Computer Vision
and Geometry Lab

Informatik I for D-MAVT

Exercise Session 2

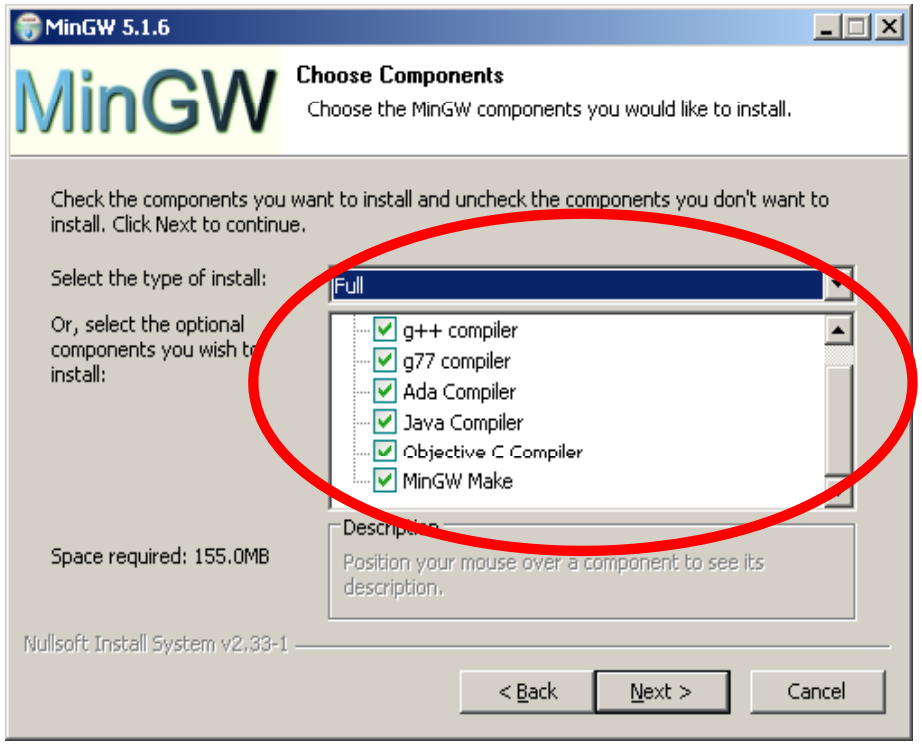
© Alexander Schwing, 2010

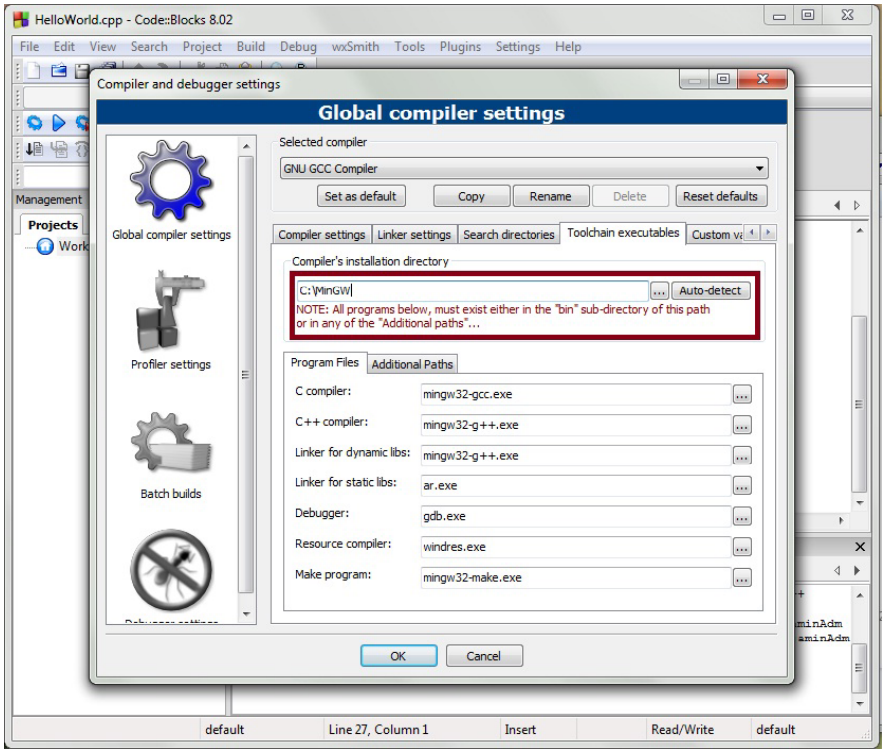
Organisatorisches

- Teaching Assistant
  - Alexander Schwing (aschwing@student.ethz.ch)
  - CAB G 89
  - Website: http://www.alexander-schwing.de/Info1DMaVt/
- Übungsabgabe
  - Auf Papier, keine Emails
    - Später wenn Programme umfangreicher werden, ev. auch Abgabe per Email möglich
- Testatbedingungen
  - 75% aller Übungen
    - Voraussichtlich 12 Serien (d.h. 9 gelöste Serien)

Probleme der letzten Übung

- Installation von MinGW





By courtesy of B. Imbach

- __DATE__ hat 2 Präfix- und Suffix-Unterstriche
- Jede Anweisung (Instruktion) endet mit einem Strichpunkt (;)

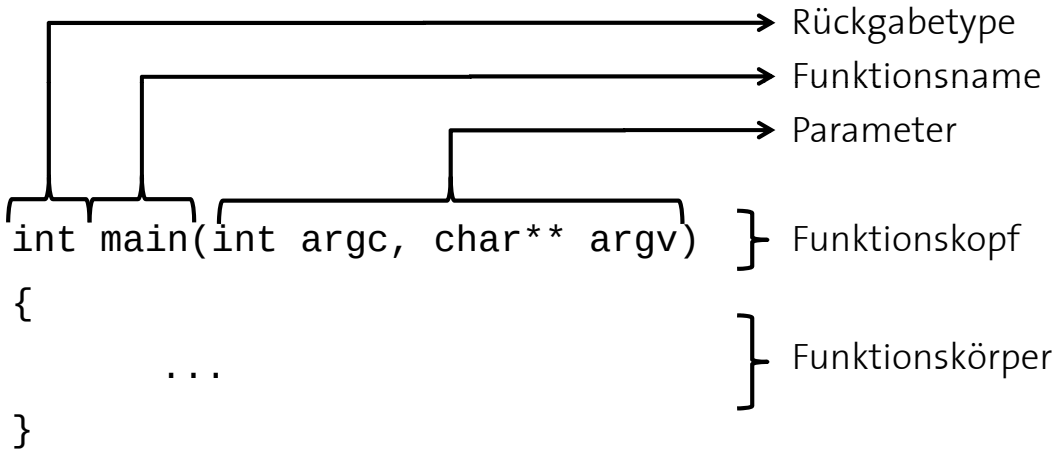
Repetition

- Ablauf beim Kompilieren:
  - Präprozessor (`#include`)
  - Kompilieren (maschinenlesbare Form)
  - Linken (Verbinden einzelner Bestandteile)
- Ablauf beim Ausführen
  - Laden des Programms in den Speicher (Betriebssystem)
  - Zeilenweises Abarbeiten ab **Funktion**

```
int main(int argc, char** argv) {
      ...
}
```

Funktion

- Auch: Subroutine, Subprogramm
- Aufbau:



Variablen

Basisdatentypen

	Name	Size [bytes on x86]	Range
Integral	`bool`	1	0 oder 1
	`char`	1	-2^7 bis 2^7-1 (signed) oder 0 bis 2^8-1 (unsigned)
	`short`	2	-2^{15} bis $2^{15}-1$ (signed) oder 0 bis $2^{16}-1$ (unsigned)
	`wchar_t`	2 bis 4	
	`int`	4	-2^{31} bis $2^{31}-1$ (signed) oder 0 bis $2^{32}-1$ (unsigned)
	`long`	4	wie `int`
	`long long`	8	-2^{63} bis $2^{63}-1$ (signed) oder 0 bis $2^{64}-1$ (unsigned)
Floating Point	`float`	4	+/- $1.175494351 \times 10^{-38}$ bis $3.402823466 \times 10^{+38}$
	`double`	8	+/- $2.2250738585072014 \times 10^{-308}$ bis $1.7976931348623158 \times 10^{308}$
	`long double`	8 bis 16	

Variablendefinition und Initialisierung

```
int Variable1, Variable2, Variable3 = 10, Variable4(20);
```

Variablen

- Gültigkeitsbereich (Scope) von Variablen

```
int a = 2, b = 3;
cout << a << " " << b;
{
        int a = 5;
        cout << a << " " << b;
}
cout << a;
```

Ausgabe:

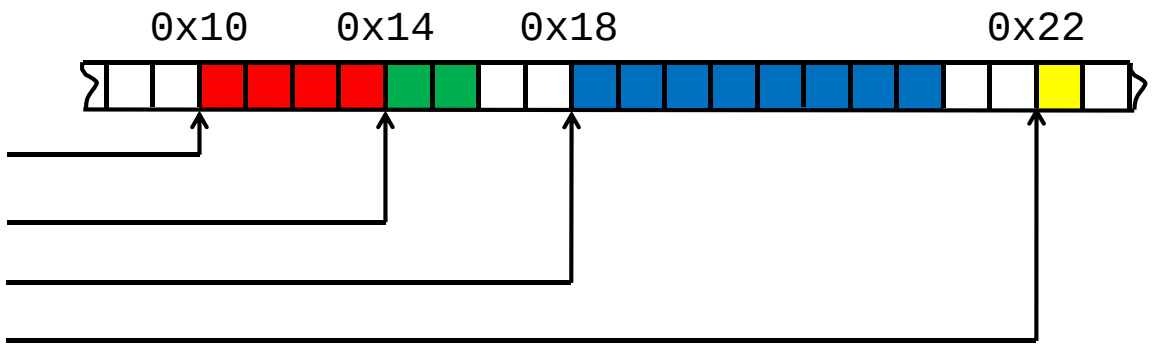
```
2 35 32
```

- Bits und Bytes
  - 1 byte = 8 bit (kleinste adressierbare Einheit)
  - 1 MiB (Mebibyte) = 1024 KiB (Kibibyte) = 1024 * 1024 byte
  - 1 MB (Megabyte) = 1000 kB (Kilobyte) = 1000 * 1000 byte
  - k bits codieren 2^k Zustände

Variablen

- Speicher

```
...
int Variable1;
short _123;
double VariAble1;
char Buch_stabe;
...
```



- Speicherbedarf bestimmen

Mit Operator: `size_t sizeof( {type-name} )`

Statement	Rückgabewert
`sizeof(short)`	2
`sizeof(double)`	8
`sizeof(wchar_t)`	bei mir: 2

```
#include <iostream>
int main() {
        size_t groesse = sizeof(wchar_t);
        std::cout << “Groesse: “ << groesse;
        std::cout << std::endl;
        return 0;
}
```

Variablen

Implizites Casting

```
int i1 = 1, i2 = 3;
double f1 = 8.98;
float f2 = 4.78;

int i3 = f1/f2;            // 8.98/4.78 = 1.87866...
float f3 = i3/i2;
```

Explizites Casting

```
int i1 = 1, i2 = 3;
double f1 = 8.98;
float f2 = 4.78f;

int i3 = int(f1/double(f2));            // 8.98/4.78 = 1.87866...
float f3 = float(i3)/i2;
```

Konstanten

```
const float sqrt2 = 1.414213562f;
```

Variablen

- Achtung:

```
...
int z = 1;
bool test_1 = z == 2;          //test_1 = false
bool test_2 = z = 3;           //test_2 = true
bool test_3 = z = 0;           //test_3 = false
bool test_4 = z == 1;          //test_4 = false
bool test_5 = z == 0;          //test_5 = true
...
```

Präprozessordirektiven

- Letzte Woche

```
#include <...>
```

- Neu

```
#define SQRT2 1.414213562
```

ASCII-Tabelle

Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
(nul)	0	0	0x00	(sp)	32	40	0x20	@	64	100	0x40	`	96	140	0x60
(soh)	1	1	0x01	!	33	41	0x21	A	65	101	0x41	a	97	141	0x61
(stx)	2	2	0x02	"	34	42	0x22	B	66	102	0x42	b	98	142	0x62
(etx)	3	3	0x03	#	35	43	0x23	C	67	103	0x43	c	99	143	0x63
(eot)	4	4	0x04	$	36	44	0x24	D	68	104	0x44	d	100	144	0x64
(enq)	5	5	0x05	%	37	45	0x25	E	69	105	0x45	e	101	145	0x65
(ack)	6	6	0x06	&	38	46	0x26	F	70	106	0x46	f	102	146	0x66
(bel)	7	7	0x07	'	39	47	0x27	G	71	107	0x47	g	103	147	0x67
(bs)	8	10	0x08	(	40	50	0x28	H	72	110	0x48	h	104	150	0x68
(ht)	9	11	0x09	)	41	51	0x29	I	73	111	0x49	i	105	151	0x69
(nl)	10	12	0x0a	*	42	52	0x2a	J	74	112	0x4a	j	106	152	0x6a
(vt)	11	13	0x0b	+	43	53	0x2b	K	75	113	0x4b	k	107	153	0x6b
(np)	12	14	0x0c	,	44	54	0x2c	L	76	114	0x4c	l	108	154	0x6c
(cr)	13	15	0x0d	-	45	55	0x2d	M	77	115	0x4d	m	109	155	0x6d
(so)	14	16	0x0e	.	46	56	0x2e	N	78	116	0x4e	n	110	156	0x6e
(si)	15	17	0x0f	/	47	57	0x2f	O	79	117	0x4f	o	111	157	0x6f
(dle)	16	20	0x10	0	48	60	0x30	P	80	120	0x50	p	112	160	0x70
(dc1)	17	21	0x11	1	49	61	0x31	Q	81	121	0x51	q	113	161	0x71
(dc2)	18	22	0x12	2	50	62	0x32	R	82	122	0x52	r	114	162	0x72
(dc3)	19	23	0x13	3	51	63	0x33	S	83	123	0x53	s	115	163	0x73
(dc4)	20	24	0x14	4	52	64	0x34	T	84	124	0x54	t	116	164	0x74
(nak)	21	25	0x15	5	53	65	0x35	U	85	125	0x55	u	117	165	0x75
(syn)	22	26	0x16	6	54	66	0x36	V	86	126	0x56	v	118	166	0x76
(etb)	23	27	0x17	7	55	67	0x37	W	87	127	0x57	w	119	167	0x77
(can)	24	30	0x18	8	56	70	0x38	X	88	130	0x58	x	120	170	0x78
(em)	25	31	0x19	9	57	71	0x39	Y	89	131	0x59	y	121	171	0x79
(sub)	26	32	0x1a	:	58	72	0x3a	Z	90	132	0x5a	z	122	172	0x7a
(esc)	27	33	0x1b	;	59	73	0x3b	[	91	133	0x5b	{	123	173	0x7b
(fs)	28	34	0x1c	<	60	74	0x3c	\	92	134	0x5c	|	124	174	0x7c
(gs)	29	35	0x1d	=	61	75	0x3d	]	93	135	0x5d	}	125	175	0x7d
(rs)	30	36	0x1e	>	62	76	0x3e	^	94	136	0x5e	~	126	176	0x7e
(us)	31	37	0x1f	?	63	77	0x3f	_	95	137	0x5f	(del)	127	177	0x7f

Operatoren

Precedence	Operator	Description	Example	Overloadable	Associativity
1	`::`	Scope resolution operator	Class::age = 2;	no	none
2	`()`	Function call	isdigit('1')	yes	left to right
	`()`	Member initalization	c_tor(int x, int y) : _x(x), _y(y*10){};	yes	
	`[]`	Array access	array[4] = 2;	yes	
	`->`	Member access from a pointer	ptr->age = 34;	yes	
	`.`	Member access from an object	obj.age = 34;	no	
	`++`	Post-increment	for(int i = 0; i < 10; i++) cout << i;	yes	
	`--`	Post-decrement	for(int i = 10; i > 0; i--) cout << i;	yes	
	`const_cast`	Special cast	const_cast<type_to>(type_from);	no	
	`dynamic_cast`	Special cast	dynamic_cast<type_to>(type_from);	no	
	`static_cast`	Special cast	static_cast<type_to>(type_from);	no	
	`reinterpret_cast`	Special cast	reinterpret_cast<type_to>(type_from);	no	
	`typeid`	Runtime type information	cout « typeid(type).name();	no	
3	`!`	Logical negation	if(!done) …	yes	right to left
	`not`	Alternate spelling for !			
	`~`	Bitwise complement	flags = ~flags;	yes	
	`compl`	Alternate spelling for ~			
	`++`	Pre-increment	for(i = 0; i < 10; ++i) cout << i;	yes	
	`--`	Pre-decrement	for(i = 10; i > 0; --i) cout << i;	yes	
	`-`	Unary minus	int i = -1;	yes	
	`+`	Unary plus	int i = +1;	yes	
	`*`	Dereference	int data = *intPtr;	yes	
	`&`	Address of	int *intPtr = &data;	yes	
	`new`	Dynamic memory allocation	long *pVar = new long;	yes	
	`new []`	Dynamic memory allocation of array	MyClass *ptr = new MyClass(args);	yes	
	`delete`	Deallocating the memory	delete pVar;	yes	
	`delete []`	Deallocating the memory of array	delete [] array;	yes	
	`(type)`	Cast to a given type	int i = (int) floatNum;	yes	
	`sizeof`	Return size of an object or type	int size = sizeof(float);	no	
4	`->*`	Member pointer selector	ptr->*var = 24;	yes	left to right
	`.*`	Member object selector	obj.*var = 24;	no	
5	`*`	Multiplication	int i = 2 * 4;	yes	left to right
	`/`	Division	float f = 10.0 / 3.0;	yes	
	`%`	Modulus	int rem = 4 % 3;	yes	
6	`+`	Addition	int i = 2 + 3;	yes	left to right
	`-`	Subtraction	int i = 5 - 1;	yes	

Operatoren

Precedence	Operator	Description	Example	Overloadable	Associativity
7	<<	Bitwise shift left	int flags = 33 << 1;	yes	left to right
	>>	Bitwise shift right	int flags = 33 >> 1;	yes	
8	<	Comparison less-than	if(i < 42) ...	yes	left to right
	<=	Comparison less-than-or-equal-to	if(i <= 42) ...	yes	
	>	Comparison greater-than	if(i > 42) ...	yes	
	>=	Comparison greater-than-or-equal-to	if(i >= 42) ...	yes	
9	==	Comparison equal-to	if(i == 42) ...	yes	left to right
	eq	Alternate spelling for ==			
	!=	Comparison not-equal-to	if(i != 42) ...	yes	
	not_eq	Alternate spelling for !=			
10	&	Bitwise AND	flags = flags & 42;	yes	left to right
	bitand	Alternate spelling for &			
11	^	Bitwise exclusive OR (XOR)	flags = flags ^ 42;	yes	left to right
	xor	Alternate spelling for ^			
12	|	Bitwise inclusive (normal) OR	flags = flags | 42;	yes	left to right
	bitor	Alternate spelling for |			
13	&&	Logical AND	if(conditionA && conditionB) ...	yes	left to right
	and	Alternate spelling for &&			
14	||	Logical OR	if(conditionA || conditionB) ...	yes	left to right
	or	Alternate spelling for ||			
15	? :	Ternary conditional (if-then-else)	int i = (a > b) ? a : b;	no	right to left
16	=	Assignment operator	int a = b;	yes	right to left
	+=	Increment and assign	a += 3;	yes	
	-=	Decrement and assign	b -= 4;	yes	
	*=	Multiply and assign	a *= 5;	yes	
	/=	Divide and assign	a /= 2;	yes	
	%=	Modulo and assign	a %= 3;	yes	
	&=	Bitwise AND and assign	flags &= new_flags;	yes	
	and_eq	Alternate spelling for &=			
	^=	Bitwise exclusive or (XOR) and assign	flags ^= new_flags;	yes	
	xor_eq	Alternate spelling for ^=			
	|=	Bitwise normal OR and assign	flags |= new_flags;	yes	
	or_eq	Alternate spelling for |=			
	<<=	Bitwise shift left and assign	flags <<= 2;	yes	
	>>=	Bitwise shift right and assign	flags >>= 2;	yes	
17	throw	throw exception	throw EClass("Message");	no	
18	,	Sequential evaluation operator	for(i = 0, j = 0; i < 10; i++, j++) ...	yes	left to right

Quiz

- Deklariere die Variablen, sodass kein impliziter Cast vorgenommen wird:

```
a) a = 2; b = a < 7; c = '6';
b) a = b || char( c – 7) == 'b';

a) int a; bool b; char c;


b) bool a; bool b; int c;
```

Übung 2

Aufgabe 1: Gültige Variablennamen

```
int _123;                Ja
float Variable A;        Nein, wegen Leerzeichen
double 123_;             Nein, weil Zahl am Anfang
char WasIstDas?;         Nein, wegen Sonderzeichen
```

Aufgabe 2: Algebraische Operationen

```
1.5f * 2        = 1.5f * 2.0f = 3.0f      - float
5 / 3                         = 1         - int
5 / 2.5         = 5.0 / 2.5   = 2.0       - double
5.0 / 3         = 5.0 / 3.0   = 1.66      - double
2.0f / 3.0      = 2.0 / 3.0   = 0.66      - double
1 % 2                         = 1         - int

2 / 4 * 4       = 0 * 4       = 0         - int
2.0 / 4 * 4     = 0.5 * 4     = 2.0       - double
2 * 4 / 4       = 8 / 4       = 2         - int
4 + 2 % 4       = 4 + 2       = 6         - int

6.0f * (2 / 4)  = 6.0f * 0.0f = 0.0f      - float
```

Übung 2

Aufgabe 3: Typkonvertierung (Casting)

```
2 / float(3)      = 2.0f / 3.0f = 0.66f      - float
double(2 / 3)     = double(0)   = 0.0        - double
```

Aufgabe 4: Programmanalyse

Quelltext	Variableninhalt	Ausgabe
`...`		
`int i1, i2 = 3, i3(4.5);`	`i1 undefiniert, i2 = 3, i3 = 4;`	
`double d1 = 4.5f;`	`d1 = 4.5`	
`i1 = i2 * d1 / i3;`	`i1 = 3, i2 = 3, i3 = 4, d1 = 4.5`	
`cout << 4 + 5 << endl;`		9
`cout << ‘4’ + ‘5’ << endl;`		105
`cout << “4” << “5” << endl;`		45
`cout << (‘4’ < ‘5’) << endl;`		1
`cout << ‘4’ << ‘5’ << endl;`		45
`cout << ‘4’ / ‘5’ << endl;`		0
`//cout << ‘4’ < ‘5’ << endl;`	`Kompilererror (niedrigere Präzedenz)`	
`...`		

Übung 2

Aufgabe 5: Ein Programm

Lese vier Zahlen (double) ein und berechne den Mittelwert:

```
#include <iostream>                                    // für cout und cin
using namespace std;
int main() {
    double input1, input2, input3, input4;             // undef. variablen
    double mittelwert;                                 // undef. variablen
    cout << "Mittelwert von vier Zahlen:" << endl;
    cout << "Zahl 1: “;                                // Zahl 1 einlesen
    cin >> input1;
    cout << "Zahl 2: “;                                // Zahl 2 einlesen
    cin >> input2;
    cout << "Zahl 3: “;                                // Zahl 3 einlesen
    cin >> input3;
    cout << "Zahl 4: “;                                // Zahl 4 einlesen
    cin >> input4;
    mittelwert = (input1 + input2 + input3 + input4)/4;  // Mittelwert berechnen
    cout << "Mittelwert: " << mittelwert << endl;      // Mittelwert ausgeben
    return 0;                                          // Rückgabe
}
```

Übung 2

z:\Classes\Informatik for D-MAVT 2010\Session 1\Example\HelloWorld\Debug\HelloWorld.exe
Mittelwert von vier Zahlen:
Zahl 1: 3.0
Zahl 2: 2.5
Zahl 3: 1.33
Zahl 4: 0.75
Mittelwert: 1.895