XSEDE Scholars Program
Introduction to C Programming

John Lockman III

June 7th, 2012

Homework 1
Problem 1

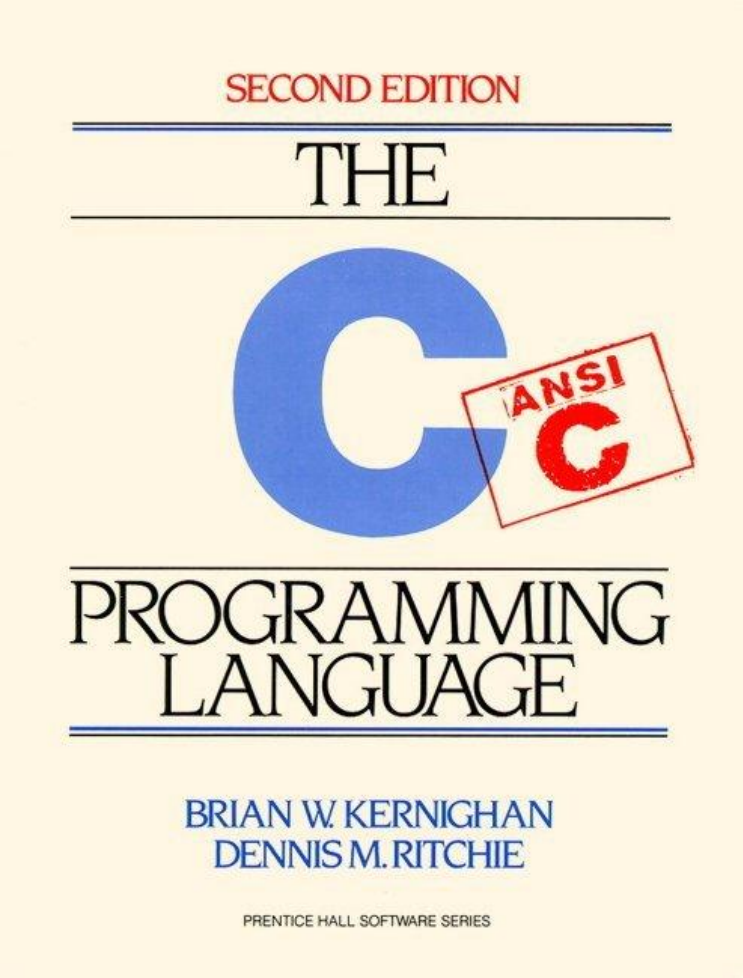
- Find the error in the following code

```
#include <stdio.h>
int main(){
 printf(Find the error!\n");
 return(0);
}
```

Get the Book

- Everybody should get a copy of:

C Programming Language (2nd Edition) [Paperback] by Brian W. Kernighan & Dennis M. Ritchie



Additional Resources

- http://www.cprogramming.com/
- This site has tutorials, example problems, and example quizzes.

Homework 1
Problem 1 - Solution

- Find the error in the following code

```
#include <stdio.h>
int main(){
 printf("Find the error!\n");
 return(0);
}
```

Homework 1
Problem 2

- Write a program to print your name on the screen.

Homework 1
Problem 2 - Solution

```
#include <stdio.h>


int main(){
    printf("My Name is John!\n");
    return 0;
}
```

Features

- C is *small* (originally only 32 keywords!)
- C is *common* (lots of C code about)
- C is *stable* (the language hasn't change much)
- C is *quick running* (close to assembly)
- C syntax is the *basis for many other languages* (csh, C++, awk, Perl).
- C is one of the easiest languages to learn.
  - Basic philosophy: Programmers know what they are doing

Comments and New Line: rules.c

```
/*
 * rules.c
 * this is a multi-line comment
 */

#include <stdio.h>
int main(){
  // this is a single line comment
  printf("Hello World!\n");
  printf("This code contains comments and prints to the screen\n");
return 0;
}
```

Some C Language Keywords

Category	Keywords
Storage class specifiers	`auto register static extern typedef`
Structure & union specifiers	`struct union`
Enumerations	`enum`
Type-Specifiers	`char double float int long short signed unsigned void`
Type-Qualifiers	`const volatile`
Control structures	`if, else, do, while, for, break, continue, switch, case, default, return, goto` ←NEVER USE
Operator	`sizeof`
Other reserved words	`asm bool friend inline`

Variables

- Information-storage places
- Compiler makes room for them in the computer's memory
- Can contain string, characters, numbers *etc.*
- Their values can change during program execution
- All variables must be declared before they are used and must have a data type associated with them
- Variable must be initialized before they are used

Types and variables

- A variable defines an area of storage in memory
- Must *declare* the *type* of every variable before we can use it.
- [type modifier] <type> <comma-separated names>;
- Motivation: makes intent of variables explicit and prevents mistakes, i.e. typos.
- Basic types: `int`, `char`, `long`, `short`, `float`, and `double`.
- Type modifiers: `signed`, `unsigned`, `long`, `short`, and `const`.
- Declarations of types should always be at the top of a syntax block or file.

Naming variables

- Variables in C can be given any name made from numbers, letters and underlines (or underscore)
- Caveat 1: Must not begin with a number.
- Caveat 2: Must not be a keyword
- A good name for your variables is important

```
/* City attributes for Houston */
   int number_buildings;
   double rain_fall_average;
   int population;
```

- Important to pick well chosen variable names and comments on variables

The `char` type

- `char` stores a character variable
  - usually 8 bits in size.
- A `char` can be assigned a value
  - between 0 and 255, or
  - a '*single quoted*' ASCII character.

```
char a, b;
   a= 123;
   b= 'a'; /* assigning a character*/
   return 0;
```

The int type

- `int` stores an integer variable
  - usually 32 bits in size.
- An `int` can be assigned a value
  - between `INT_MIN` and `INT_MAX` (defined in /usr/include/limits.h)

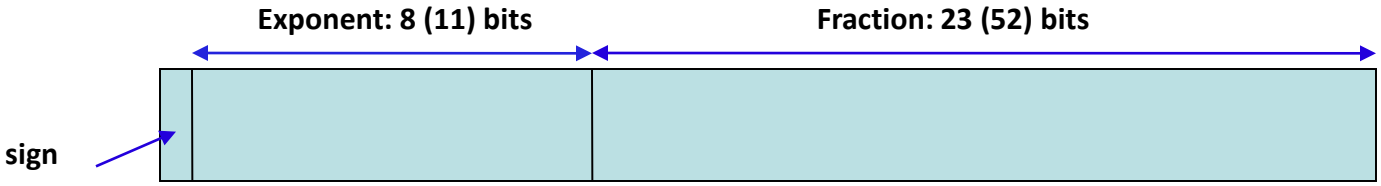
```
int x, y;
   x = 1966;
   #include <limits.h>
   y = INT_MAX;
```

The float and double type

`float and double` stores a floating point variable

- float – single-precision (32 bits)
- double – double-precision (64 bits)

- C uses the IEEE 754 float point representation standard



`MAXFLOAT` and `MINFLOAT` (`MAXDOUBLE` and `MINDOUBLE`) defined in /usr/include/values.h

Type modifiers: Signed/unsigned, long, short, const

`unsigned` can modify an `int` or `char` declaration. It means the variable can only be positive. `signed` means that it can be positive or negative.

`long` can modify an `int`, `float` or `double` declaration to increase its precision. `short` means they have less

`const` means a variable cannot be changed

```
short int small_no;
unsigned char uchar;
long double precise_number;
const double e= 2.718281828;
```

Hardware dependent: could be 80bit or 128bit

Example of Updating Variables: myAge.c

```
#include <stdio.h>
int main(){
   int age;
   age = 10;
   printf("Initial value of age is: %d\n", age);
   age = 20;
   printf("Updated value of age is: %d\n", age);
   age = age + 20;
   printf("New updated value of age is: %d\n", age);
   return 0;
}
```

Output:

```
Initial value of age is: 10
Updated value of age is: 20
New updated value of age is: 40
```

Casting between variables

- A cast is a way of telling one variable type to temporarily look like another.
- In many languages, this is not possible

```
int a;
double c;
c = a;
```

This is incorrect.

Casting between variables

- A cast is a way of telling one variable type to temporarily look like another.
- In many languages, this is not possible

```
int a;
double c;
c = a;

c= (double) a;
```

By using `(type)` in front of a variable we tell the variable to act like another type of variable. We can cast between any type.

Scope of Variables

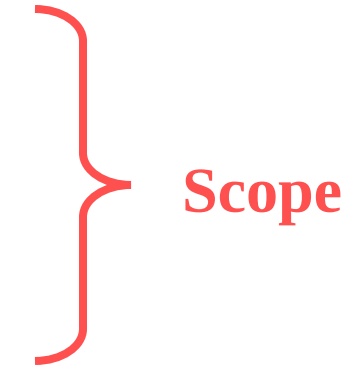
- A variable can be either of global or local scope
  - Global variables are defined outside all functions and they can be accessed and used by all functions in a program file
  - A local variable can be accessed only by the function in which it is created
- A local variable can be further qualified as **`static`**, in which case, it remains in existence rather than coming and going each time a function is called
  - **`static int x = 0;`**
- A **`register`** type of variable is placed in the machine registers for faster access – compilers can ignore this advice
  - **`register int x;`**

Variable Scope

What is scope?

- The scope of a variable is where it can be used in a program
- Normally variables are local in scope - this means they can only be used in the block where they are declared

```
{
   var1;
   {
      {
            func(var1);
      }
   }
}
```



- We can also declare global variables, but try to avoid this if possible.

Variable Scope

Local scope example

```
#include <stdio.h>
int main(){

    int i;
    i = 1966;
    {
      int i;
      i = 2007;
      printf("%d\n", i);
    }
    printf("%d\n", i);
    return 0;
}
```

Variables here are LOCAL variables

```
Output:
2007
1966
```

Variable Scope

Global scope example

```
#include <stdio.h>

int i;
int main()
{
    i = 1966;
    {
        i = 2007;
        printf("%d\n", i);
    }
    printf("%d\n", i);
    return 0;
}
```

Variable here is a GLOBAL variables

```
Output:
2007
2007
```

Constants

MAGIC numbers in programs

- A lot of programs contain MAGIC numbers

```
g = 43.2 * a + 7.1;
for (i= 7; i < 103; i+=2) {
    printf ("%d\n",i*7);
}
```

This makes code look ugly and difficult to maintain.
It is better to assign these numbers to a name
and localize it so changes can be made easily.
Three possible solutions: enum, #define and const.

Constants

enum

- We can use the **enum** syntax to define common int and char variables. E.g.:

```
enum {
     MAX_LEN= 100,
     LETTERX= 'x',
     NEWLINE= '\n'
};
```

By convention we use all capitals for constants.

```
for (i= 0; i < MAX_LEN; i++)
     printf ("%c",LETTERX);
```

Constants

Const

- Another solution is to use the **const** keyword.

```
/* Approximate value of PI */
const double PI=3.14;
/* Maximum iterations to be
   performed before exiting */
const int MAX_ITER=1000;
```

Constants

#define

- This pre-processor command replaces one thing with another before compilation

NOTE –
NO semicolon here!

```
#define PI 3.14
#define GRAV_CONST 9.807
#define HELLO_WORLD "Hello World!\n"
// Macros PI, GRAV_CONST, and HELLO_WORLD will
// be replaced with the defined values before
// the real compiler gets to work.
c= 2.0 * PI * r;
a= GRAV_CONST * (m1*m2)/ (r * r);
printf (HELLO_WORLD);
```

Constants and Constant Expressions

- The value of a constant never changes
  - `const double e = 2.71828182;`

- Macros
  - `#define MAXRECORDS 100`
  - In the code, identifiers (`MAXRECORDS`) are replaced with the values (`100`)
  - Helps to avoid hard-coding of values at multiple places
  - Example: `char records[MAXRECORDS + 1];`
  - Can be used at any place where constants can be used

- Enumeration is a list of constant values
  - `enum boolean {NO , YES};`

Expressions containing constants are evaluated at compile-time

Operators: increment/decrement

- ++i means increment i then use it
- i++ means use i then increment it

```
int i= 6;
int j;
j = i++;  /* j is assigned 6 */
```

```
int i= 6;
int j;
j = ++i;  /* j is assigned 7 */
```

All of the above also applies to --.

Basic useful operators

- Arithmetic: `+` (add), `-` (subtract), `*` (multiple), `/` (divide)

- Logical: `&&` (AND), `||` (OR), `!` (NOT)

```
(X && Y) || Z  /* TRUE if X and Y is TRUE OR Z is TRUE */
```

More useful operators

+= (add to a variable)

E.g. `a += 5 → a = a + 5;`

−= (subtract from variable)

E.g. `b -= 4 → b = b - 4;`

*= (multiply a variable)

E.g. `c *= 2 → c = c * 2;`

/= (divide a variable)

E.g. `f /= d → f = f / d;`

(x%y) returns remainder when x is divided by y

E.g. `remainder= x%y;`

Precedence and Order of Evaluation

Operators	Associativity
( ) [ ] -> .	Left to right
! ~ ++ -- + - * & (type)	Right to left
* / %	Left to right
+ -	Left to right
<< >>	Left to right
< <= > >=	Left to right
== !=	Left to right
&	Left to right
^	Left to right
|	Left to right
&&	Left to right
||	Left to right
?:	Right to left
= += -= *= /= %= &= ^= |= <<= >>=	Right to left
,	Left to right

Homework 2

- Write a program in C that declares 3 variables of type int, float, and char. You may choose any appropriate name for your variables. Use the following values for your variables:
  - int 40
  - float 3.14159265
  - char T
- After you have initialized these three variables, print them to the screen.