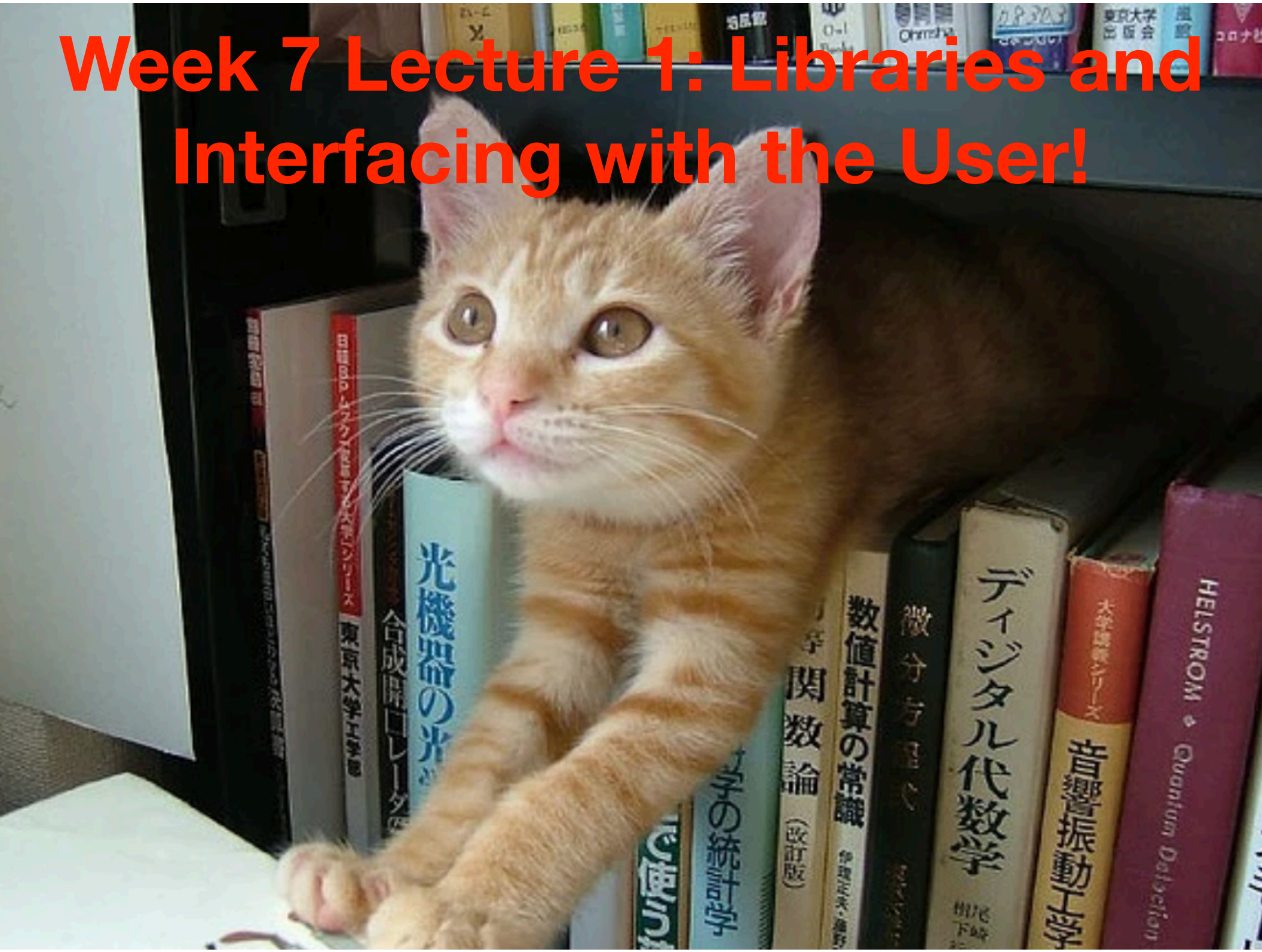
Week 7 Lecture 1: Libraries and Interfacing with the User!



Review and Preview

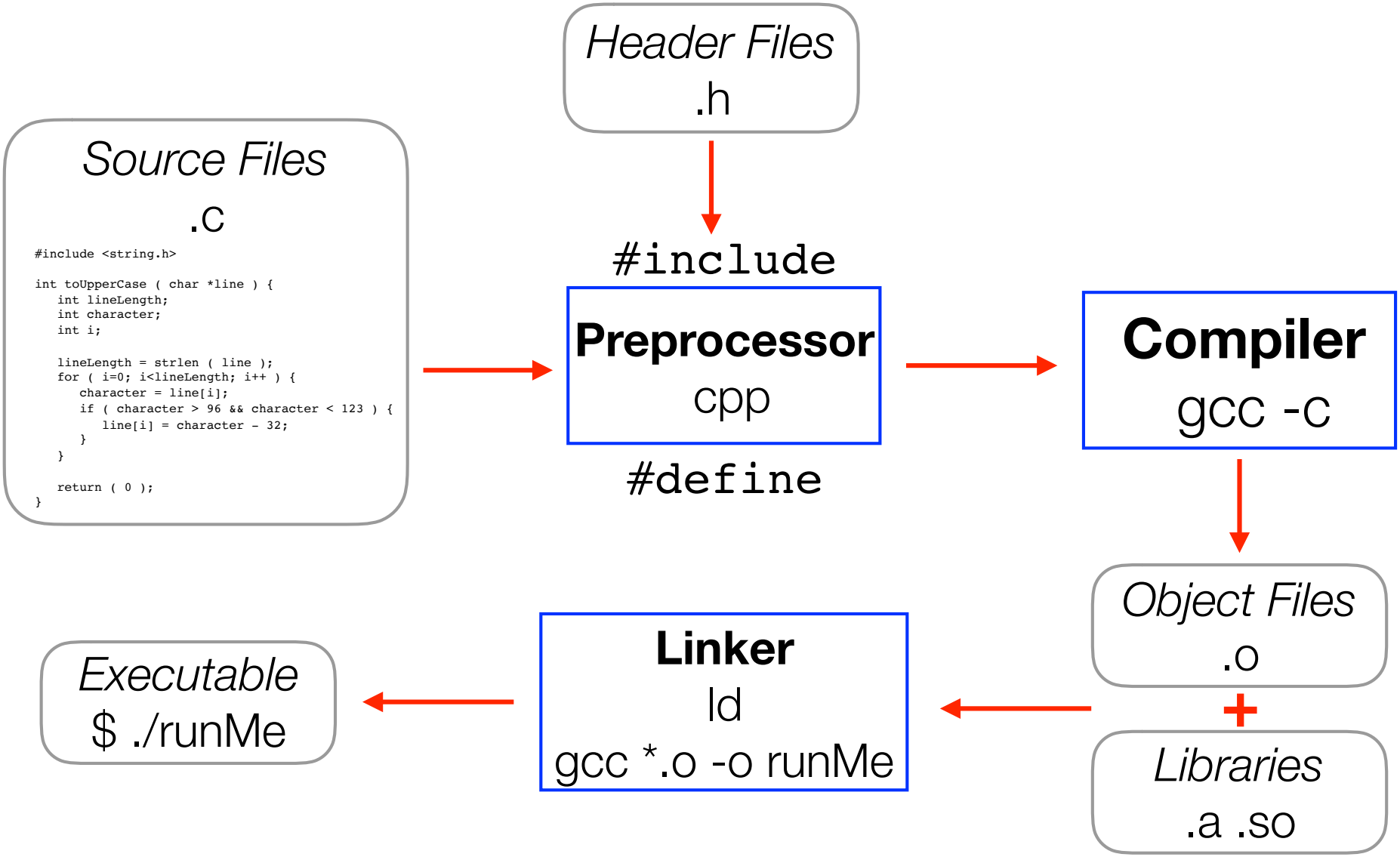
- Compiling and Linking
- Compiler Options
- make

- Libraries and Archive
- Interacting with the User (scanf, fgets)

K&R: Chapters 1-5
Sections 4.5, 4.11
Sections 5.1-5.5

Libraries and Archive

Where do all those *.o files go?



Compilation and Linking

The Linker

- The linker *links* together object files (`.o` files) into a binary executable.
- This includes
  - object files created from your source code files
  - object files that have been pre-compiled and collected into library files. These files have names ending in `.a` or `.so`.

Compiler Options

- Produce only the compiled code

```
$ gcc -c main.c
```

- Link with shared libraries

```
$ gcc -Wall main.c -o main -lm
```

Static Libraries

- The result of the linker making copies of all called library functions to the executable is called **static** linking.
- The `.a` files in Linux are static libraries.
  - These `.a` files are created using the `ar` (archive) command.
- The use of static libraries creates large executable files.

The Linker and Libraries

- The linker makes the code of library functions available to your program.
- It does this by:
  - copying the code of library functions to your object code (**static**)
  - making the library function code available at run-time (**dynamic**)

Creating a Static Library

1. Create all of your functions in separate `.c` files.

2. Create a header (`.h`) file containing all the declarations for your functions.

3. Compile all of the library functions.

4. Create a static library using the command `ar`.

5. Index the archive.

6. Link the static library to your mainline program.

ar - the archive program

- The program called **`ar`** (archive) creates a static library and can:
  - [**`c`** flag] create static libraries
  - [**`r`** flag] add (with replacement) object files in the static library
  - [**`t`** flag] list the names of object files in the library
- The library filename must start with **`lib`** and end with the **`.a`** extension

`ar rc libutil.a chop.o trim.o reduceSpace.o`

- Creates a static library named **`libutil.a`**
- Puts in copies of **`chop.o`**, **`trim.o`** and **`reduceSpace.o`**
  - object files added to it if they do not already exist
  - replaced if they are newer than those in the library
- The **`c`** flag tells **`ar`** to create the library if it does not exist
- The **`r`** flag tells it to replace older object files in the library with the new object files

Index the Static Library

- After a static library is created, or modified, it needs to be indexed.
- The index is used by the compiler to speed up symbol-lookup inside the library.
- It can be accomplished by
  - using the s flag in ar : `$ ar rcs libutil.a chop.o trim.o reduceSpace.o`
  - using the program ranlib : `$ ranlib libutil.a`

Link in the Library

- To link in libraries use the gcc command line options
  - `-L` for the path to the library files
  - `-l` to link in a library

```
-L{path to library file} -l{library name}
```

```
libutils.a is in /home/debs/CIS1300

$ gcc myprog.c -L/home/debs/CIS1300 -lutils
-o myprog
```

```
$ ar rc libutil.a chop.o convertLowerCase.o \
replaceDigits.o replacePunc.o reduceSpace.o \
trim.o

$ ranlib libutil.a

$ gcc -ansi -Wall findWords.c -o findWords \
-L. -lutil
```

```
$ ar rcs libutil.a chop.o convertLowerCase.o \
replaceDigits.o replacePunc.o reduceSpace.o \
trim.o

$ gcc -ansi -Wall findWords.c -o findWords \
-L. -lutil
```

Creating a Makefile using your Library

```
CC = gcc
UTILDIR=/home/debs/CIS1300
UTILFLAG=-lutil
UTILLIB=$(UTILDIR)libutil.a

main: main.c
  $(CC) $(CFLAGS) -o main main.c -L (UTILDIR) \
$(UTILFLAG)
```

Interacting with the User

Talking across the divide.

How Do You Ask the User for Input?

- So far we have only used command line parameters and STDIN to get input.
- But we can also “ask” the user to input data:
  - Part 1: Ask a question using `printf()`
  - Part 2: Capture the user’s input using either `scanf()` or `gets()`.

`scanf()`

- `scanf()` reads user input entered on the keyboard

```
scanf ( "%d", &num );
```

- The `scanf()` function uses the same placeholders as `printf()`:
  - `%d` : integer
  - `%f` : float
  - `%c` : character
  - `%s` : character string

`scanf()`

- `scanf()` reads characters from STDIN, interprets them according to a format, and stores the results in its arguments.
- As arguments there is
  - a control string format
  - a set of pointer arguments indicating where the converted input should be stored
- On success, it returns the number of variables filled.

```
#include <stdio.h>
int main ( int argc, char *argv[] ) {
   char nameFirst[30];
   char nameLast[30];
   int birthYear = 0;

   printf ( "Enter your first name: " );
   scanf ( "%s", nameFirst );

   printf ( "Enter your last name: " );
   scanf ( "%s", nameLast );

    printf ( "Enter the year you were born: " );
   scanf ( "%d", &birthYear );

   printf ( "Thank you %s %s (%d)\n", nameFirst,
nameLast, birthYear );
    return ( 0 );
}
```

```
#include <stdio.h>
int main ( int argc, char *argv[] ) {
   char nameFirst[30];
   char nameLast[30];
   int count = 0;

   printf ( "Enter your first name and then your
last name: " );
   count = scanf ( "%s %s", nameFirst, nameLast );

   printf ( "Thank you %s %s (%d)\n", nameFirst,
nameLast, count );

   return ( 0 );
}
```

A Problem with `scanf()`

```
#include <stdio.h>
int main ( int argc, char *argv[] ) {
    char c = 'a';
    printf("......Enter q to quit......\n");
    while ( c != 'q' ) {
        printf ( "Enter a character: " );
        scanf ( "%c", &c );
        printf ( "%c\n", c );
    }
    return ( 0 );
}
```

```
$ ./askMeProblem
......Enter q to quit......
Enter a character: a
a
Enter a character:

Enter a character: b
b
Enter a character:

Enter a character: c
c
Enter a character:

Enter a character: q
q
$
```

A Problem with `scanf()`

- Why are we getting extra lines (Enter a character:)?
- It is printing an extra “Enter a character” followed by an extra **new line**.
- Every `scanf()` leaves a **newline** character in the input (STDIN) buffer and this character is read by the next `scanf()`.
- How can you get around this problem?

A Problem with scanf() and an Alternative

- When you use scanf (like printf) not only are you reading (or writing) something, but you are interpreting it (the format part of the first function parameter).
- Put this powerful feature can also be used by a malevolent user to inject malicious code into your program.
- This is known as **buffer overflow**.

`fgets()`

- `fgets` stands for **get string from a file**.
- But on UNIX systems everything is treated as a file (sockets, streams, or actual files) including STDIN.
- **`fgets()`** is a robust alternative to **`scanf()`**, because it is simple to adjust **`fgets()`** to read from any type of file (including STDIN) so your program is very easy to adapt and modify.
- Crucially, with **`fgets()`** you can specify a specific buffer length which combats buffer overflow attacks.

`fgets()`

- The syntax is:

```
char *fgets(char *str, int n, FILE *fp);
```

- For the `FILE *fp` parameter we will use STDIN.
- The last character will be a '\n' or newline and so you must remove this unless you want the newline.
- The only disadvantage of `fgets` is that you do not get the formatting power of `scanf` but…there is a way around that.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int chop ( char * );
int main ( int argc, char *argv[] ) {
   char nameFirst[30];
   char nameLast[30];
   char string[10];
   int birthYear = 0;

   printf ( "Enter your first name: " );
   fgets ( nameFirst, 30, stdin );
   chop ( nameFirst );

   printf ( "Enter your last name: " );
   fgets ( nameLast, 30, stdin );
   chop ( nameLast );
```

```
    printf ( "Enter the year you were born: " );
    fgets ( string, 10, stdin );
    birthYear = atoi(string);

    printf ( "Thank you %s %s (%d)\n", nameFirst,
nameLast, birthYear );

    return ( 0 );

}
```

```
$ ./fgetsAskMe
Enter your first name: Willy
Enter your last name: Stacey
Enter the year you were born: 2009
Thank you Willy Stacey (2009)
$
```

```
char inputString[30];
char nameFirst[30];
char nameLast[30];
int birthYear = 0;

printf ( "Enter your first name: " );
fgets ( inputString, 30, stdin );
sscanf ( inputString, "%s", nameFirst );

printf ( "Enter your last name: " );
fgets ( inputString, 30, stdin );
sscanf ( inputString, "%s", nameLast );

printf ( "Enter the year you were born: " );
fgets ( inputString, 10, stdin );
sscanf ( inputString, "%d", &birthYear );

printf ( "Thank you %s %s (%d)\n", nameFirst,
nameLast, birthYear );
```