Lexical Elements and Operators

Lexical Elements

Kinds of tokens in C :

- Keywords
- Identifiers
- Constants
- Operators
- Punctuators

Keywords

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


Identifiers

- Sequence of letters, digits, and the special character _.
- A letter or underscore must be the 1st character of an identifier.
  - For this class, don't use identifiers that begin with an underscore.
- C is case-sensitive: `Apple` and `apple` are two different identifiers.

Identifiers (cont.)

- Valid variable names:
  - `n`
  - `x`
  - `_id`
  - `num1`
  - `a_long_identifier`
- Invalid variable names:
  - `var.1`
  - `num!2`
  - `not#this`
  - `126East`
  - `+more`

Declarations

- All variables must be declared before use. A declaration specifies a type, and contains a list of one or more variables of that type.

```
int lower, upper, step;
int c, line;
```

- A variable may be initialized in its declaration.

```
int i = 0;
```

- Variables for which there is no explicit initialization have undefined (garbage) values.

Constants

C manipulates various kinds of values.

- integer constants: `0, 37, 2001`
- floating constants: `0.8, 199.33, 1.0`
- character constants: `'a', '5', '+'`
- string constants: `"a", "Monday"`

(more on this topic later.)

Arithmetic Operators

- The binary arithmetic operators are `+, -, *, /,` and the modulus operator `%`.
- `x % y` produces the remainder when `x` is divided by y.

```
11 % 5 = 1
20 % 3 = 2
```

- Arithmetic operators associate left to right.

Precedence and Associativity of Operators

Operator Precedence:

```
x = 1 + 2 * 3;  (What is the value of x?)
x = 1 + (2*3);          x is 7? or
x= (1+2) * 3;           x is 9?
```

Associativity: (*left to right*)

```
10 + 3 + 7
10 – 3 + 7
15 / 5 * 2
```


Assignment Operator

variable = expression

- The expression can simply be a constant or a variable:

```
int x, y;
x = 5;
y = x;
x = 6;
```

- The expression can be an arithmetic expression:

```
x = y + 1;
y = x * 2;
```


Assignment Compatibility

```
int x;
double y;
x = y;          truncates y!
y = x;          it is okay.
x = 5 + 3.2;    truncates the result!
y = 5 + 3.2;    it is okay.
x = 10/4;       x is 2
y = 10/4;       y is 2.0
y = 10/4.0;     y is 2.5
```


Increment and Decrement Operators

- The increment operator ++ adds 1 to its operand.

  ++i; *equiv.* i = i + 1;

  i++; *equiv.* i = i + 1;

- The decrement operator -- subtracts 1 from its operand.

  --i; *equiv.* i = i - 1;

  i--; *equiv.* i = i - 1;

Increment and Decrement Operators

Suppose n = 5.

```
n++;            /* sets n to 6 */
n = n + 1;

++n;            /* sets n to 6 */
n = n + 1;
```


Increment and Decrement Operators (cont.)

- When **++a** is used in an expression, the value of **a** is incremented **before** the expression is evaluated.
- When **a++** is used, the expression is evaluated with the **current value** of **a** and then **a** is incremented.
- Similarly, with **--a** and **a--**.

Increment and Decrement Operators (cont.)

Suppose n = 5.

```
x = n++;        /* sets x to 5 and n to 6 */
    1. x = n;
    2. n = n + 1;
x = ++n;        /* sets x and n to 6 */
    1. n = n + 1;
    2. x = n;
```



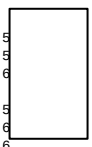
```
/* Preincrementing and postincrementing*/

#include <stdio.h>

int main(void)
{
   int c;

   c = 5;
   printf("%d\n",c);          5
   printf("%d\n", c++);       5
   printf("%d\n\n",c);        6

   c = 5;                     5
   printf("%d\n", c);         6
   printf("%d\n", ++c);       6
   printf("%d\n", c);
   return 0;
}
```



```
/*** increment and decrement expressions ***/
#include <stdio.h>

int main(void)
{
   int a =0 , b = 0, c = 0;

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

```
2 1 1
2 2 2
5 3 3
5 2 2
```


Arithmetic Assignment Operators

Assume:

```
int c = 3, d = 5, e = 4, f = 6, g = 12;
```

Operator	Expression	Explanation	Assigns
+=	c += 7	c = c + 7	10 to c
-=	d -= 4	d = d - 4	1 to d
*=	e *= 5	e = e * 5	20 to e
/=	f /= 3	f = f / 3	2 to f
%=	g %= 9	g = g % 9	3 to g