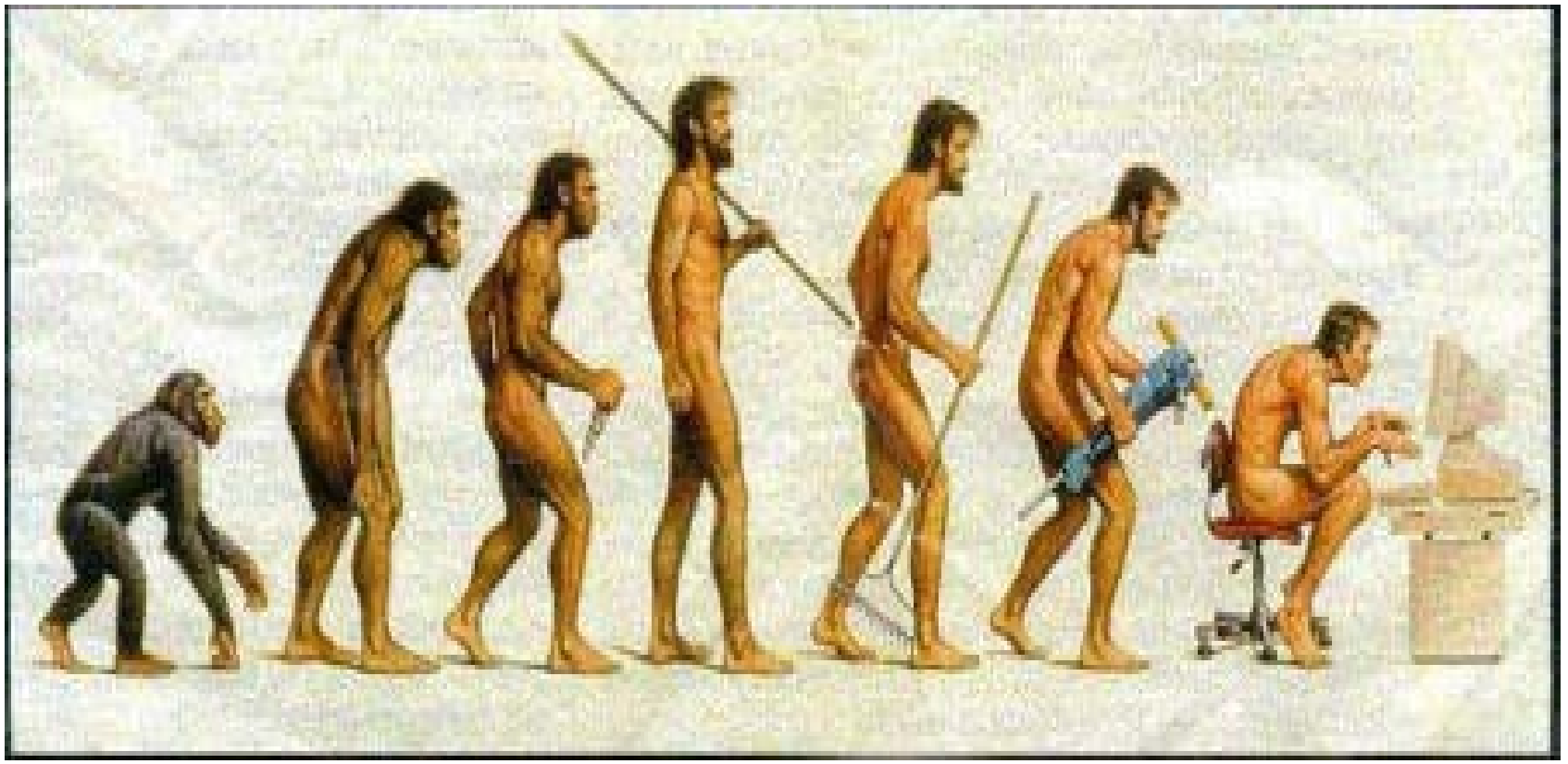
Programming Language (C)

Nalini Vasudevan
Columbia University

Why C Programming?



Why C Programming?

- Provides low-level access to memory
- Provides language constructs that map efficiently to machine instructions

About the course

Instructor: Nalini Vasudevan

Email: naliniv@cs.columbia.edu

Office Hours: 467 CSB (Open Door)

Course Time: Wednesdays, 11 am to 1 pm

Course Location: 1127 Mudd

Credits: 1

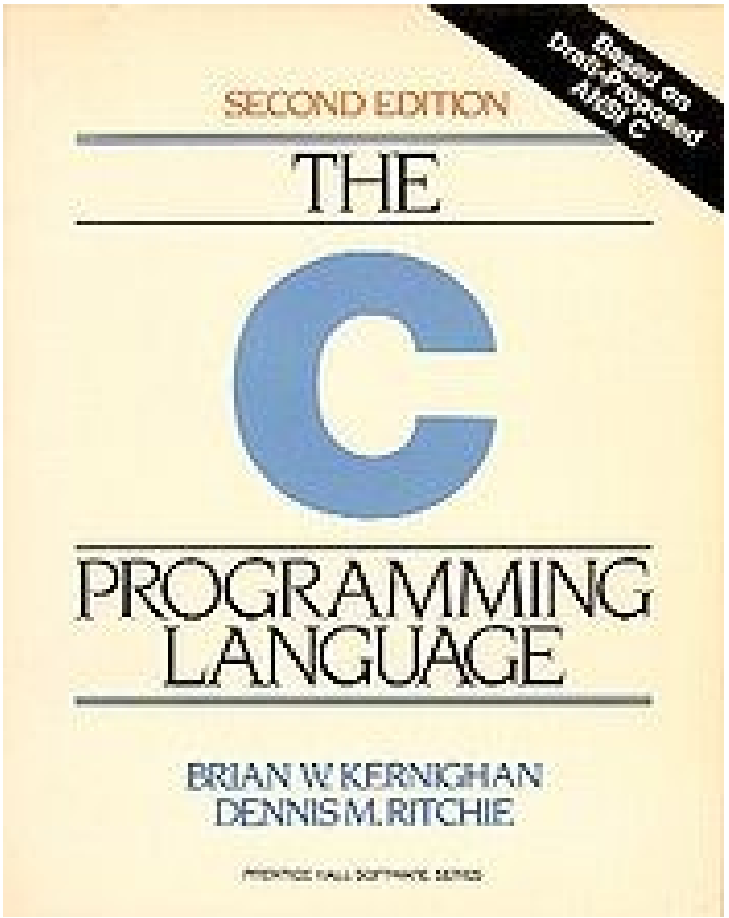
Course Duration: 9/9/09 to 10/14/09

Course Website: http://www.cs.columbia.edu/~naliniv/2009-fall-3101.php

Grading

- Homeworks: 60%
- Project: 40%
- Class Participation: Extra credit (Not applicable for CVN students)
- No Exam

Textbook


SECOND EDITION
THE
C
PROGRAMMING
LANGUAGE
BRIAN W. KERNIGHAN
DENNIS M. RITCHIE

Miscellaneous

- Questionnaire
- Notes

Let's add two numbers

Adding two numbers

```
x = 5;
y = 10;
```

Adding two numbers

```
x = 5;
y = 10;
z = x + y;
```

Adding two numbers

```
int x, y, z;
x = 5;
y = 10;
z = x + y;
```

Adding two numbers

```
int x, y, z;
x = 5;
y = 10;
z = x + y;
printf ("The sum is %d", z);
```

Adding two numbers

```
main()
{
   int x, y, z;
   x = 5;
   y = 10;
   z = x + y;
   printf ("The sum is %d", z);
}
```

Adding two numbers

```
#include<stdio.h>

main()
{
   int x, y, z;
   x = 5;
   y = 10;
   z = x + y;
   printf ("The sum is %d", z);
}
```

Adding two numbers

```
#include<stdio.h> /*Header file*/

main() /* The main function */
{
   int x, y, z; /*Variable
Declaration*/
   x = 5;
   y = 10;
   z = x + y;
   printf ("The sum is %d", z);
}
```

Program Compilation and Execution

- To compile

```
gcc -o add add.c
```

  - "-o" place the output in file add
  - "add" is the executable file

- To run

```
./add
```

C statements

- Examples
  - `x  = y + 3; /*Assignment*/`
  - `printf(”hello”); /*Function call*/`
  - `int x; /*Variable Declaration*/`
- End with a semicolon

Variables

- Hold values, must be declared before use
- Example

```
a = 3 + 4; /*a is a variable*/
```

- Types
  - `int a;  /*Integer values like 1, 44, - 26*/`
  - `char a; /*Characters like a, b, $, #, \n*/`
  - `float a; /*Decimal fractions like 0.1, 2.3*/`

int

- 4 bytes (compiler dependent)
  - A total of 2^ 32 values
  - -2^31 to 2^31 – 1
- Variants
  - `short int a; /* 2 bytes */`
  - `long int a; /* 8 bytes */`
  - `unsigned int a; /* Only positive numbers */`
    - 0 to 2^32 - 1

char

- Example
  - `var = 'x' ;`
- 1 byte
  - A total of 2^8 values
- ASCII representation
  - Ascii value of 'a' is 97
  - Ascii value of 'b' is 98

float

- Floating decimal point
- Example

```
float a;
a = 2.54;
```

- 4 bytes
  - IEEE format
  - -3.4e ^ 38 to 3.4e ^ 38

`double`

- Twice the memory as float
  - 8 bytes (generally)

sizeof

```
#include<stdio.h> /*Header file*/

main() /* The main function */
{
   int x; /*Variable Declaration*/
   printf ("x is %d bytes", sizeof(x));
}
```

Casting

- We can cast a variable to a different type than its actual type

```
int x;
float y;
x = 3;
y = (float) x; /* Explicit casting */
y = x;  /* Implicit casting */
```

printf

- Example

```
printf ("The sum of %d and %d is
  %d", x, y, z);
```

  - Ouput

```
The sum of 5 and 10 is 15
```

- Placeholders
  - `%d` int
  - `%f` float
  - `%c` char

printf

- Example

```
printf ("Hello! ");
printf ("The sum of %d and %d is
  %d", x, y, z);
```

  - Ouput

```
Hello! The sum of 5 and 10 is 15
```

printf

- Example

```
printf ("Hello!\n");
printf ("The sum of %d and %d is
  %d", x, y, z);
```

  - Ouput

```
Hello!
The sum of 5 and 10 is 15
```

Adding two numbers

```
#include<stdio.h> /*Header file*/

main() /* The main function */
{
   int x, y, z; /*Variable
Declaration*/
   x = 5;
   y = 10;
   z = x + y;
   printf ("The sum is %d", z);
}
```

scanf

```
#include<stdio.h> /*Header file*/

main() /* The main function */
{
   int x, y, z; /*Variable Declaration*/
   printf("Enter x:");
   scanf("%d", &x); /* Wait for input */
   printf ("Enter y:");
   scanf ("%d", &y); /* Wait for input */
   z = x + y;
   printf ("The sum is %d", z);
}
```