Informatik I - Übung 6

Pascal Schärli

pascscha@student.ethz.ch

29.03.2019

ETH zürich

Was gibts heute?

- Nachbesprechung
- Best-Of Vorlesung:
  - Floats
  - Vectors
  - Referenzen
  - Constants
  - Characters
- Vorbesprechung

Nachbesprechung

Pascal Schärli *29.03.2019*

Bild: look again by corpus delicti from the Noun Project

ETH zürich

Round

```
// PRE:  x is roundable to a number in
//        the value range of type int
// POST: return value is the integer
//        nearest to x, or the one
//        further away from 0 if x lies
//        right in between two integers.
int round (double x) {
    if(x > 0){
      return x + .5;
    }
    else{
      return x - .5;
    }
}
```

Best of Vorlesung

Floats

$$F * (\beta, p, e_{min}, e_{max})$$
$$\beta = 2$$
$$p = 3$$
$$e_{min} = -4$$
$$e_{max} = 4$$

(10 + 0.5) + 0.5

	dez	**bin**
	10	1.01 * 2^3
+	0.5	0.0001 * 2^3
=	10	1.01 * 2^3
+	0.5	0.0001 * 2^3
=	10	1.01 * 2^3

Floats

$$F * (\beta, p, e_{min}, e_{max})$$
$$\beta = 2$$
$$p = 3$$
$$e_{min} = -4$$
$$e_{max} = 4$$

(0.5 + 0.5) + 10

	dez	**bin**
	0.5	1.00 * 2^-1
+	0.5	1.00 * 2^-1
=	1	1.00 * 2^0
+	10	1010 * 2^0
=	12	1.10 * 2^3

Vektoren

- Vektoren helfen in C++ Listen von Elementen abzuspeichern und zu verwalten.

- std::vector ist *kein* Primitiver Datentyp wie int oder bool
  $\rightarrow$ sie haben noch weiter Funktionalität

- Vektoren müssen zuerst mit #include <vector> aus der Standardbibliothek geladen werden.

Vektoren

- Beim erstellen von einem neuen Vektor muss immer noch der zu speichernde Datentyp angegeben werden.
- Bei Bedarf kann auch die Anzahl Elemente oder der Initialisierungswert gegeben werden.

Beispiele:

```
std::vector<int> a;           // empty vector of ints

std::vector<float> b(5);      // five floats with value 0

std::vector<double> c(5,3);   // five doubles with value
```

Vektoren

- Mit der Funktion *at* kann man auf Elemente im Vektor zugreifen.
- Wir haben Indizierung bei 0, das heisst das vorderste Element hat den Index 0.

```cpp
#include <vector>
#include <iostream>

int main(){
    std::vector<int> values(5);
    for(int i = 0; i < 5; i++){
        values.at(i) = i * i;
    }

    // [..]
```

```
values -> [0, 1, 4, 9, 16]
```

```cpp
  // [...]

  std::cout << values.at(3) << std::endl;
  values.at(0) = 5;
  std::cout << values.at(0) << std::endl;

  return 0;
}
```

```
Output:
3
5
```

Vektoren

Vektoren sind *keine* Primitive Datentypen. Sie haben noch weitere Funktionalitäten:

Name	Rückgabe-Typ	Funktionalität
`size`	`unsigned int`	Grösse (Länge) vom Vektor
`empty`	`bool`	true falls Vektor leer ist
`push_back`	`void`	Element am Ende hinzufügen
`pop_back`	`void`	Löscht letztes Element
`back`	`T`	Letztes Element im Vektor

Eine komplette Liste findet ihr hier unter "Member Functions"

Vektoren

```cpp
#include <iostream>
#include <vector>

int main(){

    std::vector<int> test(5);
    test[1] = test.size();
    test.pop_back();
    test[2] = test.size();
    test.push_back(2);
    test[3] = test.empty();

    for(unsigned int i = 0; i < test.size(); i++){
        std::cout << test.at(i) << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

```
0 5 4 0 2
```

Vektoren

```
0 0 0 0 0
```

```
0 5 0 0 0
```

```
0 5 0 0
```

```
0 5 4 0
```

```
0 5 4 0 2
```

```
0 5 4 0 2
```

```
std::vector<int> test(5);
test.at(1) = test.size();
test.pop_back();
test.at(2) = test.size();
test.push_back(2);
test.at(3) = test.empty();
```

Vektoren

Indizierung bei 0 nicht vergessen! Bei Zugriff auf Daten, welche nicht existieren gibt es einen Fehler.

```
std::vector<int> v(5, 0);

v.at(5) = 3;

for(unsigned int i = 0; i < v.size(); i++){
    std::cout << v.at(i) << " ";
}
```

```
terminate called after throwing an instance
of 'std::out_of_range'
```

UPPER CASE

Skizziert ein Programm, welches einen gegebenen Text in Grossbuchstaben umwanden kann:

```
H3llo World!
```

```
H3LLO WORLD!
```

```
#include <iostream>
#include <vector>
#include <ios>

// POST: Converts lowercase letters a-z to upper case A-Z
void char_to_upper(char& letter) {
  // TODO
}

// POST: Converts all letters to upper-case.
void to_upper(std::vector<char>& letters) {
  // TODO
}

int main () {
  // Say to std::cin not to ignore whitespace.
  std::cin >> std::noskipws;

  std::vector<char> letters;
  char ch;

  // Step 1: Read input.
  do {
    std::cin >> ch;
    letters.push_back(ch);
  } while (ch != '\n');

  // Step 2: Convert to upper-case.
  // TODO

  // Step 3: Output.
  // TODO

  return 0;
}
```

UPPER CASE

```
// POST: Converts the letter to upper case.
void char_to_upper(char& letter) {
  if ('a' <= letter && letter <= 'z') {
    letter -= 'a' - 'A';  // 'a' > 'A'
  }
}
```

```
// POST: Converts all letters to upper-case.
void to_upper(std::vector<char>& letters) {
  for (unsigned int i = 0; i < letters.size(); ++i) {
    char_to_upper(letters[i]);
  }
}
```

UPPER CASE

```
int main () {
  // Say to std::cin not to ignore whitespace.
  std::cin >> std::noskipws;

  std::vector<char> letters;
  char ch;

  // Step 1: Read input.
  do {
    std::cin >> ch;
    letters.push_back(ch);
  } while (ch != '\n');

  // Step 2: Convert to upper-case.
  to_upper(letters);

  // Step 3: Output.
  for (unsigned int i = 0; i < letters.size(); ++i) {
    std::cout << letters[i];
  }
  return 0;
}
```

Referenzen

In C++ können wir existierende Variablen "Verlinken"

```
int a = 3;
int& b = a;
b = 2;
std::cout << a;
```

```
2
```

Referenzen

Call by Value

```cpp
#include <iostream>

void foo(int a, int b){
    int temp = a;
    a = b;
    b = temp;
}

int main(){
    int a = 5;
    int b = 7;
    foo(a, b);
    std::cout << a << " " << b;
    return 0;
}
```

```
5 7
```

Call by Reference

```cpp
#include <iostream>

void foo(int& a, int& b){
    int temp = a;
    a = b;
    b = temp;
}

int main(){
    int a = 5;
    int b = 7;
    foo(a, b);
    std::cout << a << " " << b;
    return 0;
}
```

```
7 5
```

Referenzen

Warum braucht es Referenzen überhaupt?

- Referenzen erlauben uns mehrere Werte zurückzugeben in einer Funktion.

```
void scale(double& x, double& y, double amount){
    x *= amount;
    y *= amount;
}
```

- Die Übergebenen Parameter müssen nicht kopiert werden.

```
void read_i (Vector& v, unsigned int i);
```

- Bestimmte Dinge können nicht kopiert werden.

```
std::ostream o = std::cout;
o << "It works!\n";
```

```
error
```

```
std::ostream& o = std::cout;
o << "It works!\n";
```

```
It works!
```

Referenzen

Der Rückgabetyp von Funktionen kann auch eine Referenz sein:

```cpp
#include <iostream>

int square(int n){
    return n*=n;
}

int main(){
    int n = 3;
    square(square(n));

    std::cout << n;
    return 0;
}
```

```
3
```

```cpp
#include <iostream>

int square(int& n){
    return n*=n;
}

int main(){
    int n = 3;
    square(square(n));

    std::cout << n;
    return 0;
}
```

```
error: cannot bind
non-const lvalue
reference of type ‘int&’
to an rvalue of type ‘int’
```

```cpp
#include <iostream>

int& square(int& n){
    return n*=n;
}

int main(){
    int n = 3;
    square(square(n));

    std::cout << n;
    return 0;
}
```

```
81
```

Characters

Ein Character ist ein Buchstabe, welchen in C++ als 8-Bit Zahl gespeichert wird. Die Buchstaben sind mit ASCII Codiert.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Characters

```
char c1 = 'i';
std::cout << c1;
```

```
i
```

```
char c2 = 110;
std::cout << c2;
```

```
n
```

```
char c3 = c1 - 3;
std::cout << c3;
```

```
f
```

```
char c4 = c1 - 'a' + 'A';
std::cout << c4;
```

```
I
```

Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
64	40	100	@	@	96	60	140	`	`
65	41	101	A	A	97	61	141	a	a
66	42	102	B	B	98	62	142	b	b
67	43	103	C	C	99	63	143	c	c
68	44	104	D	D	100	64	144	d	d
69	45	105	E	E	101	65	145	e	e
70	46	106	F	F	102	66	146	f	f
71	47	107	G	G	103	67	147	g	g
72	48	110	H	H	104	68	150	h	h
73	49	111	I	I	105	69	151	i	i
74	4A	112	J	J	106	6A	152	j	j
75	4B	113	K	K	107	6B	153	k	k
76	4C	114	L	L	108	6C	154	l	l
77	4D	115	M	M	109	6D	155	m	m
78	4E	116	N	N	110	6E	156	n	n
79	4F	117	O	O	111	6F	157	o	o
80	50	120	P	P	112	70	160	p	p
81	51	121	Q	Q	113	71	161	q	q
82	52	122	R	R	114	72	162	r	r
83	53	123	S	S	115	73	163	s	s
84	54	124	T	T	116	74	164	t	t
85	55	125	U	U	117	75	165	u	u
86	56	126	V	V	118	76	166	v	v
87	57	127	W	W	119	77	167	w	w
88	58	130	X	X	120	78	170	x	x
89	59	131	Y	Y	121	79	171	y	y
90	5A	132	Z	Z	122	7A	172	z	z
91	5B	133	[	[	123	7B	173	{	{
92	5C	134	\	\	124	7C	174			|
93	5D	135	]	]	125	7D	175	}	}
94	5E	136	^	^	126	7E	176	~	~
95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Reverse Vector

Skizziert ein Programm, welches einen gegebenen Vektor umdrehen kann.


```
// POST: the targets of i and j got their
//       values swapped
void swap (int& i, int& j) {
    // TODO
}

// PRE:  length is the real length of the
//       vector sequence.
// POST: Reverses the sequence stored in
//       the vector sequence.
void reverse(std::vector<int>& sequence,
             unsigned int length) {
    // TODO
}
```

```
int main () {
  unsigned int length;
  std::vector<int> sequence (10);

  // Step 1: Read input.
  std::cin >> length;
  if (length > 10) {
    std::cout << "Bad input" << std::endl;
    return 0;
  }

  // TODO: Read Vector from std::cin

  // Reverse sequence;
  reverse(sequence, length);

  // TODO: Output Vector to std::cout

  return 0;
}
```

Reverse Vector

```
// POST: the targets of i and j got their values swapped
void swap (int& i, int& j) {
    const int tmp = i;
    i = j;
    j = tmp;
}
```

```
// PRE: length is the real length of the vector sequence.
// POST: Reverses the sequence stored in the vector sequence.
void reverse(std::vector<int>& sequence, unsigned int length) {
  if (length > 1) {
    int front = 0;
    int back = length-1;
    while (front < back) {
      swap(sequence.at(front), sequence.at(back));
      ++front;
      --back;
    }
  }
}
```

Reverse Vector

```
int main () {
  unsigned int length;
  std::vector<int> sequence (10);

  // Step 1: Read input.
  std::cin >> length;
  if (length > 10) {
    std::cout << "Bad input" << std::endl;
    return 0;
  }
  for (unsigned int i = 0; i < length; ++i) {
    std::cin >> sequence.at(i);
  }

  // Step 2: Reverse sequence;
  reverse(sequence, length);

  // Step 3: Output
  for (unsigned int i = 0; i < length; ++i) {
    std::cout << sequence.at(i) << " ";
  }
  std::cout << std::endl;

  return 0;
}
```

Referenzen

Was ist der Output von diesem Programm?

```
1 1 1 1 1
```

```
#include <iostream>

int foo (int a, int b) {
    a += b;
    return a;
}

int main() {
    int a = 0;
    int b = 1;
    for (int i=0; i<5; ++i) {
        b = foo (a, b);
        std::cout << b << " ";
    }
    return 0;
}
```

Referenzen

Was ist der Output von diesem Programm?

```
1 2 4 8 16
```

```
#include <iostream>

int foo (int& a, int b) {
    a += b;
    return a;
}

int main() {
    int a = 0;
    int b = 1;
    for (int i=0; i<5; ++i) {
        b = foo (a, b);
        std::cout << b << " ";
    }
    return 0;
}
```

Referenzen

Was ist der Output von diesem Programm?

```
1 1 1 1 1
```

```
#include <iostream>

int foo (int a, int& b) {
    a += b;
    return a;
}

int main() {
    int a = 0;
    int b = 1;
    for (int i=0; i<5; ++i) {
        b = foo (a, b);
        std::cout << b << " ";
    }
    return 0;
}
```

Const

Mit dem Keyword const kann man bestimmen, dass gewisse Variabeln Konstant sind, sich also nicht veränderen können.

- Das kann dazu führen, dass das Programm schneller läuft.
- Bei komplizierteren Funktionen kann es helfen um sicherzugehen, dass bestimmte Werte sicher nirgends verändert werden.

Compile-Error

```cpp
#include <iostream>

int foo(const int x){
        x = 5;
        return x;
}

int main(){

    std::cout << foo(3) << std::endl;

    return 0;
}
```

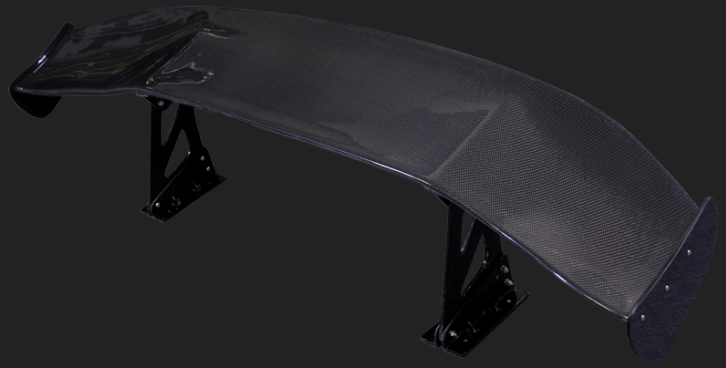
```
error: assignment of read-only parameter ‘x’
```

Runtime Error

```
#include <iostream>

int& foo(int x){
        x = 5;
        return x;
}

int main(){

    std::cout << foo(3) << std::endl;

    return 0;
}
```

```
Segmentation fault (core dumped)
```

Vorbesprechung



Pascal Schärli *29.03.2019*

ETH zürich

1: Const & Reference

```
T foo (S i)
{
    return ++i;
}
```

```
1. T = int,         S = int
2. T = int,         S = const int
3. T = int,         S = int&
4. T = int&,        S = int
5. T = const int&,  S = int&
```

1. Sind die Funktionen Semantisch Korrekt
   (Tipp: Compile Errors)
2. Falls ja, kann die Funktion auch fehlerfrei ausgeführt werden? *(Tipp: Runtime Errors)*
3. Falls ja, gebt eine möglichst Präzise Post-Condition an.

2: Number of Occurences

Schreibt folgendes Programm:

1. Lies so lange Integer ein, bis ein negativer Integer kommt und speichere diese (Ohne den negativen) in einem Vektor v
2. Lies eine weitere Zahl n ein
3. Zähle wie viele Male n im Vektor v vorkommt

Tipps:

- Strukturiert das Programm in Funktionen

```
void read_vector(std::vector<int> & values) {/*...*/}
int occurrences(const std::vector<int> & values, int toFind) {/*...*/}
```

- Benutzt keine Funktionen welche noch nicht besprochen wurden!

3: Long. Increasing Subseq.

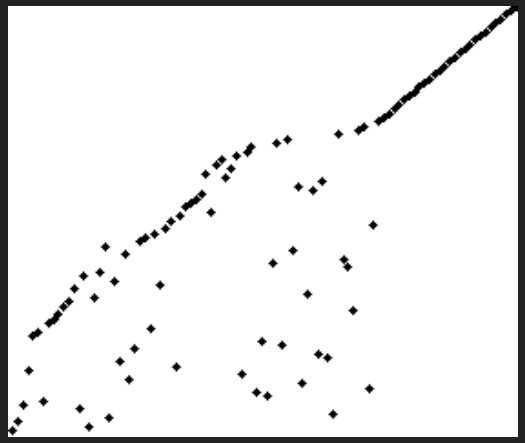
Schreibt folgendes Programm:

1. Lies so lange Integer ein, bis ein negativer Integer kommt und speichere diese (Ohne den negativen) in einem Vektor v
2. Bestimmt die Länge der längsten *streng* monoton steigenden Untersequenz.

Tipps:

Jetzt könnt ihr eure einlese-Funktion von der Aufgabe 2 wiederverwenden

4: Bubble Sort



Schreibt folgendes Programm:

1. Lies einen Integer, welcher die Länge des Vektors festlegt
2. Lies alle Werte vom Vektor ein
3. Sortiere diese Werte mit Bubble Sort

Tipps:

- Wenn ihr den Algorithmus nicht versteht gibt es online unzählige Erklärungen dazu
- Ihr müsst eine eigene Print-Funktion für den Vektor machen

Program of the Week

Viel Spass!


A'LA'IH, DO'NEH'LINI,
DO'NEH'LINI, A'LA'IH,
A'LA'IH, DO'NEH'LINI,
DO'NEH'LINI, DO'NEH'LINI,
A'LA'IH, A'LA'IH,
DO'NEH'LINI, A'LA'IH,
DO'NEH'LINI, DO'NEH'LINI,
DO'NEH'LINI . . .
FOR ADDED SECURITY, AFTER WE ENCRYPT THE DATA STREAM, WE SEND IT THROUGH OUR NAVAJO CODE TALKER.
... IS HE JUST USING NAVAJO WORDS FOR "ZERO" AND "ONE"?
WHOA, HEY, KEEP YOUR VOICE DOWN!