Leibniz Universität Hannover

Programmieren 1

Remaining C Keywords, Extended Backus-Naur Form

(nicht klausurrelevant, da wegen der längeren Winterpause nicht als Vorlesung gehalten)

Human-Computer Interaction Group

Prof. Dr. Michael Rohs
michael.rohs@hci.uni-hannover.de

Lectures

#	Date	Topic	HÜ→	HÜ←
1	14.10.	Organization, computers, programming, algorithms, PostFix introduction (execution model, IDE, basic operators, booleans, naming)	1	20.10. 23:59
2	21.10.	PostFix (primitive types, functions, parameters, local variables, tests), recipe for atomic data	2	27.10. 23:59
3	28.10.	PostFix (operators, array operations, string operations), recipes for enumerations and intervals	3	3.11. 23:59
4	4.11.	Recipes for compound and variant data, iteration and recursion, PostFix (loops, association arrays, data definitions)	4	10.11. 23:59
5	11.11.	C introduction (if, variables, functions, loops), Programming I C library	5	17.11. 23:59
6	18.11.	Data types, infix expressions, C language (enum, switch)	6	24.11. 23:59
7	25.11.	Compound and variant data, C language (formatted output, struct, union)	7	1.12. 23:59
8	2.12.	C language (arrays, pointers) arrays: fixed-size collections, linear and binary search	8	8.12. 23:59
9	9.12.	Dynamic memory (malloc, free), recursion (recursive data, recursive algorithms)	9	15.12. 23:59
10	16.12.	Linked lists, binary trees, search trees	10	22.12. 23:59
11	13.1.	C language (program structure, scope, lifetime, linkage), function pointers, pointer lists	11	12.1. 23:59
12	20.1.	List and tree operations (filter, map, reduce), objects, object lists	12	19.1. 23:59
13	27.1.	Dynamic data structures (stacks, queues, maps, sets), iterators, documentation tools	(13)	

Preview

- Remaining C keywords
- Processing input using Extended Backus-Naur Form (EBNF)

REMAINING C KEYWORDS

C Keywords (C89)

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Goto Statement

- Jump to another point in the same function
  - Can lead to “spaghetti code”. Should be avoided.
- Example: Find letter ‘x’ in a String

```
String s = "...";
int n = strlen(s);
for (int i = 0; i < n; i++) {
  if (s[i] == 'x') {
    goto found;
  }
}
printsln("'x' not found");
return false;
found:
printsln("'x' found");
return true;
```

label found, target of goto, must be in same function

Goto Statement and Early Return

- Jump to another point in the same function
  - Can lead to “spaghetti code”. Should be avoided.
- Example: Find letter ‘x’ in a String

```
String s = "...";
int n = strlen(s);
for (int i = 0; i < n; i++) {
  if (s[i] == 'x') {
    goto found;
  }
}
printsln("'x' not found");
return false;
found:
printsln("'x' found");
return true;
```

- Early return

```
String s = "...";
int n = strlen(s);
for (int i = 0; i < n; i++) {
  if (s[i] == 'x') {
    printsln("'x' found");
    return true;
  }
}
printsln("'x' not found");
return false;
```

Early Return vs. Structured Programming

- Early return

```
String s = "...";
int n = strlen(s);
for (int i = 0; i < n; i++) {
  if (s[i] == 'x') {
    printsln("'x' found");
    return true;
  }
}
printsln("'x' not found");
return false;
```

- Structured programming

```
String s = "...";
int n = strlen(s);
bool found = false;
for (int i = 0; !found && i < n; i++) {
  if (s[i] == 'x') {
    found = true;
  }
}
if (found) printsln("'x' found");
else printsln("'x' not found");
return found;
```

Go to Check Postcondition

```
int find_char(String s, char c) {
  int n = strlen(s);
  int i = 0;
  for (; i < n; i++) {
    if (s[i] == c) {
      // return i; <-- do not return without checking postcondition
      goto check_postcondition;
    }
  }
  i = -1; // not found
  check_postcondition:
  ensure("correct", (i >= 0 && forall(j, i, s[j] != c) && s[i] == c)
                 || (i == -1 && forall(j, n, s[j] != c)));
  return i;
}
```

Automatic Variables

- Explicit designation of an “automatic” variable
- keyword auto
  - `auto int i = 4;`
- A variable that is automatically created/destroyed as the program flow enters/exits its scope
  - Non-static local variables are automatic variables by default
- Rarely used

Register Variables

- register modifier in variable definitions
  - `register int i;`

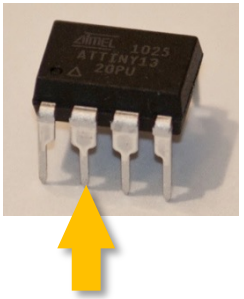
- Like automatic variables, but asks the compiler to put variable into processor register, if possible
  - Accessing processor register is very efficient
  - Limited number of registers

- Rarely needed: Optimizing compilers can figure out automatically what variables to put into registers

Volatile Variables

- Tells compiler that value of variable can change "on its own"
  - Prevents optimizations based on assumption
    that variables don't change without assignment
- Used if certain addresses represent I/O ports (not RAM)
  - Often the case with microcontrollers
- Example:

```
static volatile char *p = 0x8000; // 0x8000 is an I/O port (not RAM)
void main(void) {
    *p = 0;
    while (*p != 1); // would be an infinite loop, if p pointed to memory
    // exited loop when *p was 1...
}
```



C Keywords (C89)

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

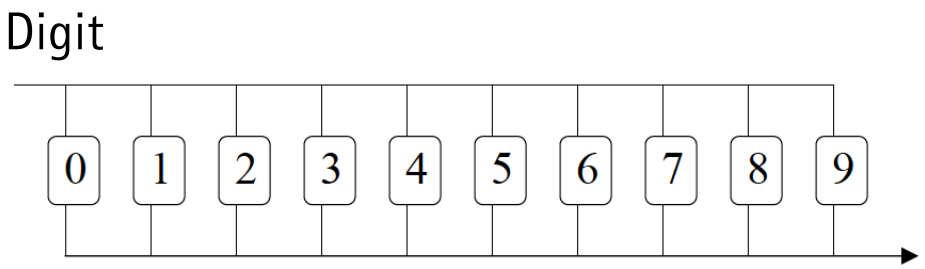
Jippee!!!

Leibniz Universität Hannover

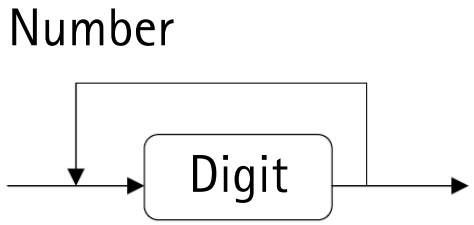
PROCESSING INPUT USING EBNF AND FINITE STATE MACHINES

Grammars for Describing Syntax

- How to precisely describe the syntax of programming languages?
- Grammar
  - Set of syntax rules, determines the set of valid sentences
- Example: Grammar of natural numbers (including 0)
  - Digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
  - Number = Digit {Digit}.
- Graphical notation:



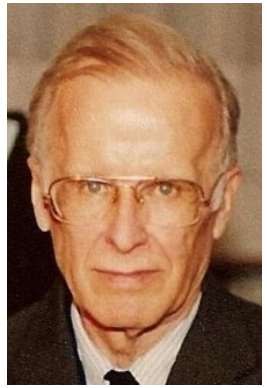
"one digit"



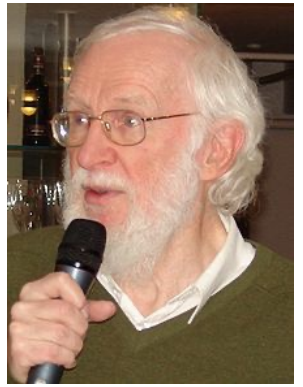
"one or more digits"

Grammar in Extended Backus-Naur Form (EBNF)

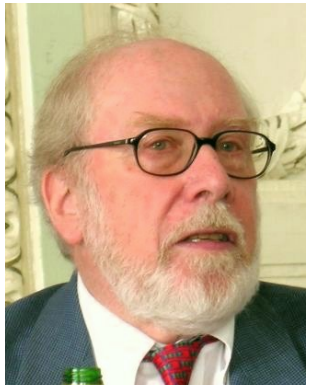
Notation	Meaning	Example	Instances
=	definition	A = x y z .	
.	termination	A = x .	
|	alternation	x | y	x, y
( )	grouping	(x | y) z	xz, yz
[ ]	option	[x] y	xy, y
{ }	repetition	{x} y	y, xy, xxy, xxxy, ...



John Backus, 1924-2007,
Turing Award 1977



Peter Naur, 1928-2016,
Turing Award 2005



Niklaus Wirth, *1934,
Turing Award 1984

Examples

- Grammar of floating point numbers
  - FloatingPointNumber = ["+"|"-"] Number "." Number ["E" ["+"|"-"] Number].
- Grammar of the if-Statement
  - IfStatement = "if" "(" Expression ")" Statement ["else" Statement].

What is a Valid Integer Number?

- "an optional sign followed by one or more digits"
- in EBNF:
  Integer = ["-"] digit {digit}.
  digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".

- Recognizing an Integer: ["-"] digit {digit}

```
bool isint(char* s) {
  if (*s == '-') s++;
  if (isdigit(*s)) s++; else return false;
  while (isdigit(*s)) s++;
  if (*s != '\0') return false;
  return true;
}
```

```
Integer =
 ["-"]
 digit
 {digit}.
```

What is a Valid Integer Number?

- "an optional sign followed by one or more digits"
- in EBNF:
  Integer = ["-"] digit {digit}.
  digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".

- Recognizing an Integer: ["-"] digit {digit}

```
bool isint(void) {
  if (c == '-') next();
  if (isdigit(c)) next(); else return false;
  while (isdigit(c)) next();
  return true;
}
```

```
static char* input;
static char c;

void next(void) {
  c = *input;
  input++;
}
```

What is a Valid Floating Point Number?

- "an optional sign followed by one or more digits followed by a decimal point followed by one or more digits"
- in EBNF:
  Float = ["-"] digit {digit} "." digit {digit}.
- or:
  Float = Integer "." digit {digit}.
  Integer = ["-"] digit {digit}.

What is a Valid Floating Point Number?

```
// Checks if current input is an integer.
bool isint(void) {                                  // Integer =
  if (c == '-') next();                             // ["-"]
  if (isdigit(c)) next(); else return false;        // digit
  while (isdigit(c)) next();                        // {digit}.
  return true;
}

// Checks if current input is a float.
bool isfloat(void) {                                // Float =
  if (!isint()) return false;                       // Integer
  if (c == '.') next(); else return false;          // "."
  if (isdigit(c)) next(); else return false;        // digit
  while (isdigit(c)) next();                        // {digit}.
  return true;
}
```

Summary

- Remaining C keywords
- Processing input using Extended Backus-Naur Form (EBNF)

Leisure Reading

- We avoided the goto statement as its use can result in unstructured and difficult to read code
- If you are interested in the (earlier) scientific debate about goto, see:
  - Edsger Dijkstra: Go To Statement Considered Harmful. Communications of the ACM, March 1968. http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD215.PDF
  - Frank Rubin: "GOTO Considered Harmful" Considered Harmful. Communications of the ACM, March 1987. http://web.archive.org/web/20090320002214/http://www.ecn.purdue.edu/ParaMount/papers/rubin87goto.pdf
  - Donald Moore, Chuck Musciano, Michael J. Liebhaber, Steven F. Lott and Lee Starr. "'GOTO Considered Harmful' Considered Harmful" Considered Harmful? Communications of the ACM, May 1987.