01/27/2010

Recitation #2

Administrative

- Any questions on lecture content?
- Wiki page on installing Eclipse on Windows/Linux/MacOS
- Lab notes will be uploaded in the same location: http://mclserver.eecs.ucf.edu/trac/rpillat/wiki/COP3402Spring2011
- Assignment #2 posted by Dr. Hughes, due February 10th, 11:59 p.m.

Assignment 1

- Some **Pascal peculiarities**:
  - Interesting comparison of Pascal and C: http://en.wikipedia.org/wiki/Comparison_of_Pascal_and_C
  - Arrays
    - Indices traditionally starting at 1, unless you specify a custom range like [0..9] instead of [1..10], Pascal can use any ordinal type as array index → in C/C++ always starting at 0
    - Packed array are string array of fixed length
  - `i = ord(c)` of character c gives decimal ASCII code (see ASCII table below), `c = chr(i)` does the opposite
  - routines that don't return a value are called `procedure k(q: integer);`
  - routines that do return a value are functions: `function f(x, y: integer): integer;`

	Pascal	C Equivalent	C++ Equivalent
Variable Declaration	`var a : integer`	`int a`	`int a`
Fixed-length String	`packed array[] of char`	`char[]`	`std::string`
Type definition	`type`	`typedef`	`typedef`
Comments	`{ … } , (* … *)`	`/* … */`	`//, /* … */`
Blocks	`begin … end`	`{ … }`	`{ … }`
Enumeration type	`type color = (r, g, b);`	`enum color {r, g, b};`	`enum color {r, g, b};`
Arrays	`var a : array[0..9] of integer`	`int a[10]`	`int a[10]`
Equality Test	`=`	`==`	`==`
Assignment Operator	`:=`	`=`	`=`
Printout	`writeln`	`printf`	`std::cout`

- Biggest problem area in reading the code is how they handle numbers:

```
if ch in ['0'..'9'] then
begin (* number *)
   k := 0; inum := 0; sy := intcon;  (* assume it's an integer number *)
   repeat inum := inum*10 + ord(ch) - ord('0');
      k := k+1; nextch
   until not (ch in ['0'..'9']);
   if (k > kmax) or (inum > nmax) then
      begin error(21); inum := 0; k := 0 end;
   if ch = '.' then (* real number *)
   begin nextch;
      if ch = '.' then ch := ':' else
         begin sy := realcon; rnum := inum; e := 0;
            while ch in ['0'..'9'] do begin
               e := e-1;
               rnum := 10.0*rnum + (ord(ch)-ord('0')); nextch
            end;
            if ch = 'e' then readscale;
            if e <> 0 then adjustscale
         end
   end
   else if ch = 'e' then begin (* real number *)
      sy := realcon; rnum := inum; e := 0;
      readscale; if e <> 0 then adjustscale;
   end
end else
```

Reading an integer number

Found a "." Could be subrange or real.

Just a subrange

Reading a floating-point number (ignoring the dot for now)

Scientific Notation

If there were any numbers after "." adjust number.

The variable "e" keeps track of how many positions the decimal point has to move. "e" can change in this code or in function readscale.

Context-Free Grammar

- Dr. Hughes gave you the Pascal-S grammar in EBNF form
- Necessary for **syntactic analysis**
- Syntax of programming language constructs can be specified by context-free grammars
- Grammars are described by EBNF (Extended Backus Naur Form) notation → see Recitation #1
- Parser can be generated automatically from EBNF
- Go through some of the grammar and explain

Assignment 2

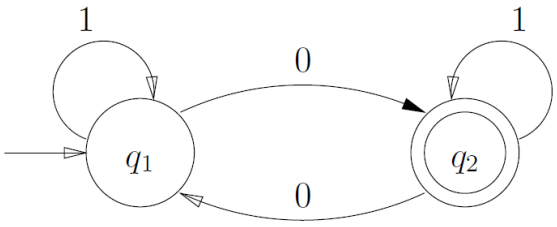
- From Dr. Hughes website:
  "*Starting with pascal.lex, change the rules in my regular expressions to accommodate exponents in numbers. Note: An integer followed by an exponent is a real.*
  Change it to include // and %, as you did in Assign#1. I already did the comments, except for directives. While you must just submit the modified grammar, I strongly recommend you download FLEX and try it. That's what we will do for checking your changes."
- **Flex**
  - http://flex.sourceforge.net/ for source, http://gnuwin32.sourceforge.net/packages/flex.htm for Windows executable)
  - Concise language to automatically generate source code for lexical analyzers / parsers
  - The description is in the form of pairs of regular expressions and C code, called rules. Flex generates a C source file
  - Flex++ is a variant of Flex that generates C++ code
  - Format of .lex file:

```
definitions
```

```
%%
rules
%%
user code
```

- Is usually used as input for syntactic analyzer, like YACC or BISON
- Demo code generation on computer

- **Regular Expressions**
  - Expressions that describe a set of strings
  - Equivalent to all strings a Deterministic Finite Automata (DFA) can recognize
  - Very easy to generate recognition code as DFA
  - See reference sheet on back of assignment sheet
  - For example:
    - `(a|b)*` denotes the set of all strings with no symbols other than a and b, including the empty string: {ε, a, b, aa, ab, ba, bb, aaa, ...}
    - `ab*c?` denotes the set of strings starting with a, then zero or more bs and finally optionally a c: {a, ac, ab, abc, abb, abbc, ...}
    - `^[hc]at` matches "hat" and "cat", but only at the beginning of the string or line
    - `(^[1-9]{1}$|^[1-4]{1}[0-9]{1}$|^50$)` is any number from 1 to 50 inclusive



Is equivalent to: `1*0(1|01*0)*`

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