Introduction to Computer Science and Programming in C

Session 5: September 16, 2008

Columbia University

Announcements

- Reminder: Homework 1 is out. Due 9/23
- TA: Peter Lu. Office hours Thurs. 4 PM - 6 PM.
- Use the message boards.
  Counts as class participation.
- If you need to email, send to all three of us for fastest response:
  {bert@cs, ds2664@, yl2505@}columbia.edu

Review

- Finished discussing Hello World
  - Discussion of syntax (#include, statements;)
- Variables and basic types (int, char, float)
- printf(“formatted text”, arg1, arg2,...);

Today

- A few more tidbits on basic variable types
- Advanced types: Arrays and strings
- Input:
  - Reading strings
  - Command Line Input

Tidbit 1: Initialization

- Variables must be **initialized**.
  - ```
    int y;
    printf("%d\n",y);
    ```
  - ```
    int y=0; /* y is definitely 0 */
    printf("%d\n",y);
    ```
- Forgetting to initialize can cause unexpected results.

Tidbit 2: Effects of Casting

- Casting a **float** as an **int** causes **truncation**

```
float a = 3.1;
int x = (int) a;                    /* x is now 3 */
```

- Be careful with math:

```
float a, b, c;
int x = 2, y = 3;
a = x/y;                    /* what happens here? */
b = (float)x/y;
c = (float)x / (float)y;
```

Tidbit 3: Special Operators

- In addition to standard arithmetic (^*/+-), **int** variables have special shortcut operators for **incrementing** and **decrementing**.

```
int i = 0;
i = i+1;        /* Standard way of incrementing i */
i++;            /* After this line, increment i */
++i;            /* Before this line, increment i */
```

- The statements `j = i++;` and `j = ++i;` have different results.

Arrays

- **Array** – a block of variables grouped together:
  - declaration:
    ```
    int score[5];
    ```
  - referenced by **index** (starts at 0)**:**
    ```
    score[0] = 3;
    ```
  - Be careful about indexing!
    ```
    score[5] ???
    ```

Arrays

- Can be “multi-dimensional”
  - `int tictactoe[3][3];`
  - Array of arrays
- Each element should be initialized.

Strings

- Array of characters:
  `char name[30];`
- C has many built in string functions”
  `#include <string.h>`
- `strcopy(name,”Sam”);`
  `int length = strlen(name);`
- Placeholder for printf() is %s
  `printf(“Hello, %s\n”, name);`

Strings

- Size of array is not necessarily length of string:
  ```
  char name[30];
  strcopy(name,"Sam");
  ```
- `strlen(name)` is 3.
- Special character to indicate end of string:
  `\0` (backslash zero)
- `'S' , 'a' , 'm' , '\0'`

Reading Strings

- In <stdio.h>,
```
fgets(string, sizeof(string), stdin);
```
- Copies a string from keyboard (stdin) into "string"
- fgets() will copy the newline ('\n')
- Replace newline with '\0'
```
char name[30];
fgets(name,sizeof(name),stdin);
name[strlen(name)-1] = '\0';
```

Reading Numbers

- Read a string, convert it to a number
- sscanf(string, “formatted text”, &var
  - Stands for “string scanf()”
- ```
  char line[30];
  int age;
  printf(“What is your age?\n”);
  fgets(line, sizeof(line), stdin);
  sscanf(line,”%d\n”, &age);
  ```
  ```

Command-Line

- Sometimes it is cleaner to read input from the command line
  - $ ./multiply 4 5
    4 times 5 is 20
- The program, “multiply”, takes two integers and prints the result of multiplying them.

Command-Line

```
/*
multiply.c – Takes two integers as command line
arguments and displays their product
*/

#include <stdio.h>
int main(int argc, char *argv[])  /* arguments! */
{
    int a, b, c;

    sscanf(argv[1], “%d”, &a);
    sscanf(argv[2], “%d”, &b);
    c = a*b;
    printf(“%d times %d is %d\n”, a, b, c);
    return 0;
}
```

Reading

- Practical C Programming, Chapter 4