CS 240 Programming in C

Control Statements, Operators

September 16, 2019

Schedule

- This class we will do a little recap of pass classes, and continue with getchar() and putchar().
- Later we will get into operators and counting programs which are closely related to our second homework.
- If we still have time we will introduce array.

Recap

- Header file examples.
- For-loop, while-loop
- Getchar() and putchar()

Counting Characters, Version 1

```
#include <stdio.h>

int main(void)
{
  int cnt;

  cnt = 0;
  while (getchar() != EOF)
    cnt++;
  printf("%d\n", cnt);
  return 0;
}
```

Statements: if, else if, else

```
if (condition 1)
  statement 1
else if (condition 2)
  statement 2
[...]
else
  statement n
```

- A way to express multiway decisions
- Conditions are evaluated in order
- If a condition is satisfied, corresponding statement is executed, entire construction is finished
- If no condition is satisfied, else statement is executed
- If there is no else statement, nothing happens
- Note: there can be any number of else ifs

Relational Operators

- Check the relationship between the values of their operands
- The expression always evaluates to 1 (true) or 0 (false)
- `x == y`: the values of x and y are equal
- `x != y`: the values of x and y are not equal
- `x > y`: x is greater than y
- `x < y`: x is less than y
- `x >= y`: x is greater than or equal to y
- `x <= y`: x is less than or equal to y

Assignment Versus Equality Operators

- = assignment operator (not a statement)
- == equality operator
- `if (c == '\n')` tests whether c is the newline character
- `if (c = '\n')` assigns the newline character to c, and then tests whether the newline character is zero

Increment and Decrement Operators

- `++x` is `x = x + 1`
  prefix increments before the variable is used
- `x++` is `x = x + 1`
  postfix increments after the variable is used
- `--x` is `x = x - 1` (prefix)
- `x--` is `x = x - 1` (postfix)
- When used just for the increment/decrement effect, there is no difference
- We will see situations where it makes a big difference

Counting Characters, Version 2

```
#include <stdio.h>

int main(void)
{
  int cnt;

  for (cnt = 0; getchar() != EOF; cnt++)
    ;
  printf("%d\n", cnt);
  return 0;
}
```

- The grammatical rules of C require a for statement to have a body, so we have an empty statement (called a null statement)
- As with while loops, the body is not executed if the condition is false upon entry

Character Constants

- Constants: Fixed values the program may not alter during execution
- A character written between single quotes represents an integer value equal to the numerical value of the character in the machine's character set
- This is another way to write a small integer
- Escape sequences are used to indicate hard-to-represent characters
- Preceded by backslash \
- Count as one character
- `\n` new line
- `\t` horizontal tab
- `\"` double quote
- `\\` backslash
- `\xhhh` hex number, where hhh are hex digits
- `\oooo` octal number, where ooo are octal digits

Count Newlines

```
#include <stdio.h>

int main(int argc, char *argv[]) {
  int c, nl;

  nl = 0;
  while ((c = getchar()) != EOF)
    if (c == '\n')
      ++nl;

  printf("%d\n", nl);
  return 0;
}
```

Count Words

```
#include <stdio.h>
#define OUT 0
#define IN  1
int main(int argc, char *argv[]) {
  int c, numL = 0, numW = 0, numC = 0, state = OUT;
  while ((c = getchar()) != EOF) {
    numC++;
    if (c == '\n')
      numL++;
    if (c == ' ' || c == '\n' || c == '\t')
      state = OUT;
    else if (state == OUT)
      state = IN, numW++;
  }
  printf("%d %d %d\n", numL, numW, numC);
  return 0;
}
```

Counting Digits

```
#include <stdio.h>
int main(void) {
  int c, whiteCnt, otherCnt, i;
  int digitCnt[10];
  for (i = 0; i < 10; i++)
    digitCnt[i] = 0;
  while ((c = getchar()) != EOF) {
    if (c >= '0' && c <= '9')
      digitCnt[c - '0'] ++;
    else if (c == ' ' || c == '\n' || c == '\t')
      whiteCnt++;
    else
      otherCnt++;
  }
  printf("digits =");
  for (i = 0; i < 10; i++)
    printf(" %d", digitCnt[i]);
  printf(", white space = %d, other = %d\n", whiteCnt, otherCnt);
}
```

Logical Operators

- Apply logical functions to Boolean arguments – arguments that evaluate to true or false
- Recall that 0 is false, and nonzero is true
- Evaluated left-to-right
  Evaluation stops as soon as truth or falsehood is known
- NOT operator
  `!x`
  converts a nonzero operand into 0, zero operand into 1
- AND operator
  `x && y && ... && z`
  1 if all operands are true, 0 otherwise
- OR operator
  `x || y || ... || z`
  1 if any operand is true, 0 otherwise

Arrays

- A way to store many values under one name
- Name of array is a pointer to sequential memory locations containing elements of defined type
- `int digitCnt[10];`
  - `int` is the type of elements in the array
  - `digitCnt` is the name of the array
  - 10 is the number of elements in the array
- Array indices always start at 0, so the elements of `digitCnt` correspond to the indices 0 through 9
- Index or subscript
  - Number used to indicate an element of an array
  - Enclosed in brackets following array name
  - `digitCnt[0]` refers to the first element in the array named `digitCnt`
- The size of array is non-changeable once been declared.

Array Initialization

- The general form for declaring an array is

```
type arrayName[arraySize];
int numArray[5]; // declare an array of 5 ints
```

- We can also declare and initialize at the same time

```
int numArray[5] = {0, 1, 2, 3, 4};
```

- If we leave out arraySize, an array just large enough to hold the initialization is created

```
int numArray[] = {0, 1, 2 , 3, 4}; /* same as above */
```

- We can create strings (arrays of chars) using the double quote notation

```
char str[] = "hello";
// array of 6 chars: h e l l o \0
```

Characters as Integers

- `(c >= '0' && c <= '9')`
- Each character constant has an ASCII code
- The ASCII codes for the characters '0' through '9' are 48 to 57 (see ASCII table)
- We can test whether the ASCII code of a character falls within the range
- In other words, check
  `48 <= ASCII code of c <= 57`
- `digitCnt[c - '0'];`
- If we are here, then c must be a digit
- We can subtract the code of '0' from c to determine which digit and the increment the corresponding counter
- For example,
  c is '7' (ASCII code 55)
  c - '0' is 55 - 48
  `digitCnt[c - '0']++` increments the count for 7

Character Arrays (Strings)

- String constant (or literal)
  - A sequence of 0 or more characters surrounded by double quotes
  - Ended with a null character '\0'
- Quotes are not part of the string – they serve only to delimit
- Stored as an array of characters
- We can define/initialize an array to contain the string "hello" and the end of line character:
  `char str[7] = "hello\n";`
- Why do we need 7 slots in this array?