CSC111

Admin

- Assignment 4 released
  - Due in 10 days
- Exams -> handed back in labs this week
  - Mark appeals for the midterm exam must be done in person at office hours
- Reference sheets – please come to Friday’s office hours to pickup
- Reading break – Office hours won’t be happening
  - I will respond to forums for questions
  - Can email for on campus office hour apportionment.

Pointers

- Variable that stores an address of another variable
  - Must declare the pointer
    - type *var_name;
  - Two new operators
    - * - used to declare pointers and to dereference the pointer
    - & - used to get the address of a variable
  - Can have pointers to pointers
  - Allow for pass by reference

Example

```
int x;
x = 4;
int* iptr;
iptr = &x;

*iptr;

int z = *iptr;

*iptr = 20;
```

dereference operator

Can be read as:
"value pointed to by …"

the value pointed to by `iptr` is `x`

`z` gets the value 4

`x` gets the value 20

```
#include <stdio.h>

void add1( int* num_ptr);

int main ( ) {
        int x = 12;

        int* ptr = &x;
        add1(ptr);
        add1(&x);

        return 0;
}
/* Purpose: adds 1 to number pointed to by num_ptr
 * Parameters: int* num_ptr – address of an int
 */
void add1 ( int* num_ptr) {
        *num_ptr = *num_ptr + 1;
}
```

Function PROTOTYPE

Function CALLs
must pass **expected argument**

Documentation

Function DEFINITION

Pass by reference vs Pass by value

- When passing a value to a function as a parameter we are passing by value
- We can pass a pointers as a parameter – this passes the value stored at the address by reference

Condition Driven Loops

`scanf` documentation

Purpose: scans standard input entered through the keyboard
and stores it in the specified format to the specified address

Parameters: a string - containing format specifiers,
pointer types - addresses to store scanned values

Returns: int – the number of values scanned successfully

Example usage:

```
int i;
double d;
int scanned;
scanned = scanf("%d %lf", &i, &d);
// scanned is 2 if user enters 2 valid numbers
```

Examples usage of scanf function

```
int x, y, scanned;
scanned = scanf("%d %d", &x, &y);
printf("%d, %d, %d\n", x, y, scanned);
// if user enters: 5 9      output will be: 5, 9, 2
// if user enters: 5 ab     output will be: 5, -, 1
// if user enters: 5.1 6.5 output will be: 5, -, 1
// if user enters: bc 6     output will be: -, -, 0
// if user enters: ds ac    output will be: -, -, 0
```

NOTICE: as soon as invalid input is entered, scanf does not behave as expected

Critical pieces of the while loop

```
int n;

n=1;

while (n < 100) {

    printf("%d\n", n);

    n *= 2;

}
```

initialization of the variable to be used in the condition

the condition for which to continue to loop on

update the variable used in the condition such that the condition will eventually become FALSE

A sentinel…

- A special value that marks the end of a sequence of values
- When a program reads the sentinel value in a loop, it knows it has reached the end of the sequence, so the loop terminates.

Example:

- a user repeatedly enters positive numbers and enters -1 to stop
- The sentinel here is the -1

Output if -1 is entered when first scanf executes:

```
enter an int, -1 to stop
done
```

```
int val, scanned;
printf(“enter an int, -1 to stop\n”);
scanned = scanf("%d", &val);
while (scanned == 1 && val != -1){
     printf(“%d\n”, val);
     printf(“enter an int, -1 to stop\n”);
     scanned = scanf("%d", &val);
}
printf(“done”);
```

Output if 10 is entered when first line executes:

```
enter an int, -1 to stop
10
enter an int, -1 to stop
...
done
```

Why is this helpful?

- Generally, we have count driven loops
- Condition driven allows us to stop the loop after an unknown number of iterations
- When is this used:
  - User input
  - Roll dice until conditions are meet (assignment)
  - Waiting for message to arrive
  - ….

Consider the following program.

```
#include <stdio.h>
int main( void ) {
   double a,b;
   int ret_val;

   printf("enter two numbers separated by a space: ");
   ret_val = scanf("%lf %lf", &a, &b);

   printf("a: %.1f, b: %.1f, returned value: %d\n", a, b, ret_val);

   return 0;
}
```

- What is the output when the program is run with the following keyboard inputs entered? (indicate garbage value with –):
  - 5 7
  - 5.4 7.25
  - Abc 5.7
  - Abc efg
  - 3.8 82ef

Design a function that:

- Repeatedly prompts a user to enter integers one at a time
- If the user enters -1 or an invalid integer the loop terminates
- The function should count the number of odd positive values the user enters and return that count.

Design a function that:

- Repeatedly prompts a user to enter positive integers one at a time
- If the user enters -1 the loop terminates
  - If the user only enters -1 both the min and max should be set to -1
  - If the user only enters -1 An error message will be printed
- If the value is invalid, the user is prompted to input a number again
- The function store the minimum and maximum values entered in pointers passed to the function.