Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.087: Practical Programming in C

IAP 2010

Problem Set 1 – Solutions

Writing, compiling, and debugging programs. Preprocessor macros. C file structure. Variables. Functions and program statements. Returning from functions.

Out: Monday, January 11, 2010. **Due:** Tuesday, January 12, 2010.

Problem 1.1

(a) What do curly braces denote in C? Why does it make sense to use curly braces to surround the body of a function?

Answer: The curly braces denote a block of code, in which variables can be declared. Variables declared within the block are valid only until the end of the block, marked by the matching right curly brace `'}'`. The body of a function is one such type of block, and thus, curly braces are used to describe the extent of that block.

(b) Describe the difference between the literal values `7`, `"7"`, and `'7'`.

Answer: The first literal describes an integer of value 7. The second describes a null-terminated string consisting of the character `'7'`. The third describes the character `'7'`, whose value is its ASCII character code (55).

(c) Consider the statement
```
double ans = 10.0+2.0/3.0-2.0*2.0;
```

Rewrite this statement, inserting parentheses to ensure that `ans` = 11.0 upon evaluation of this statement.

Answer:
```
double ans = 10.0+2.0/((3.0-2.0)*2.0);
```

Problem 1.2

Consider the statement
```
double ans = 18.0/squared(2+1);
```

For each of the four versions of the function macro `squared()` below, write the corresponding value of `ans`.

1. `#define squared(x) x*x`

   Answer: $18.0/2 + 1 * 2 + 1 = 9 + 2 + 1 = 12$.

2. `#define squared(x) (x*x)`

   Answer: $18.0/(2 + 1 * 2 + 1) = 18/5 = 3.6$.

3. `#define squared(x) (x)*(x)`

   Answer: $18.0/(2+1)*(2+1) = 6*3 = 18$.

4. `#define squared(x) ((x)*(x))`

   Answer: $18.0/((2+1)*(2+1)) = 18/9 = 2$.

Problem 1.3

Write the "Hello, 6.087 students" program described in lecture (slides 22 and 27) in your favorite text editor and compile and execute it. Turn in a printout or screen shot showing

- the command used to compile your program
- the command used to execute your program (using `gdb`)
- the output of your program

Answer:

```
dweller@dwellerpc:~$ gcc -g -O0 -Wall hello.c -o hello.o
dweller@dwellerpc:~$ gdb hello.o
GNU gdb (GDB) 7.0-ubuntu
Copyright (C) 2009 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/dweller/hello.o...done.
(gdb) run
Starting program: /home/dweller/hello.o
hello, 6.087 students

Program exited normally.
(gdb) quit
dweller@dwellerpc:~$
```

Problem 1.4

The following lines of code, when arranged in the proper sequence, output the simple message "All your base are belong to us."

1. `return 0;`
2. `const char msg[] = MSG1;`
3. `}`
4. `#define MSG1 "All your base are belong to us!"`
5. `int main(void) {`
6. `#include <stdio.h>`
7. `puts(msg);`

Write out the proper arrangement (line numbers are sufficient) of this code.

Answer: 6, 4, 5, 2, 7, 1, 3. Note that line 4 also can go any place before line 2, and line 2 can also go before the function starts in line 5. Also, line 6 can go anywhere before the function starts.

Problem 1.5

For each of the following statements, explain why it is not correct, and fix it.

(a)
```
#include <stdio.h>;
```

Answer: Preprocessor macros like `#include` should not be terminated with semicolons. The correct code is:

```
#include <stdio.h>
```

(b)
```
int function(void arg1)
{
  return arg1-1;
}
```

Answer: The void type signifies an empty variable, which cannot be used in any way. The correct code is:

```
int function(int arg1)
{
  return arg1-1;
}
```

(c)
```
#define MESSAGE = "Happy new year!"
puts(MESSAGE);
```

Answer: The form of the `#define` statement does not include an assignment operator:

```
#define MESSAGE "Happy new year!"
puts(MESSAGE);
```

6.087 Practical Programming in C

January (IAP) 2010