Getting Started With C

Java and *C*

- Java is derived from *C.*
- Many of its syntactic characteristics are similar to *C.*
- However, there are some huge differences.

Arithmetic Operators

- Arithmetic operators are the same:
  `+, -, *, /, %, ++, --`

Relational Operators

- Relational operators work the same way but return different results:
  `>, >=, <, <=, ==, !=`
- In Java, they return values `FALSE` and `TRUE`
- In C, they return values `0` and `1`
- In C,
  - a value of 0 means *false*
  - any value that is not zero means *true*
  - e.g., 1, 5, -1000000, 3.14159, 6.626068×10^{-34}

Conditional and Bitwise Operators

- Conditional execution operators are same in Java and C:
  `||, &&, ?` (followed by) `:`
- Bitwise operators are same in Java and C: (TBD)
  `|, &, ^` for bit-by-bit operations with a word
- Shift operators differ a little bit (TBD)
  `<<` (left shift) is the same
  `>>` (right shift) is machine dependent in C
  - i.e., whether to fill from left with zeros or sign bits

Assignment and Unary Operators

- Assignment operators work the same:
  `=, +=, -=, *=, /=, &=, |=, ^=`
- The following unary operators are available C but not in Java (TBD)

`~`	invert the bits of a word
`*`	pointer dereference
`&`	pointer creation
`(type)`	cast (i.e., forceable type conversion)
`sizeof`	# of bytes in operand or data type
`->`	pointer dereference with field selection

Statements

- Statements in *C*:
  - Labeled statement
  - Expression statement
  - Compound statement
  - Selection statement
  - Iteration statement
  - Jump statement

Statements

- Statements in *C*:
  - Labeled statement
  - Expression statement
  - Compound statement
  - Selection statement
  - Iteration statement
  - Jump statement

E.g., cases of a **`switch`** statement
Similar to Java

Statements

- Statements in *C*:
  - Labeled statement
  - Expression statement
  - Compound statement
  - Selection statement
  - Iteration statement
  - Jump statement

Any expression followed by **`';'`**
Much like to Java

Statements

- Statements in *C*:
  - Labeled statement
  - Expression statement
  - Compound statement
  - Selection statement
  - Iteration statement
  - Jump statement

Sequence of statements enclosed in **`"{}"`**
Called a "block" in Java

Statements

- Statements in *C*:
  - Labeled statement
  - Expression statement
  - Compound statement
  - Selection statement
  - Iteration statement
  - Jump statement

```
switch (expr)
if (expr)statement
if (expr) statement
  else statement
```
Same as in Java

Statements

- Statements in *C*:–
  - Labeled statement
  - Expression statement
  - Compound statement
  - Selection statement
  - Iteration statement
  - Jump statement

```
while (expr) statement
do statement while (expr
for (exp1; exp2, exp3)
  statement
```
Very similar to Java

Statements

- Statements in *C*:
  - Labeled statement
  - Expression statement
  - Compound statement
  - Selection statement
  - Iteration statement
  - Jump statement

`break;`
`continue;`
`return expr`
Very similar to Java

Formatted Input & Output

- Very different between *C* and *Java*
- Handled by library functions in *C*
  - `printf()`
  - `scanf()`
  - `getc()`
  - `putc()`
  - Many others!

`printf()` – Print formatted data

`printf("string containing '%' specifiers", expr1, expr2, expr3, …);`

- Copy the string, character-by-character, to the output.
- When the *ith* `'%'` is encountered, treat it as a *conversion specifier* for converting the *value* of $\texttt{expr}_i$
  - Copy the converted value to the output per instructions encoded in the *conversion specifier*
- Return number of characters printed

`printf()` conversion specifiers

- `%d` or `%i`
  - Treat expression as a decimal number (with sign)
- `%u`
  - Treat expression as unsigned decimal number (TBD)
- `%f`
  - Treat expression as double precision floating point number; print without exponent
- `%e` or `%E`
  - Treat expression as double precision floating point number; print with exponent (base 10) — *scientific notation* (TBD)
- `%c`
  - Treat value of expression as the code for a single character
- `%s`
  - Treat expression as a pointer to a string (TBD)
- **…**

`printf()` conversion specifiers, cont'd

- Conversion specifiers may optionally contain
  - Right or left alignment in the field
  - Minimum field width (padded on right or left)
  - Precision – i.e.,
    - Maximum length of string
    - Number of decimal places of floating point value
- Examples
  - `%6d` – print signed decimal number in 6-char field
  - `%8.4f` – print floating point number with four places after decimal point, field width of 8 characters

`scanf()` – Scan formatted data

`scanf("string containing '%' specifiers", &var1, &var2, &var3, …);`

- Scan the input, matching the string character by character.
- When the *ith* `'%'` is encountered, treat as a *conversion specifier* for converting next sequence of characters and storing result in $\texttt{var}_i$
  - Copy the converted value to the output per instructions encoded in the *conversion specifier*
- Stop if input does not match string or conversion not successful
- Return number of successful conversions.

`scanf()` – Typical Usage

```
int j;
double x;
scanf("%d %f", &j, &x);
```

- Scan the input, skipping blanks and tabs
- Try to match a signed integer; if successful, store result in `j`
- Continue scanning, skipping blanks and tabs
- Try to match a floating point number. If successful, store in `x`
- Return number of items stored.

Your First C Program

```
#include <stdio.h>

int main()
{
  printf("Hello, CPSC 275!\n");
  return 0;
}
```


Compiling and Running C Programs

- To compile:

```
$ gcc -o hello hello.c
```

compiler output source

- To run:

```
$ ./hello
```


Compiling and Linking Using `gcc`

- Linking is automatic when using `gcc`; no separate link command is necessary.
- Without the `-o` option, after compiling and linking the program, `gcc` leaves the executable program in a file named `a.out` by default.

Compiling and Linking

- Before a program can be executed, three steps are usually necessary:
  - ***Preprocessing.*** The ***preprocessor*** obeys commands that begin with # (known as ***directives***)
  - ***Compiling.*** A ***compiler*** translates then translates the program into machine instructions (***object code***).
  - ***Linking.*** A ***linker*** combines the object code produced by the compiler with any additional code needed to yield a complete executable program.
- The preprocessor is usually integrated with the compiler.

Exercise 1

Write a C program which will read an integer, which represents the temperature in Fahrenheit, convert and print it in Celsius. Use the following formula:

$$C = (5/9)\ (F - 32)$$

Exercise 2

Write a C program which will read two integers, **`low`** and **`high`**, which represent a range of temperatures in Fahrenheit, and an integer **`step`**, print a Fahrenheit-Celsius conversion table. For example, if **`low`** = 100, **`high`** = 200, **`step`** = 20, then the table would look like:

```
100   37
120   48
140   60
160   71
180   82
200   93
```

25

Exercise 3

Modify the temperature conversion program to print the table in reverse order, that is, from **`high`** to **`low`**.

26