Exercise Week 06

GianAndrea Müller
mailto:muellegi@student.ethz

May 7, 2018

Exercise Week 06

GianAndrea Müller
mailto:muellegi@student.ethz

May 7, 2018

Time Schedule

- 10' Referenzen
- 15' Statische Arrays
- 15' Vektoren
- 5' Schriftzeichen

Time Schedule

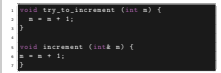
- 10' Referenzen
- 15' Statische Arrays
- 15' Vektoren
- 5' Schriftzeichen

Learning Objectives
- Kenntnis des Nutzens von Referenzen
- Verständnis der Grenzen des Arrays und Kenntnis möglicher Alternativen

Learning Objectives

- Kenntnis des Nutzens von Referenzen
- Verständnis der Grenzen des Arrays und Kenntnis möglicher Alternativen

Referenzen II

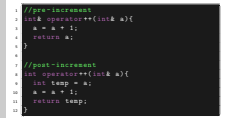


Referenzen II

- Wir haben letze Woche gesehen wie nützlich call by reference sein kann.
- Jetzt sehen wir uns als Beispiel einer weiteren Anwendung von Referenzen die Implementation der Inkrement-Operatoren an.

```
1 void try_to_increment (int m) {
2   m = m + 1;
3 }
4
5 void increment (int& m) {
6 m = m + 1;
7 }
```

Referenzen II

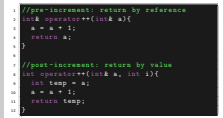


Referenzen II

- Das pre-increment gibt direkt die inkrementierte Variable als l-value zurück.
- Das post-increment gibt den Wert der nicht-inkrementierten Variable als r-value zurück.

```
1  //pre-increment
2  int& operator++(int& a){
3    a = a + 1;
4    return a;
5  }
6
7  //post-increment
8  int operator++(int& a){
9    int temp = a;
10   a = a + 1;
11   return temp;
12 }
```

Referenzen II

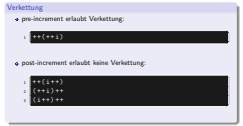


Referenzen II

- Wieso kann das post-increment nicht mit return by reference funktionieren?

```
1  //pre-increment: return by reference
2  int& operator++(int& a){
3    a = a + 1;
4    return a;
5  }
6
7  //post-increment: return by value
8  int operator++(int& a, int i){
9    int temp = a;
10   a = a + 1;
11   return temp;
12 }
```

- Das pre-increment erlaubt eine Verkettung, da es einen l-value zurückgibt, der wiederum als Funktionsargument aufgenommen werden kann.
- Beim post-increment funktioniert das nicht, da ein r-value nicht Funktionsargument der Inkrementierungsfuntion sein kann. Diese verlangt nämlich einen veränderlichen l-value.



Referenzen II

Verkettung

- pre-increment erlaubt Verkettung:

```
1 ++(++i)
```

- post-increment erlaubt keine Verkettung:

```
1 ++(i++)
2 (++i)++
3 (i++)++
```

Referenzen II
Verkettung
pre-increment erlaubt Verkettung:
++(++i) //funktioniert
post-increment erlaubt keine Verkettung:
++(i++) //funktioniert nicht
(++i)++ //funktioniert
(i++)++ //funktioniert nicht

Referenzen II

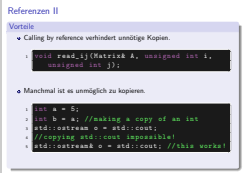
Verkettung

- pre-increment erlaubt Verkettung:

```
1 ++(++i) //funktioniert
```

- post-increment erlaubt keine Verkettung:

```
1 ++(i++) //funktioniert nicht
2 (++i)++ //funktioniert
3 (i++)++ //funktioniert nicht
```



Referenzen II

Vorteile

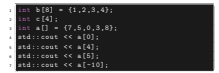
- Calling by reference verhindert unnötige Kopien.

```
1 void read_ij(Matrix& A, unsigned int i,
    unsigned int j);
```

- Manchmal ist es unmöglich zu kopieren.

```
1 int a = 5;
2 int b = a; //making a copy of an int
3 std::ostream o = std::cout;
4 //copying std::cout impossible!
5 std::ostream& o = std::cout; //this works!
```

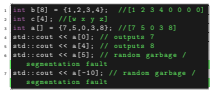
Statische Arrays



Statische Arrays

```
1 int b[8] = {1,2,3,4};
2 int c[4];
3 int a[] = {7,5,0,3,8};
4 std::cout << a[0];
5 std::cout << a[4];
6 std::cout << a[5];
7 std::cout << a[-10];
```

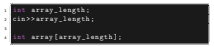
Statische Arrays



Statische Arrays

```
1 int b[8] = {1,2,3,4};  //[1 2 3 4 0 0 0 0]
2 int c[4]; //[w x y z]
3 int a[] = {7,5,0,3,8}; //[7 5 0 3 8]
4 std::cout << a[0]; // outputs 7
5 std::cout << a[4]; // outputs 8
6 std::cout << a[5]; // random garbage /
    segmentation fault
7 std::cout << a[-10]; // random garbage /
    segmentation fault
```

Statische Arrays

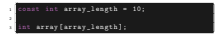


Statische Arrays

- Die Laenge eines Arrays muss vor Laufzeit definiert sein!
- Der Code kompiliert zwar auf Codeboard, generell ist das Verwenden einer nicht-konstanten Arraylänge verboten und wird auf anderen Compilern Kompilierfehler auslösen.

```
1 int array_length;
2 cin>>array_length;
3
4 int array[array_length];
```

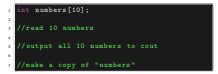
Statische Arrays



Statische Arrays

```
1 const int array_length = 10;
2
3 int array[array_length];
```

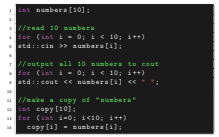
Exercise 06_1 ∼ 5'



Exercise 06_1 ∼ 5'

```
1 int numbers[10];
2
3 //read 10 numbers
4
5 //output all 10 numbers to cout
6
7 //make a copy of "numbers"
```

Solution 06_1 ∼ 5'



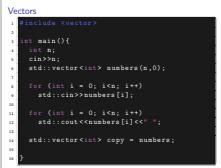
Solution 06_1 ∼ 5'

```cpp
int numbers[10];

//read 10 numbers
for (int i = 0; i < 10; i++)
std::cin >> numbers[i];

//output all 10 numbers to cout
for (int i = 0; i < 10; i++)
std::cout << numbers[i] << " ";

//make a copy of "numbers"
int copy[10];
for (int i=0; i<10; i++)
  copy[i] = numbers[i];
```

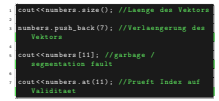


- Vektoren koennen ihre Laenge bei Laufzeit zugewiesen bekommen.
- Der Zugriff funktionert gleicht wie beim Array.
- Der Kopiervorgang ist viel einfacher als beim Array, da der operator= ueberladen ist...

Vectors

```
1  #include <vector>
2
3  int main(){
4    int n;
5    cin>>n;
6    std::vector<int> numbers(n,0);
7
8    for (int i = 0; i<n; i++)
9      std::cin>>numbers[i];
10
11   for (int i = 0; i<n; i++)
12     std::cout<<numbers[i]<<" ";
13
14   std::vector<int> copy = numbers;
15
16 }
```

Vectors



```
cout<<numbers.size(); //Laenge des Vektors

numbers.push_back(7); //Verlaengerung des
   Vektors

cout<<numbers[11]; //garbage /
   segmentation fault

cout<<numbers.at(11); //Prueft Index auf
   Validitaet
```

vector

Vectors

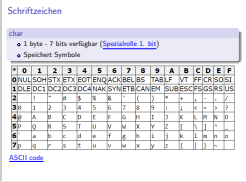
```
cout<<numbers.size(); //Laenge des Vektors

numbers.push_back(7); //Verlaengerung des
   Vektors

cout<<numbers[11]; //garbage /
   segmentation fault

cout<<numbers.at(11); //Prueft Index auf
   Validitaet
```

vector

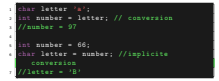

Schriftzeichen

char
- 1 byte - 7 bits verfügbar (Spezialrolle 1. bit)
- Speichert Symbole

*	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

ASCII code

Schriftzeichen



Schriftzeichen

- Zeichen können in c++ mit Hochkommas geschrieben werden.
- Die Zuweisung einer char zu einer int und umgekehrt bewirkt eine automatische / implizite Typenkonversion. Für die Konversion wird der ASCII Code verwendet.

```
1 char letter 'a';
2 int number = letter; // conversion
3 //number = 97
4
5 int number = 66;
6 char letter = number; //implicite
    conversion
7 //letter = 'B'
```

Schriftzeichen
Das Alphabet
(65)– 10 00001 = 'A'
(66)– 10 00010 = 'B'
(97)– 11 00001 = 'a'
(98)– 11 00010 = 'b'

Schriftzeichen

- Die Zeichen des Alphabets sind im ASCII Code so positioniert, dass durch Weglassen der ersten 2 bits direkt die Position des Buchstabens im Alphabet errechnet werden kann.

Das Alphabet

(65)–	10 00001 =	$'A'$
(66)–	10 00010 =	$'B'$
	⋮	
(97)–	11 00001 =	$'a'$
(98)–	11 00010 =	$'b'$