C Boot Camp

CMPU 224 – Computer Organization
Jason Waterman
C Basics

• The things that will help you in this course
  • You have seen many of these concepts before
C history

- C
  - Dennis Ritchie in late 1960s and early 1970s
  - systems programming language
    - make OS portable across hardware platforms
    - not necessarily for real applications – could be written in Fortran or PL/I
- C++
  - Bjarne Stroustrup (Bell Labs), 1980s
  - object-oriented features
- Java
  - James Gosling in 1990s, originally for embedded systems
  - object-oriented, like C++
  - ideas and some syntax from C
Comments

• /* any text until */

• // C++-style comments

• Convention for longer comments:
  /*
   * AverageGrade()
   * Given an array of grades, compute the average.
   */
Data objects

• Variable = named container that can hold a value
• default value is (mostly) undefined – treat as random
  • compiler may warn you about uninitialized variables
• Every data object in C has
  • a name and data type (specified in definition)
  • an address (its location in memory)
  • a size (number of bytes of memory it occupies)
  • visibility (which parts of program can refer to it)
  • lifetime (period during which it exists)
• Examples:
  • int x = 42;
  • float y = 42.0;
  • char z = ‘z’;
Arrays/Strings

- Arrays: fixed-size collection of elements of the same type
  - int A[10]; // A is array of 10 int’s

- Strings: Null-character (‘\0’) terminated character arrays
  - Null-character tells us where the string ends
  - All standard C library functions on strings assume null-termination.
Control structures

• Similar to Java
• sequencing: ;
• grouping: {...}
• selection: if, switch
• iteration: for, while
Sequencing and grouping

• statement1 ; statement2; statement n;
  • executes each of the statements in turn
  • a semicolon after every statement
  • not required after a {...} block

• {statements} {declarations statements}
  • treat the sequence of statements as a single operation (block)
  • data objects may be defined at beginning of block
The if statement

• Same as Java
  
  ```java
  if (condition1) {statements1}
  else if (condition 2) {statements2}
  else if (condition n-1) {statements n-1}|
  else {statementsn}
  ```

• evaluates statements until find one with non-zero result
• executes corresponding statements
The switch statement

• Allows choice based on a single value
  
  switch(expression) {
    case const1: statements1; break;
    case const2: statements2; break;
    default: statementsn;
  }

• Effect: evaluates integer expression
• looks for case with matching value
• executes corresponding statements (or defaults)
The switch statement

```c
enum weather{rain, snow, sun};
enum weather w = snow;

switch(w) {
  case rain:
    printf("bring umbrella\n");
  case snow:
    printf("wear jacket\n");
    break;
  case sun:
    printf("wear sunscreen\n");
    break;
  default:
    printf("strange weather\n");
}
```
Repetition

- C has several control structures for repetition

<table>
<thead>
<tr>
<th>Statement</th>
<th>repeats an action...</th>
</tr>
</thead>
<tbody>
<tr>
<td>while(c) {}</td>
<td>zero or more times, while condition is ≠ 0</td>
</tr>
<tr>
<td>do {...} while(c)</td>
<td>one or more times, while condition is ≠ 0</td>
</tr>
<tr>
<td>for (start; cond; upd)</td>
<td>zero or more times, with initialization and update</td>
</tr>
</tbody>
</table>
The break statement

• break allows early exit from one loop level

```plaintext
for (init; condition; next) {
    statements1;
    if (condition2) break;
    statements2;
}
```
The continue statement

• continue skips to next iteration, ignoring rest of loop body
• does execute next statement
  
  ```
  for (init; condition1; next) {
    statement2;
    if (condition2) continue;
    statement2;
  }
  ```

• often better written as an if block
Functions and Program Structure

• Functions break large computing tasks into smaller ones
• Programs may reside in one or more source files
  • Source files may be compiled separately
  • Can be linked with other compiled functions and libraries to form an executable
• A function declaration tells the compiler about a functions name, return type and parameters
• A function definition provides the body of the function
Data objects and pointers

• The memory **address** of a data object, e.g., int x
  • can be obtained via &x (**address of** operator)
  • and has a data type int * (in general, type *)
  • has a value which is a large (8 byte) unsigned integer

• The **size** of a data object, e.g., int x
  • can be obtained via sizeof(x)
  • has data type size_t
  • has a value which is a small(ish) integer
  • is measured in bytes
Data objects and pointers

• Every data type $T$ in C has an associated pointer type $T*$
• A value of type $T*$ is the address of an object of type $T$
• If an object $\textbf{int} \ *xp$ has value $&x$, the expression $*xp$ dereferences the pointer and refers to $x$, thus has type $\textbf{int}$
Data objects and pointers

• If p contains the address of a data object, then *p allows you to use that object

• *p is treated just like normal data object

```c
int a, b, *c, *d;
a = 2; b = 3;
c = &a; d = &b;
*c = 3;
c = d;
```