Computer Science 5C
Chapter 2--Lexical Elements, Operators, and the C System

Dr. Scott A. Brandt
Professor, Computer Science Department
Director, UCSC/LANL Institute for Scalable Scientific Data Management
The C language

- C really is a language
- It has a syntax -- rules that define the combinations of symbols that are allowed
- It has a grammar -- rules that define the allowed strings of words (i.e., programs)
- It has semantics -- rules for interpreting the meaning of programs or functions (i.e., what the program will do).
Tokens

- Tokens are the lowest-level meaningful elements of a program
  - The compiler parses sets of characters into tokens
- C tokens
  - keywords
  - identifiers
  - constants
  - string constants
  - operators
  - punctuators
- Many errors are the result of incorrect token creation
**Characters and Lexical Elements**

- **Lowercase letters:** a, b, c, …, z
- **Uppercase letters:** A, B, C, …, Z
- **Digits:** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- **Others:** +, -, *, /, =, (, ), {, }, [ , ] , < , > , ‘ , “ , ! , @ , # , $ , % , & , _ , | , ^ , ~ , \ , . , , , ; , : , ?
- **White space:** blank, newline, tab, etc.

- These characters are used to specify tokens
  - White space and punctuation separate other tokens
Sum

/* Read in two scores and print their sum. */
#include <stdio.h>

int main(void)
{
    int score_1, score_2, sum;

    printf("Input two scores as integers: ");
    scanf("%d%d", &score_1, &score_2);
    sum = score_1 + score_2;
    printf("%d + %d = %d\n", score_1, score_2, sum);
    return 0
}
Comments

- Arbitrary strings of characters between /* and */
  - Not a token --- turned into whitespace
  - Only for readability to humans
- /**/
- /* a;lsdjf;aslkdjf; */
  /* alskfj;akj;df */
- /*
  * This is a multiline comment
  */
Keywords

• Reserved words with a special meaning
• Cannot be used for anything else, redefined, etc.
• 32 standard C 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
• Become familiar with them
  • I do not expect you to memorize all of them (yet)
Identifiers

• Names for things in a program
• Identifiers are tokens
• Composed of strings of letters, numbers, and _
  • Must start with a letter or _
• Identifiers: scott, x, _foo, score_1, this_is_good
• Not identifiers: 1_score, sc ott, this$, 123, “abc”
• Good identifiers are meaningful, short, clear
• Bad identifiers are confusing, long, vague
Constants

• C objects whose value does not change
• Integers: 0, 17, 325
• Floats: 3.14159265, 37.333, 1.0
• Characters: ‘a’, ‘b’, ‘c’, ‘\n’
• Negatives: -33, -25, -1.0
• Other bases: 017, 0xA4
String constants

• Any sequence of characters inside double quotes
  • “This is a string constant”
  • “asdlfija;sdlfkja;i8hwr. vn”
  • “123”
  • “This is a quote character in a string \“ ”
  • “This is a slash in a string \ ”
• /* “this is not a string” */
Operators and Punctuation

• Some arithmetic operators: +  -  *  /  %
• a+b   --- arithmetic expression
• a_b   --- 3 letter identifier
• % has two meanings
  • printf(“%d\n”, a);   --- format specifier
  • a = b % 7;   --- modulus (remainder) operator
• int main(void)
  {
    int a, b = 2, c = 3;
    a = 17 * (b + c);     /* () are an operator */
Precedence and Associativity

• *Precedence* implicitly specifies the order in which operators are evaluated

• *Associativity* implicitly specifies the order when the precedence is the same

• () explicitly specifies an evaluation order
  • Do the things inside the parentheses first

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>left to right</td>
</tr>
<tr>
<td>++ (postfix)</td>
<td></td>
</tr>
<tr>
<td>-- (postfix)</td>
<td></td>
</tr>
<tr>
<td>+ (unary)</td>
<td>right to left</td>
</tr>
<tr>
<td>- (unary)</td>
<td></td>
</tr>
<tr>
<td>++ (prefix)</td>
<td></td>
</tr>
<tr>
<td>-- (prefix)</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>left to right</td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>%</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>left to right</td>
</tr>
<tr>
<td>-</td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>right to left</td>
</tr>
<tr>
<td>+=</td>
<td></td>
</tr>
<tr>
<td>-=</td>
<td></td>
</tr>
<tr>
<td>*=</td>
<td></td>
</tr>
<tr>
<td>/=</td>
<td></td>
</tr>
<tr>
<td>etc.</td>
<td></td>
</tr>
</tbody>
</table>
Increment and Decrement Operators

- `++i  /* equivalent to i = i + 1 */`
- `x++  /* equivalent to x = x + 1 */`
- `x--    /* equivalent to x = x - 1 */`
- Difference? When it occurs
- NO: 12++
- NO: ++(a * b - 1)
Assignment operators

• \textit{variable} = \textit{expression}
• \texttt{a = b + c;}
• \texttt{b = 2;}
• \texttt{x = (a + 7) * 3 + (27 - time\_of\_day) / 4;}
• \texttt{a = b = c = 0;}
• \texttt{k += 2;}
• More generally: \textit{variable op=} \textit{expression}
Declarations and Initialization

- *Declarations* declare variables to the compiler
  - Specify a type
  - Specify a name
  - May also specify a value;
  - Must come before the rest of the code

- int i;
- int i = 1;
- int start, end, middle = 5;
- float temp = 37.0;
Powers of 2

```c
#include <stdio.h>

int main(void)
{
    int exponent = 0, power_of_two = 1;
    while(++exponent <= 10)
        printf("%5d", power_of_two *= 2);
    printf("\n");
    return 0;
}
```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int i, n;

    printf("\n%s\n%s", "some randomly distributed integers will be printed.",
            "How many do you want? ");
    scanf("%d", &n);
    for(i = 0; i < n; ++i) {
        if(i % 6 == 0)
            printf("\n");
        printf("%9d", rand());
    }
    printf("\n");
    return 0;
}