-xlen)] = s[i];
    }
    else // shift suffix to the right
    { for (i=slen;i>=pos+xlen;i--) // including EOS
        s[i+(ylen-xlen)] = s[i];
    }

    for (i=0;i<ylen;i++) // insert y into s
        s[pos+i] = y[i];
} // replace_x_with_y
```

Replacing substring - String

```
prefix = s.substr (0, beginpos);  
beginpos2 = beginpos + mergers[j][0].size();  
suffix = s.substr (beginpos2, s.size()-beginpos2);  
s = prefix + mergers[j][1] + suffix;
```

3.6. Main - String

```
for (int i=0;i<nlines;i++)
{ getline (cin, s);

    for (int j=0;j<nmergers;j++)
    {
        while ((beginpos = s.find (mergers[j][0])) != string::npos)
        { // we found an occurrence, starting at beginpos
            prefix = s.substr (0, beginpos);
            beginpos2 = beginpos + mergers[j][0].size();
            suffix = s.substr (beginpos2, s.size()-beginpos2);
            s = prefix + mergers[j][1] + suffix;
        } // while

    } // for j

    cout << s << endl;
} // for i
```

Problems with problem statement

...

Problems with problem statement

- company name split between lines
- overlapping corporate names (even with same name)
- new name may contain old name
- subsequent changes
- cyclic changes
- length of corporate names may be more than 1000
- length of new text line may be more than 1000

1.3. Programming Hints

- write comments first
- document each variable
- use symbolic constants
- use enumerated types for a reason
- use subroutines to avoid redundant code...

```

while (c!='0')
{ cin >> c;
  if (c == 'A')
  { if (row-1 >= 0)
    { temp = b[row-1][col];
      b[row-1][col] = ' ';
      b[row][col] = temp;
      row = row-1;
    }
  }
  else if (c=='B')
  { if (row+1 <= BOARDSize-1)
    { temp = b[row+1][col];
      b[row+1][col] = ' ';
      b[row][col] = temp;
      row = row+1;
    }
  }
  ...
}

```

2.3. Going to War

- 52 playing-cards
A, K, Q, J, T, 9, 8, 7, 6, 5, 4, 3, 2
s, h, d, c
- rules...

Sample input

```
4d Ks As 4h Jh 6h Jd Qs Qh 6s 6c 2c Kc 4s Ah 3h Qd 2h 7s 9s 3c 8h Kd
8d 8c 9c 7c 5d 4c Js Qc 5s Ts Jc Ad 7d Kh Tc 3s 8s 2d 2s 5h 6d Ac 5c
```


2.4. Hitting the Deck

representation for (packets of) cards...

2.4. Hitting the Deck

representation for (packets of) cards: two queues of:

- pairs of characters / strings of length 2...
- pairs of numbers
- only value numbers...
- values of **ranking function**

Ranking function

```
const int NCARDS = 52; // number of cards
const int NSUITS = 4   // number of suits
char values[] = "23456789TJQKA";
char suits[] = "cdhs";

int rank_card (char value, char suit)
{ int i, j;    // counters

    for (i=0; i<(NCARDS/NSUITS); i++)
        if (values[i]==value)
            for (j=0; j<NSUITS; j++)
                if (suits[j]==suit)
                    return (i*NSUITS + j);

    cout << "Warning: bad input value=" << value
          << ", suit=" << suit << endl;
}
```

Unranking functions

```
char suit (int card)
{
    return (suits[card % NSUITS]);
}
```

```
char value (int card)
{
    return (values[card / NSUITS]);
}
```

```
int intvalue (int card)
{
    return (card / NSUITS);
}
```

Ranking / unranking functions also useful for other combinatorial objects.

2.7. Testing and Debugging

- test given input
- test incorrect input (if necessary)
- test boundary conditions,
e.g., empty input, one item, values that are zero
- test instances where you know answer
- test big instances

2.7. Testing and Debugging

- get to know debugger
- display non-trivial datastructures
- test invariants rigorously...

```
for (i=0; i<NCARDS; i++)  
    if (i != rank_card (value(i), suit(i)))  
        cout << "Error: rank card(" << value(i) << "," << suit(i) << ")=" <<  
            << rank_card (value(i), suit(i)) << " not " << i << endl;
```

2.7. Testing and Debugging

- make your print statements mean something
- make your arrays a little larger

3.8.2. Where's Waldorf?

3.8.2. Where's Waldorf?

- representation grid
- base 1
- upper case / lower case
- multiple occurrences of word (≥ 1)
- blank line before every input and between consecutive outputs
- algorithm...
- subroutine for searching in one direction
- representation directions

3.8.3. Common Permutation

3.8.3. Common Permutation

- understand the problem
- algorithm...
- boundary case...

1.6.1. The $3n + 1$ Problem

1.6.1. The $3n + 1$ Problem

- brute force fast enough?
- store and re-use values computed before (upto a certain maximum?)

3.8.6. File Fragmentation

3.8.6. File Fragmentation

- two fragments per file
- algorithm...