Lexical Elements

chapter 2

Comments

```
Arbitrary strings placed between the delimiters /* and */.

/*

* A comment can be written in this fashion

* to set it off from the surrounding code.

*/
```

The compiler changes each comment into a single blank character.

```
In this course we do not use the c++ style!!!

// This is a comment in c++
```

Keywords

Reserved words with strict meanings.

C does a lot with relatively few keywords.

Some implementations on some systems may have additional keywords.

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

Identifiers

A token composed of a sequence of letters, digits and the special character _ (underscore)

First character cannot be a digit!!

5a_cse is not legal

- Case sensitive: CSE_5a is different from cse_5a
- Some identifiers are already taken: keywords.
- Choose name that are meaningful (cse_5a is not a good example!!)
- In ANSI C the first 31 characters of an identifier are identified.

Constants

```
Examples for constant decimal integers

0

0123
```

But

```
123456789000 /*too large for the machine */
0123 /*an octal integer?*/
-49 /*constant expression*/
123.0 /* a floating constant*/
```

String Constants

A sequence of characters in double quotes marks.

String constants are treated as arrays of characters, so that "a" is different than 'a'

```
"a string of text"
""    /* the null string */
"a = b + c"    /* nothing executed here */
"a string with double quotes \" within"
"a string backslash \\ is in this string"
```

Two string constants separated by white space are concatenated "abc" "def" is equivalent to "abcdef"

Operators and Punctuators

In C many characters have particular meaning. For example

```
+ - * / %
a+b /* the expression a plus b */
a_b /*a 3-character identifier*/
Some symbols have meanings that depend on context, for example
fprintf("%d",a); versus a = b % 7;
or
a+b ++a a+=b
```

Punctuations include parentheses, braces, commas, and semicolons

Precedence of Operators

Basically, mathematical expressions are evaluated as in mathematical convention.

$$1 + 2 - 3 + 4 - 5$$
 /* gives -1 */

Things may be a little tricky when dealing with mixture of postfix, prefix and unary operators for example

```
-a * b - c is equivalent to ((-a) * b) - c
```

Increment & Decrement Operators

The increment ++ and decrement -- operators are unary. They can be used as prefix and postfix operators.

Both can be applied to variables but not to constants.

What is the difference between ++i and i++ ?

```
int a = 0, b = 0, c = 0;
a = ++c;
b = c++;
printf( "%d %d %d\n", a, b, ++c );
```

What will be printed?

Assignment Operators

C treats = as a operator with precedence lower than all we have discussed.

```
b = 2;
c = 3;
a = b + c;
Is equivalent to
a = (b = 2) + (c = 3);
More useful statements
a = b = c = 0;
```

More assignment Operators

A list of all assignment operators

Operator Precedence and Associativity

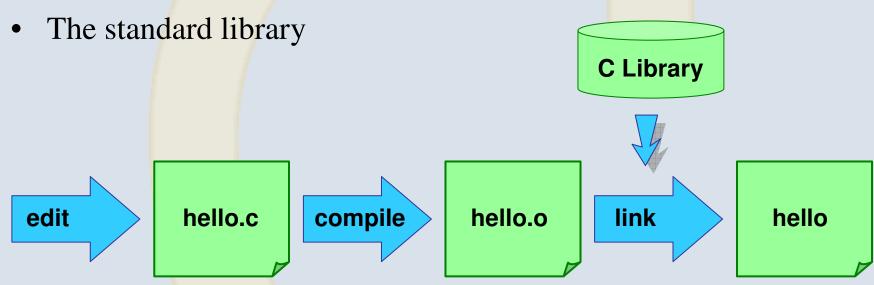
Operator	Associativity
() $++ (postfix) (postfix)$	left to right
+ (unary) - (unary) ++ (prefix) (pr	right to left
* / %	left to right
+ -	left to right
= += -= *= /= e	tc. right to left

An example - powers of 2

```
/* Some powers of 2 are printed. */
#include <stdio.h>
int main (void)
   int i = 0, power = 1;
  while (++i <= 10)
      printf("%6d", power *= 2);
  printf("\n");
   return 0;
```

The C System

• The preprocessor



Using a function from standard library

```
#include <stdio.h>
#include <stdlib.h>
int main (void)
   int i, n;
   printf("\n%s",
      "How many random numbers do you want to see? ");
   scanf("%d", &n);
   for (i = 0; i < n; ++i) {
      if (i % 10 == 0)
         putchar('\n');
      printf("%7d", rand());
   printf("\n\n");
   return 0;
```