3. Character Input/Output
ENEE 140

Prof. Tudor Dumitraş
Assistant Professor, ECE
University of Maryland, College Park

http://ter.ps/enee140

Today’s Lecture

• Where we’ve been
  – Variables and Constants
  – Arithmetic operations
  – while loops

• Where we’re going today
  – Increment, relational and logical operators
  – Branching: if statement
  – Loops: for
  – Data types: chars
  – Input and output

• Where we’re going next
  – Functions
**Reminder: Textbook Clarifications**

- If you find the K&R textbook confusing ...

... Consult Steve Summit’s excellent notes on the textbook:

– Linked from the class web page

---

**Increment Operators**

- We’ve seen
  
  \[
  a = a + 1; \\
  \]

  increment by assigning old value + 1

- Increment and decrement operators in C

  \[
  a++; \\
  ++a; \\
  a--; \\
  --a; \\
  \]

  same as \( a = a + 1 \);

  same as \( a = a + 1 \);

  same as \( a = a - 1 \);

  same as \( a = a - 1 \);

  – There is a subtle difference between \( a++ \) and \( ++a \) (more on this later)

- Assignment operations also return the value assigned

  \[
  a = 0; \\
  b = a++; \\
  a = b = 0; \\
  \]

  both a and b become 1

  both a and b become 0
Value of Assignment Expression

- In C, an assignment expression returns the value that is assigned
  \[ b = (a = 0); \]
  \[ a \text{ becomes 0, and } b \text{ also becomes 0} \]

- This means that you can write things like this:
  \[ c = b = a = 0; \]
  \[ a, b \text{ and } c \text{ become 0} \]

Specifying Conditions

- We’ve seen
  \[ \text{while (condition) } \{ \text{ statements executed repeatedly while condition is true} \} \]

  How can we specify the condition?
Relational and Logical Operators

- We’ve seen: **relational operators**, used for specifying conditions
  
  ```
  if (a < b) {...}  \quad \text{condition: if a less than b}
  if (a > b) {...}  \quad \text{condition: if a greater than b}
  if (a <= b) {...} \quad \text{condition: if a less than or equal to b}
  if (a >= b) {...} \quad \text{condition: if a greater than or equal to b}
  if (a == b) {...} \quad \text{condition: if a equal to b}
  if (a != b) {...} \quad \text{condition: if a not equal to b}
  ```

- **Logical operators** are used for combining conditions
  
  ```
  if (cond1 && cond2) {...} \quad \text{condition: both cond1 and cond2}
  if (cond1 || cond2) {...} \quad \text{condition: either cond1 or cond2}
  if (!cond1) {...} \quad \text{condition: not cond1}
  ```

Branching

- Execute statements conditionally
  
  ```
  if (condition) {
      statements
  }
  ```

- Provide alternative to the condition
  
  ```
  if (condition) {
      statements
  } else {
      statements_2
  }
  ```
Loops
• We’ve seen
  ```c
  int i = 0;
  while (i < 10) {
      ...
      i++;
  }
  ```

  *initialize i*
  *test !(exit condition)*
  *increment i*

• Iterate over a set of values
  ```c
  int i;
  for (i = 0; i < 10; i++) {
      ...
  }
  ```

  *iterate over i in [0, 10]*

• **Important:** every loop must have an *exit condition* that eventually becomes true

Common Mistake: Infinite Loops
• While loop example:
  ```c
  int i = 0;
  while (i < 10) {
      printf("%d\n", i);
      if (i > 0) {
          i++;
      }  
  }
  ```

  *i is never incremented*

• For loop example:
  ```c
  int i = 0;
  for (; i < 10 ; ) {
      printf("%d\n", i);
  }
  ```

  *you may omit any of the 3 components of a for statement ... but you must still ensure the loop exit*
Implementation Options for Conditional Execution

- How many times is the block executed?
  ```
  if (i < 10) {
    block of statements
  }
  ```
  0 or 1 times

- How many times is the block executed?
  ```
  while (i < 10) {
    block of statements
  }
  ```
  0→∞ times

- How many times is the block executed?
  ```
  for (i = 0; i <= 10; i++) {
    block of statements
  }
  ```
  11 times (assuming that i is not modified inside the block)

Data Types

- We’ve seen
  ```
  int a = 1;
  float b = 1.1;
  ```
  integer variable  floating-point variable

- Larger data types (can hold larger values)
  ```
  long a = 1;
  double b = 1.1;
  ```
  integer variable  floating-point variable

- Characters
  ```
  char c = 'A';
  char c = '\n';
  ```
  holds one character

- A data type is a set of rules for handling a certain kind of variables
  - Rules govern the interpretation of internal representations and the operations allowed
  - We will discuss the implications of int and float representations in future lectures
  - In C, you must specify the type when declaring each variable
The **char** Data Type

- Internally, characters are represented as integers

- Rules for interpreting the value of the stored data
  
  ```
  char c = 'D' + 1;  // value of c is 'E'
  int diff = 'c' - 'a';  // value of diff is 2
  if (c >= 'A' && c <= 'Z') {...}  // check if c is uppercase
  ```

- A–Z have consecutive codes (numerical values). So do a–z and 0–9
  
  - The offset between the lowercase and uppercase versions of a character is always the same
    
    ```
    'A' - 'a' == 'B' - 'b'
    ```
  
  - Converting a lowercase character to uppercase
    
    ```
    c = c + 'A' - 'a';  // add the offset of the uppercase range
    ```

Reading and Writing Characters

- Read one character from the input
  
  ```
  int c = getchar();
  ```

- Write one character to the output
  
  ```
  putchar(c);
  printf("%c", c);
  ```

- Important: `getchar()` returns an **int** rather than a **char**
  
  - This allows the function to return the special value **EOF** when no more input is available
    
    ```
    while (getchar() != EOF) {
        ... 
    }
    ```
More on the char Data Type

• Internally, characters are represented as integers

• The corresponding value of the character is determined by an encoding scheme
  – For char: American Standard Code for Information Interchange (ASCII)
  – Other encoding schemes: Unicode

• You can examine the internal encoding of characters

```
printf("%d", c);
```

• Good programming practice: Do not rely on the internal values of the encoding
  
  ```
c = c + 'A' - 'a';
c = c - 32;
``` instead of

Review of Lecture

• What did we learn?
  – Increment, relational and logic operators
  – Value of assignment expression
  – if and for statements
  – Character representation
  – Special characters, EOF
  – Character I/O

• Next lecture
  – Functions

• Assignments for this week
  – Read K&R Chapters 1.7, 1.8, 7.2, 7.4, B4
  – Weekly challenge: temperature_conversion_function.c
  – Homework: enee140_lab03.pdf, due on Friday at 11:59 pm
  – Quiz 3, due on Monday at 11:59 pm