Grundlagen der Programmierung in C

Basics

Wintersemester 2005/2006
G. Zachmann
Clausthal University, Germany
zach@in.tu-clausthal.de

Was ist ein Programm?

- Abstrakt: Zeichenfolge entsprechend einer formalen Grammatik
  - Formale Grammatik besteht aus Regeln folgender Art:

```
statement-list :
        ;                                   [leeres Statement]
        statement ; statement-list
```

  - Terminale sind Zeichen eines Alphabets
  - Spätere Vorlesung zu formalen Sprachen
- Konkret:
  - Wörter (*Identifier*) und Zahlen (*Literals*)
  - Operatoren (`+`, `%`, `&`, …)
  - Sonstige Sonderzeichen (fast jedes hat eine eigene Bedeutung)

Whitespace

- Space, Tab, oder Newline
- Wird vom Compiler ignoriert
  - Siehe Obfuscated Code

Kommentare

- Werden vom Compiler ignoriert
- 2 Arten:
  - C++-style:

```
const int Ntries;  // this is an inline comment
// the rest of the line is treated like a comment
```

  - C-style:

```
const int Ntries;
/*  this is a multiline comment:
    Everything is treated like a comment.
    Comments can't be nested. The comment is
    closed with a */
```


Identifier

- Anderes Wort für Name (Variablename, Funktionsname, Keywordname)
- Zeichenkette
  - Zugelassen: alphanumerische Zeichen und Underscore (_)
  - Erstes Zeichen darf nicht Ziffer sein
    - `blub` und `_bla` sind ok
    - `2pi` nicht ok
- Kein Limit auf Länge
  - Achtung: manche unterscheiden nur die ersten 32 Zeichen
- Case-sensitiv

Keywords

- Wörter mit spezieller Bedeutung
- Sind reserviert, kann man nicht als Namen verwenden

keyword			
asm	else	operator	throw
auto	enum	private	true
bool	explicit	protected	try
break	extern	public	typedef
case	false	register	typeid
catch	float	reinterpret_cast	typename
char	for	return	union
class	friend	short	unsigned
const	goto	signed	using
const_cast	if	sizeof	virtual
continue	inline	static	void
default	int	static_cast	volatile
delete	long	struct	wchar_t
do	mutable	switch	while
double	namespace	template	
dynamic_cast	new	this	

Eingebaute (built-in) Typen

- Jede Variable muß einen bestimmten Typ haben
- Muß zur Compile-Zeit feststehen und kann nicht gewechselt werden!

```
int                 single precision integer
long int            double precision integer
float               single precision real
double              double precision real
long double         extended precision real
unsigned int        unsigned integer
char                single character (a letter)
bool                logical variables
```

- Exkurs:
  - Smalltalk kennt keine Typen, alles wird zur Laufzeit gecheckt
  - Viele Skript-Sprachen haben nur einen Typ (String)

Literals

Numerische Literals

`123 0123 0x123`	int's, decimal, octal, hex
`123l 123u`	long, unsigned
`'A' '1' '\t'`	characters, tab
`3.14f 3.14 3.14L`	float, double, long double
`3e-2 0.03 0.3e-1`	scientific notation
`true false`	boolean

Konstante Strings

`""`	null string ('\0')
`"name"`	'n' 'a' 'm' 'e' '\0'
`"a \"string\""`	prints: a "string"
`"a string "`	string concatenation for
`"spanning two lines"`	better readability

Spezielle Zeichen (escape sequence)

`'\a'`	alert (bell)
`'\\'`	backslash
`'\b'`	backspace
`'\r'`	carriage return
`'\"'`	double quote
`'\f'`	form feed
`'\t'`	tab
`'\n'`	newline
`'\0'`	null character
`'\''`	single quote
`'\v'`	vertical tab
`'\101'`	101 octal, 'A'
`'\x041'`	hex, 'A'

Strings

- Kein eingebauter Typ an sich
- String-Konstanten: wie eingebaute Typen schreiben
- Repräsentierung im Speicher:

Deklaration

- Assoziiert Bedeutung (d.h. vor allem:Typ) mit Identifier
- Exkurs:
  - In allen compilierten Sprachen muß Identifier erst deklariert werden, bevor er benutzt werden kann
    - →Strong type checking
  - In den meisten interpretierten Sprachen (insbesondere Skript-Sprachen) nicht notwendig/möglich
    - →Weak type checking, "run-time type" checking

Variablen

- Funktion wie in Mathematik, speichert Wert
- Variable = Paar (Identifier, Typ)
- Deklaration:

  T V;

  mit: T = bekannter Typ, V = Variablenname
- Beispiele:

```
int i;                  // the variable 'i'
float f1, f2;
char c1,                // comment
     c2;                // comment
```


Initialisierung

- Deklaration kann gleichzeitig zur Belegung mit Wert fungieren:

```
float pi = 3.1415926;       // definition
int seconds_per_day = 60*60*24;
```

- Sollte man (fast) immer machen!

Konstanten

- Deklaration:
  `const` *type var-name = const-expression;*
- Beispiele:

```
const float Pi = 3.1415926;
const unsigned int answer = 42;
```


Wertebereiche

- Jeder Typ hat bestimmten Wertebereich
- Plattform-abhängig!
- Symbolische Konstanten für diese Bereiche:
  - Bekommt man durch: `#include <limits>`
  - Beispiele:

```
#include <stdlib.h>
#include <stdio.h>
#include <limits>
using namespace std;
...
int a = numeric_limits<int>::max();
char c = numeric_limits<char>::min();
float eps = numeric_limits<float>::epsilon();
```


Style Guidelines für Variablen

- Style Guidelines sind sehr wichtig:
  - Lesbarkeit
  - Wartbarkeit des Programms
- "Kleine" Variablen:
  - Laufvariablen, Variablen mit sehr kurzer Lebensdauer
  - 1-3 Buchstaben
  - `i,j,k,…` für int's
  - `a,b,c,x,y,z …` für float's
- "Große" Variablen:
  - Längere Lebensdauer, größere Bedeutung
  - Labeling names! ("Sprechende Namen")
  - `mein_alter, meinAlter, determinante, …`

Ausdrücke

- Ausdruck ("expression") = mathematischer Term
- Beispiel: `sin(x)*sin(2*x)`
- Compiler generiert Anweisungen zur Auswertung
- FORTRAN (Formula Translator) war Ende der 50er Jahre die erste Programmiersprache mit dieser Fähigkeit
- Ausdruck setzt sich zusammen aus:
  - Literalen (Konstanten), Variablen, Funktionen
  - Operatoren
  - Klammern
  - Übliche Regeln

Operatoren

Arithm. Operator	Meaning
`-i  +w`	unary + and -
`a*b  a/b  i%2`	mult., div., modulo
`a+b  a-b`	binary + and -
`i=3;  a=3.0`	int/float assignment

Relational Operators	
`<`	less than
`>`	greater than
`<=`	less or equal
`>=`	greater or equal
`==`	equals
`!=`	not equal

Self-increment and decrement	Equivalent to
`k = ++j;`	`j=j+1; k=j;`
`k = j++;`	`k=j;  j=j+1;`
`k = --j;`	`j=j-1; k=j;`
`k = j--;`	`k=j;  j=j-1;`

Boolean Operators	
`!`	unary not
`&&`	logical and
`||`	logical or

Bit-wise Operators	
`~ i`	bitwise Complement
`i & j`	bitwise AND
`i | j`	bitwise OR
`i ^ j`	bitwise XOR
`i << n`	left shift (n positions)
`i >> n`	right shift (n positions)

Assignment operator	equivalent
`x □= y`	`x = x □ (y)`
Bsp.:	
`x -= y`	`x = x - y`
`x /= y`	`x = x / y`
`x &= y`	`x = x & y`

Ganzzahlarithmetik

- Wertebereich
  - Siehe `numeric_limits`
  - Typ. -2^{31} … $2^{31}-1$ (32-bit int's)
- Wrap-around: max.int +1 == min.int ! Kein Überlauf!
- Division: rundet zur 0!
  - Beispiel: 3/2 == 1, und -3/2 == -1
- Division und Modulo: `(x/y)*y + x%y == x`
- Beispiele:

```
a    = 1234
b    = 99
sum  = a + b #1333
prod = a * b #122166
quot = a / b #12
rem  = a % b #46
```

Beispiel: Berechnung von Schaltjahren

```
int y = 2005;
# Durch 4 teilbar, aber nicht durch 100
bool isLeapYear = (y % 4 == 0) && (y % 100 != 0);
# Oder durch 400 teilbar
bool isLeapYear = isLeapYear || (y % 400 == 0);
```