C Programming

Recitation 1

March 2017

Hello

- Open hello.c file that contains following lines:

```
/* hello world */
#include <stdio.h>
int main() {
	printf("Knowledge is power\n");
	return 0;
}
```

- Compile with the following command:

```
gcc hello.c -Wall -ansi -pedantic-errors -o hello
# compiler: gcc
# file(s) to compile: hello.c
# name of the executable: -o hello
# compiler flags: -Wall -ansi -pedantic-errors
```

- Run as:

```
./hello
```

Details - Hello World

- Main function is the entry point
  - It has a type: *int*
  - It has no arguments *()*
  - Function body is between parentheses

```c
int main() {
}
```

- Main returns 0, an integer, since its type is *int*

```c
return 0;
```

- A statement to print a string to the standard output
  - String to be printed is between double quotes
  - Statements end with a semicolon *;*

```c
printf("Knowledge is power\n");
```

- We need to tell where to find the *printf* function
  - Include standard input/output library

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

Printf

- We will use *printf* function to output different types of data
- Open and examine *print.c*
- Try to compile and run as

```
gcc print.c -Wall -ansi -pedantic-errors -o print
./print
```

- There are a couple of errors with this code, let's fix the problems

Errors

```
print.c:11:5: error: expected ';' before 'printf'
```

- Python is the coolest but we are stuck with C this semester: semicolons everywhere

```
print.c:5:5: warning: implicit declaration of function 'printf' [-↩
    Wimplicit-function-declaration]
```

- We forgot to include stdio.h, it may work though since the compilers are smart

```
print.c:15:1: warning: control reaches end of non-void function [-↩
    Wreturn-type]
```

- Return an int, 0 is used for successful termination

```
print.c:13:5: warning: format '%d' expects argument of type 'int', ↩
    but argument 2 has type 'double'
```

- Warnings are not always innocent, see what happens when you try to print a floating point as an integer

Printf Format Specifiers

Type	Specifier	
char	%c	char
int	%d %i	decimal
	%u	unsigned decimal
	%o	octal
	%x	hexadecimal
	%l{d | i | u | o | x}	long
float double	%f	decimal points
	%e	exponent
	%L{f | e}	long

Solution

- Correct version should look like this:

```
#include <stdio.h>
int main() {
    printf("characters %c%c%c\n", 'a', '\t', '1');
    /* %d and %i behaves same for printf but differently for ↩
    scanf */
    printf("integers %d %i %i\n", 3, 04, 0xA);
    /* you can use type casting or arithmetic operations */
    printf("floating points %f %e\n", 2.5, (double)1000);
    printf("floating point precision %.2f\n", 3.1415);
    printf("floating point arithmetic %.10f\n", 1.0/7);
    printf("%f\n", 3.0);
    return 0;
}
```

- Compile and run as follows:

```
gcc print.c -Wall -ansi -pedantic-errors -o print
./print
```

Makefile

- Typing commands to compile can be difficult especially when you have several files
- We write those commands in *Makefiles* once and use them by calling the *make* command
- Makefiles contain commands in the format below:

```
target: dependencies
[tab] system command
```

Makefile

- Let's build a Makefile for the two source files we have
- Create a new file named *Makefile* and put the following content

```
hello:
	gcc -Wall -ansi -pedantic-errors hello.c -o hello

print:
	gcc -Wall -ansi -pedantic-errors print.c -o print

clean:
	rm -f hello print
```

- Do not forget to type tabs before commands
- We also have a clean command that removes previously build executables

Makefile

- Use the Makefile as follows:

```
$ make clean
rm -f hello print

$ make hello
gcc -Wall -ansi -pedantic-errors hello.c -o hello

$ make print
gcc -Wall -ansi -pedantic-errors print.c -o print
```

- If no target specified, make runs the first target (or you can specify the default)
- If the target files exist make command does not rebuild, use clean

Scanf

- We will use *scanf* function for inputs
- Open and examine *scan.c*
- Modify the makefile and try to compile and run as

```
make scan
./scan
```

- Let's fix the problems

Errors

```
scan.c:1:19: error: extra tokens at end of #include directive
```

- No need for semicolons after includes
- Try to compile without *-pedantic-errors* and see that it is now a *warning* instead of an *error*

```
scan.c:8:12: error: conflicting types for 'f'
scan.c:7:11: note: previous declaration of 'f' was here
```

- We cannot use the same name for different variables, nor we can change the type of the variables

```
scan.c:14:24: error: 'd' undeclared (first use in this function)
scan.c:14:24: note: each undeclared identifier is reported only once↩
          for each function it appears in
```

- Rename the double as d
- First round of problems are fixed, try to compile and run again

Errors

```
scan.c:14:5: warning: format '%f' expects argument of type 'float *',↩
    but argument 2 has type 'double'
scan.c:14:5: warning: format '%lf' expects argument of type 'double↩
    *', but argument 3 has type 'double'
```

- We need to tell *scanf* where to put the data (an address) it tries to read
- This is done using ampersand -&- before the variable name

```
scan.c:16:5: warning: format '%f' expects argument of type 'double',↩
    but argument 2 has type 'float *'
scan.c:16:5: warning: format '%f' expects argument of type 'double',↩
    but argument 3 has type 'double *'
```

- Using & in *printf* may go unnoticed, it may work without runtime errors but produces unwanted results

Errors

- Printf may work without errors because addresses are also numbers
- Scanf, on the other hand, almost always fails at runtime because the address provided randomly may not exist or does not belong to your program
- For example, type 'double *' is pointer to double, shows the address of a double in the memory
- We will cover the details of addresses and pointers in the following lectures
- Take the warnings seriously

Scanf Format Specifiers

Type	Specifier	
char	%c	char
int	%d	decimal
	%u	unsigned decimal
	%o	octal
	%x	hexadecimal
	%i	decimal / octal / hexadecimal
	%l{d | i | u | o | x}	long
float	%f	decimal points
	%e	exponent
double	%lf	decimal points
	%le	exponent
	%L{f | e}	long

Scanf

```
#include <stdio.h>
int main() {
    char c;
    int i;
    float f;
    double d;
    scanf("%c %d", &c, &i);
    printf("%c %d\n", c, i);
    scanf("%f %lf", &f, &d);
    printf("%f %f\n", f, d);
    return 0;
}
```

- Compile and run as follows:

```
make scan
./scan
```

More on Makefile

- Let’s use variables for flags so that we can change all commands easily from one location

```
# compiler
CC=gcc
# compiler flags
CFLAGS=-Wall -ansi -pedantic-errors
# link/load flags ex: "-lm" for math.h
LDFLAGS=

all: hello print scan types

hello:
	$(CC) $(CFLAGS) $(LDFLAGS) hello.c -o hello

print:
	$(CC) $(CFLAGS) $(LDFLAGS) print.c -o print

scan:
	$(CC) $(CFLAGS) $(LDFLAGS) scan.c -o scan

types:
	$(CC) $(CFLAGS) $(LDFLAGS) types.c -o types

clean:
	rm -f hello print scan types
```

- See that the target *all* has no commands, but its dependencies handle everything

Types

- We will use use various format options for printf and scanf and work with different types of numbers
- Open and examine *types.c*
- Try to compile and run as

```
make  types
./types
```

- More and more problems

Errors

```
types.c :24:5: warning: format '%f' expects argument of type 'float ↩
    *', but argument 4 has type 'double *'
```

- Scanf, unlike printf, expects different specifiers for float and double,
- Use %lf or %le for doubles

```
types.c :34:5: warning: ISO C90 does not support the '%lf' ↩
    gnu_printf format
```

- We don't use %lf to print doubles but just %f

```
types.c :34:5: warning: format '%lf' expects argument of type '↩
    double', but argument 2 has type 'long double'
```

- long double uses capital l in the printf format specifier, use %Lf or %Le

Types

- Now that we fixed the compile errors, try to compile and run as

```
make types
./types
```

- Observe that we cannot read the 2 characters we type
- What do you think the problem is?

Scanf

- The spaces we put between specifiers are very important
- They tell scanf function to skip all the white spaces between inputs
- Scanf matches all the characters with appropriate data types
- But spaces and newlines are also characters and when we try to read with %c they are not skipped
- The current version of the program reads the leftover newline from the previous scanf
- Notice the output, it contains ascii values of the characters. 10 is for the newline
- Add a space before the first %c, also try other alternatives and see how it behaves

```
scanf(" %c %c", &c1, &c2);
```

Moodle

`http://cengclass.ceng.metu.edu.tr`

- We will use Moodle for in lab and take home exams
- Please open the link above in your browser and login with your department accounts
- In the home page, select *C Programming*, then click *Enroll*
- Welcome to CENG140
- See the recitation files
- Click *Recitation 1*, we will use it for the next example
- This will show you the in lab exam environment

Task

- Let’s use what we have learned today and complete simple task in Moodle

Evaluation

- There are two ways to submit your codes
- The first one is to use the editor in Moddle
- The second one is to work on a file in your computer and upload it to Moodle
- If you choose to work locally, please upload your file frequently
- In case of a problem with your computer, you can start where you left of easily.
- If you evaluate on Moodle, you will see which tasks are passed and which ones are failed
- Alternatively, you can use evaluate.sh file if you are working on your local directories (you can also download your files from Moodle)
- It will present more details such as expected output and differences with your outputs