

# Basic C Syntax

Comp-206 : Introduction to Software Systems  
Lecture 10

Alexandre Denault  
Computer Science  
McGill University  
Fall 2006

# Next Week

- I'm away for the week.
  - ◆ I'll still check my mails though.
- No class Tuesday, since it counts as a Monday.
- Thursday class will be give by Jun, one of the T.A

# Quiz

- In addition to a process id, what is allocated to a process when it is created?
- What makes a good password?
- What are the three file permission levels?
- Give two commands that allows you to scroll, page-by-page, through the content of a file.
- What command can be used to see the list of currently running processes?
- What is the different between a command-line text editor and GUI text editor?
- What positional variable contains the number of arguments on the command line?
- Give a test command that will determine if "test.sh" is executable.
- Name two differences between Java and C.

# Structure of a C program

- A C program has the following components (usually found in this order) :
  - ◆ Preprocessor Commands
  - ◆ Type Definitions
  - ◆ Function Prototypes
  - ◆ Variables
  - ◆ Functions
- Every C program must have a main function.

# Compile Time vs Run Time

This is when the compiler is working on your program.

- The compiler knows the name and type of every variable.
- Errors are explained, and usually a suggestion about which line has an error is given.

This is when your application is running.

- The operating system has no idea what your variable are, or what type they have.
- The operating system has no idea what lines of codes are, or what errors can occur.

# Importance of clean coding

- When programming in C, a clean coding style is mandatory.
- Compile time errors are cryptic at best. Don't expect too much help from the compiler.
- Runtime errors are worst. Since a compiled executable has very little debugging information, the errors are even more cryptic.
  - ◆ Core Dump
  - ◆ Segmentation Fault

# Variable Declaration

- Variables are usually declared at the top of files and functions.

```
#include <stdio.h>

int myglobalinteger;

int main() {

    int mylocalinteger;

    // do something
}
```

# Assigning a value

- Just like Java, you can assign a value to a variable using the equal sign.
- In C, you can chain assignments.
- Unlike Java, variables are NOT defaulted to 0;

```
int main() {  
  
    int a,b,c,d;  
  
    a = 10;  
    b = c = 5;  
    printf("a:%d, b:%d, c:%d, d:%d", a, b, c, d);  
}
```

# typedef

- The typedef command allows for the creation of custom types.
- This will become useful latter in the course.

```
typedef scalefactor int;
```

```
int main() {
```

```
    scalefactor a;
```

```
    a = 10;
```

```
    printf("The scale factor is:%d", a);
```

```
}
```

# Constants

- In C, a variable can be declared as constant.
- The value of a constant is initialized when the variable is declared. That value cannot be changed.
- An optimizing compiler can use the constant declaration to simplify and optimize the code.

```
int const a = 1;
```

```
const int a = 2;
```

# Arithmetic Operations

- C provides the basic arithmetic operations : + - \* /
- For efficiency purposes, it also provides an increment and decrement operator : ++ and --
- The modulus (%) operator is also provided.
- Note that / operation for float and integer is very different. Unless both operands are float, the division will be integer based.

```
float a, b;  
a = 3.0 / 2;    // a = 1.0  
b = 3.0 / 2.0; // b = 1.5
```

# Comparison Operators

- C provides the following comparison operators:
  - ♦ `==` : equality
  - ♦ `!=` : not equal
  - ♦ `<` : smaller than
  - ♦ `>` : greater than
  - ♦ `<=` : smaller or equal than
  - ♦ `>=` : greater or equal than
- Please note that testing for equality is done using the `==` operator, which is not the same as `=`

```
if ( a == 1 ) { // Good test
```

```
if ( a = 1 ) { // Assigns the value of 1 to a  
              // and will always test as true
```

# Logical Operators

- C provides the following logical operator:
  - ♦ `&&` : AND
  - ♦ `||` : OR
  - ♦ `!` : NOT
- These can be used with the comparison operators:

```
if ( (a == 5) || (a == 6) ) // a = 5 or 6
if ( (a == 3) && (b == 4) ) // a = 3 and b = 4
if ( !(a == 5) )           // a is not 5
if ( a != 5 )             // a is not 5
```

# If statement

- If statements in C are identical to if statements in Java.

```
if (expression) {  
    statement;  
} else if (expression) {  
    statement;  
} else {  
    statement;  
}
```

- If you omit the bracket, then you are limited to one statement in your if block.
- Given the complexity of C debugging, ALWAYS put your brackets.

# ? operator

- The ? operator is designed to replace small if statements. Its syntax is as follows:
  - (expression) ? (statement if true) : (statement if false)
- The following example calculates the absolute value of an integer.

```
int a, aabs;
```

```
a = some random int value;
```

```
aabs = (a > 0) ? a : -a;
```

# Switch statement

- A switch statement allows testing of a variable under multiple condition:

```
switch(variable) {  
    case constant1:  
        statements;  
        break;  
    case constant2:  
    case constant3:  
        statements;  
        break;  
    default:  
        statements;  
}
```

# Break keyword

- Note that the break keyword is necessary. Otherwise, the evaluation will fall through the next block.

```
switch(variable) {  
    case constant1:  
        statements;  
    case constant2:  
    case constant3:  
        statements;  
        break;  
    default:  
        statements;  
}
```

# For loop

- The for loop in C is identical to its Java counterpart.

```
for (expression1; expression2; expression3) {  
    statements;  
}
```

- It's components are as follows:

- Expression 1 is used for setting the initial value of the loop.
- Expression 2 is the condition that is tested at each iteration. If the expression is evaluated as false, the loop terminates.
- Expression 3 is executed as every iteration. It is usually used to increment a counter.

# While loop

- The while loop is very similar to a for loop.

```
while (expression) {  
    statements;  
}
```

- The statements in the loop will be executed until the expression is evaluated as false (as equal to zero) .
- This makes the following while loop legal:

```
int i = 10;  
while (i-->0) {  
    statements;  
}
```

# Every for loop is a while loop

- The following for loop ...

```
for (expression1; expression2; expression3) {  
    statements;  
}
```

- ... could be transformed as the following while loop.

```
expression1;  
while (expression2) {  
    statements;  
    expression3;  
}
```

# Every while loop is a for loop

- The following for loop ...

```
while (expression) {  
    statements;  
}
```

- ... could be transformed as the following while loop.

```
for ( ;expression; ) {  
    statements;  
}
```

# Break and Continue keywords

- The control flow of a loop can be altered using the break or continue keyword.
  - ♦ continue will skip to the end of the current iteration to the next iteration.
  - ♦ break will exit the loop (just as it exits a switch statement).
- For example, the following loop will print out the modulus of 3 smaller than 10.

```
i = 0;
while(1==1) {
    if ((i%3)!=0) continue;
    if ((i >= 10)==0) break;
    printf("%d\n", i);
}
```

# Arrays

- In C, arrays are blocks of memory.
- They can be single or multi dimensional.
- Arrays will get much more powerful when we start working with pointers.
- Declaring an array is pretty simple:  

```
int listofint[50];
```
- Using an array is also straight forward  

```
listofint[0] = 10;
```
- C does NOT do bounding checks, so be careful.
- Also not that the content of the array is initialized with whatever is lying around in memory.

# Array of Characters

- In C, strings are null terminated arrays of characters.
- However, C has no built-in facilities to deal with strings.
- The following assignment would be **illegal**;

```
char[50] myString;  
myString = "Hello World!";
```

- C has special functions to deal with Strings. We will take a look at them in a latter lecture.

|    |     |     |     |     |    |    |     |     |     |     |    |    |    |    |
|----|-----|-----|-----|-----|----|----|-----|-----|-----|-----|----|----|----|----|
| H  | e   | l   | l   | o   |    | W  | o   | r   | l   | d   | !  | \0 |    |    |
| 72 | 101 | 108 | 108 | 111 | 32 | 87 | 111 | 114 | 108 | 100 | 33 | 0  | 33 | 66 |

# Functions

- As previously mentioned, functions in C are similar to their Java counterpart.
- The following example function adds two integers together:

```
int add (int a, int b) {  
    return a + b;  
}
```

- Each function name must be unique. C does not support function overloading.

# Function Prototyping

- C uses a single pass compiler. This means that when compiling, each file is only read once.
- When checking the code for correctness, the compiler goes from top-to-bottom.
- If a function uses a function that was defined after it, the compiler will report an error.
- Function prototyping allows us to declare a future function, without having to give the code for it.
- It is considered good practice to declare all your functions (except for main) at the top of your file (or the header if you are using one).

# Incorrect code

```
main() {  
    a();  
    b();  
}
```

```
void a() {  
    //do something  
}
```

```
void b() {  
    //do something  
}
```

# Better code

```
void a() {  
    //do something  
}
```

```
void b() {  
    //do something  
}
```

```
main() {  
    a();  
    b();  
}
```

# Even better code

```
void a(); ← Function prototype  
void b(); ←
```

```
main() {  
    a();  
    b();  
}
```

```
void a() {  
    //do something  
}
```

```
void b() {  
    //do something  
}
```

# Popular Function

- `printf` is the default command to print out data to the command line (STDOUT).
- It is a very popular command, since it exists in many programming language, including Java.
- It is, in many ways, similar to `System.out.print`.
- However, since C does not use type information at runtime, it is a little trickier to use.

- `printf` takes a variable number of arguments, the first being a format string.
- The format string contains the string to output with variable tags.
  - For example : `printf("The temperature is %d. \n", temperature);`
  - Variable tags are denoted by a percent sign `%` and a code.
  - In this case, `%d` is use to indicate an integer.
  - The second argument will replace the first tag.
  - If a second tag was used, it would be replaced by the third argument.
  - The string is terminated by `\n`. This is a newline character.

# Printf Conversion

- %d or %i : signed integer
- %x : unsigned hexadecimal integer
- %u : unsigned decimal integer
- %c : unsigned char
- %s : char\* (string)
- %f : float or double of the form [-]mmm.ddd
- %m.df : float or double of the form [-]mmm.ddd where m and d specifies the maximum number of digits.
- %E : double of the form [-]m.dddExx

- The simplest input function is `getchar`.

```
int getchar(void)
```

- It retrieves one character from STDIN.

- You can combine redirection and `getchar` to create a simple program that reads from a file.

```
readingapp <text.txt
```

- The output equivalent function is `putchar`.

```
int putchar(int)
```

- It displays one character to STDOUT

- Again, you can combine redirection and `putchar` to create a simple program that writes a file.

```
writingapp >output.txt
```