

198:211 Computer Architecture

Week 2/Part 1
Fall 2010

- Topics:
 - Comparison of Java and C
 - C Programming Language Review

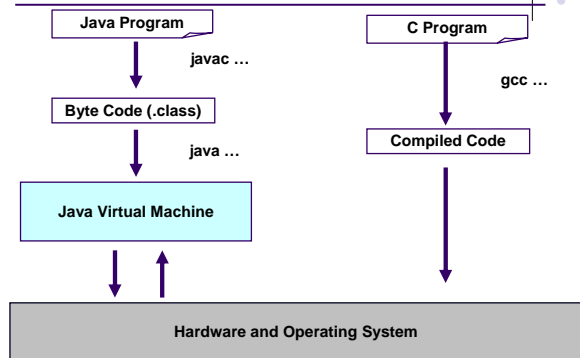
Intro to C

- TAs in the recitations will go over C in detail
 - Other details: cereal machines, accounts etc
 - Compiling, debugging tools (GCC)
- Learn C by programming
 - Don't wait until Programming assignments are due
 - Start by coding, testing small C programs
- Remember you already know JAVA
 - Learning another language is easy

Why C after Java!!

- It is good to be bilingual or multilingual!
 - More job opportunities!!!
- Java is high level Programming language
- C is both high level and low level
 - Better understanding of low-level mechanisms
 - Better Understand Language-architecture interface... Objective of this course
 - Learn C/C++, JAVA, and Python
- Memory-management

Java verse C



Java vs C

Java	C
object-oriented	function-oriented
strongly-typed	Flexible (cast)
No pointers	pointers
Automatic memory mgmt	Left to programmer
Strings as type	Only char arrays
layered I/O model	byte-stream I/O

Java vs C

```
public class hello {
    public static void main (String args
    []) {
        System.out.println
            ("Lady Gaga");
    }
}

#include <stdio.h>
int main(int argc, char
*argv[])
{
    printf("Lady Gaga\n");
    /* \n is linefeed, \t
    tab */
}
```

Data types

```
main() { int a, b, c, sum;
a = 1; b = 2; c = 3;
sum = a + b + c;
printf("sum is %d", sum);
}
```

```
main() { int a, b;
float c, sum;
a = 1; b = 2; c = 3.5;
sum = a + b + c;
printf("sum is %f", sum);
}
```

Numeric data types

- char
 - Individual characters (Range 127 to -128)
- int
 - Integers
 - Short (-65536 to 65535) or
 - Long -2,147,483,648 to 2,147,483,647
- float
 - Real numbers 3.4 E +/- 38 (32 bits long)
- double
 - Real numbers with double precision 3.4 E +/- 308 (64 bits long)
- Modifiers
 - Short (16 bit), long (32bit)
 - Control the range of numbers
 - signed, unsigned
 - for integers and whole numbers respectively

Arithmetic Operators

Symbol	Operation	Usage
• *	multiply	$x * y$
• /	divide	x / y
• %	modulo	$x \% y$
• +	addition	$x + y$
• -	subtraction	$x - y$

- All associate left to right.
- * / % have higher precedence than + -.

Special Operators: ++ and --

- Changes value of variable before (or after) its value is used in an expression.

Symbol	Operation	Usage
• ++	postincrement	$x++$
• --	postdecrement	$x--$
• ++	preincrement	$++x$
• --	predecrement	$--x$

- **Pre:** Increment/decrement variable **before** using its value.
- **Post:** Increment/decrement variable **after** using its value.

Examples

```
#include <stdio.h>
main()
{
    int i = 3, j = 4, k;
    k = i++ + --j;
    printf("i = %d, j = %d, k = %d", i, j, k);
}

#include <stdio.h>
#include <stdlib.h>
int main()
{
    char weight[4];
    int w;

    w = 140;
    printf("Here is what you weigh now: %d\n", w);
    w--;
    printf("w--: %d\n", w);
    w++;
    printf("++w: %d\n", w);
    printf("pre DECR %d\n", --w);
    printf("post INCR %d\n", w++);
    printf("value of w %d\n", w);
    return(0);
}
```

Relational Operators

Symbol	Operation	Usage
• >	greater than	$x > y$
• >=	greater than or equal	$x >= y$
• <	less than	$x < y$
• <=	less than or equal	$x <= y$
• ==	equal	$x == y$
• !=	not equal	$x != y$

- Result is 1 (TRUE) or 0 (FALSE).
- **Note:** Don't confuse equality (==) with assignment (=).

Logic Operators

- | Symbol | Operation | Usage |
|--------|-------------|--------|
| ! | logical NOT | !x |
| && | logical AND | x && y |
| | logical OR | x y |
- Treats entire variable (or value) as TRUE (non-zero) or FALSE (zero).
 - Result is 1 (TRUE) or 0 (FALSE).

Bit operators

- In C, there are operators that work on bits of a word
- & logical AND
- | inclusive OR
- ~ NOT
- Example
 - x = 8 y = 7
 - x & y
 - x | y
 - !X
 - E.g., 72 & 184 = 8 ; 72 | 184 = 248 ;

Variable Declarations

- Variables are used as names for data items.
 - Each variable has a *type*, which tells the compiler how the data is to be interpreted (and how much space it needs, etc.).
- ```
#include <stdio.h>

int main()
{
 int pints=1;
 float price = 1.45;

 printf("You want %d
 pint.\n",pints);
 printf("That be $%f
 , please.\n", price);
 return(0);
}
```
- `int counter;`
  - `int startPoint;`
  - `Float pi=3.14;`
  - `int` is a predefined integer type in C.

## Control Structures

- Same control structures as Java. Same syntax
- **Conditional**
  - if
  - if-else
  - switch
- **Iteration**
  - while
  - for
  - do-while
- also has the `break` and `continue` expressions.

## Control Structures

- Same control structures as Java. Same syntax
- **Conditional**
  - if
  - if-else
  - switch
- **Iteration**
  - while
  - for
  - do-while
- also has the **break** and **continue** expressions.

## Sequencing and grouping

- Statement\_1 ; statement\_2; statement \_n;
  - executes each of the statements in turn
  - a semicolon after every statement
  - not required after a {...} block

## The if statement

- Same as Java

```
if (condition1)
{statements1}
```

```
else if (condition2)
{statements2}
```

```
else if (conditionn)
{statementsn-1}
```

```
else {statementsn}
```
- evaluates statements until find one with non-zero result
- executes corresponding statements

```
#include <stdio.h>

main()
{
 int i=3,j=5;
 if(i<j) printf("i<j\n");
}

#include<stdio.h>

main()
{
 int i=5,j=3;

 if (i > j)
 i=i+1;
 printf("%d",j);
 }
 else
 j=j+1;
}
```

Examples from: <http://www.java2s.com/Tutorial/C/>

## 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

```
int fork;
switch(fork) {
 case 1:
 printf("take left");
 case 2:
 printf("take right");
 break;
 case 3:
 printf("make U turn");
 break;
 default:
 printf("go straight");
}
```

## Repetition

- C has several control structures for repetition

| Statement                 | repeats an action...                                                       |
|---------------------------|----------------------------------------------------------------------------|
| while(c) {}               | zero or more times, while c ≠ 0<br>Remember: True means any non-zero value |
| do {...} while(c)         | one or more times,<br>while condition is ≠ 0                               |
| for (start; cond; update) | zero or more times, with<br>initialization and update                      |

## while loop and for loop

```
#include <stdio.h>
main(){
 int i = 0;
 while (i < 5){
 printf("the value of i is %d\n", i);
 i = i + 1;
 }
}

#include <stdio.h>
main(){
 int i, n = 5;
 for(i = 0; i < n; i = i + 1)
 { printf("the numbers are %d \n", i);
 }
}
```

Examples from: <http://www.java2s.com/Tutorial/C/>

## The break statement

- **break** allows early exit from one loop level

```
for (init; condition; next) {
 statements1;
 if (condition2) break;
 statements2;
}

#include <stdio.h>
main(){
 int i = 0;
 while (1)
 {
 i = i + 1;
 printf("the value of i is %d\n", i);
 if (i > 5) {
 break;
 }
 }
}
```

## The continue statement

- continue skips to next iteration, ignoring rest of loop body

```
#include<stdio.h>

main(){
 int i;
 for(i = -10; i < 11; i++){
 if (i<0) {
 continue;
 }
 printf(" the value of i is %d\n", i);
 }
}
```

## Functions in C

- Functions in C are similar to methods in Java (minus the associated objects).
- Functions are pass-by-value.
- Using functions has three aspects:
  1. Writing the function declaration.
  2. Calling the function.
  3. Writing the function body.
- Writing the function declaration is not always needed but generally preferred.

## Functions in C

- Declaration (also called prototype)

```
int Factorial(int n);
```

type of return value

name of function

types of all arguments

- Function call -- used in expression

```
a = x + Factorial(f + g);
```

3. use return value in expression

2. execute function

1. evaluate arguments

## Function Definition

- State type, name, types of arguments
  - must match function declaration
  - give name to each argument (doesn't have to match declaration)

```
int Factorial(int n)
{
 int i;
 int result = 1;
 for (i = 1; i <= n; i++)
 result *= i;
 return result;
}
```

gives control back to calling function and returns value

## Input and Output

- Variety of I/O functions in *C Standard Library*.
- Must include `<stdio.h>` to use them.
- `printf("%d\n", counter);`
  - String contains characters to print and formatting directions for variables.
  - This call says to print the variable `counter` as a decimal integer, followed by a newline (`\n`).
- `scanf("%d", &startPoint);`
  - String contains formatting directions for looking at input.
  - This call says to read a decimal integer and assign it to the variable `startPoint`. (Don't worry about the `&` yet.)

## Output Examples

- This code:

```
printf("%d is a prime number.\n", 43);
printf("43 plus 59 in decimal is %d.\n",
43+59);
printf("43 plus 59 in hex is %x.\n", 43+59);
printf("43 plus 59 as a character is %c.\n",
43+59);
```
- produces this output:
  - 43 is a prime number.
  - 43 + 59 in decimal is 102.
  - 43 + 59 in hex is 66.
  - 43 + 59 as a character is f.

## More About Output

- Can print arbitrary expressions, not just variables.
- `printf("%d\n", startPoint - counter);`
- Print multiple expressions with a single statement.
- `printf("%d %d\n", counter, startPoint - counter);`
- Different formatting options:
  - `%d` decimal integer
  - `%x` hexadecimal integer
  - `%c` ASCII character
  - `%f` floating-point number

## Output Examples (continued)

- Formatting instructions can contain additional information:
  - **Min Field Width.Precision**
  - **Min Field Width** The minimum number of spaces the number is allow to occupy.
  - **Precision** Float: Num of digits to the right of decimal point  
Int: Min Number of digits to be printed  
String: Number of chars from string to print
- ```
int iv = 12345;
```

<code>printf("%2.3d\n", (iv));</code>	12345
<code>printf("%10d\n", (iv));</code>	12345
<code>printf("%10.5f\n", (3.1456123));</code>	3.14561
<code>printf("%10.2f\n", (3.1456123));</code>	3.15
<code>printf("%.2f\n", (3.1456123));</code>	3.15

Examples of Input

- Many of the same formatting characters are available for user input.
- `scanf("%c", &nextChar);`
 - reads a single character and stores it in nextChar
- `scanf("%f", &radius);`
 - reads a floating point number and stores it in radius
- `scanf("%d %d", &length, &width);`
 - reads two decimal integers (separated by whitespace), stores the first one in length and the second in width
- Must use ampersand (&) for variables being modified.
(Explained later when we talk about pointers)
- Exactly how this matching is done will be covered later.

Comments

- Begins with `/*` and ends with `*/`
- Can span multiple lines.
- Cannot have a comment within a comment.
- Comments are not recognized within a string.
 - example: `"my/*don't print this*/string"`
would be printed as: `my/*don't print this*/string`
- **As before, use comments to help reader, not to confuse or to restate the obvious**

main Function

- Every C program must have a function called `main()`.
- This is the code that is executed when the program is run.
- The code for the function lives within brackets:
 - `main()`
 - `{`
 - `/* code goes here */`
 - `}`