CSE / ENGR 142 Programming I

Loop Development

Goals

- Getting from problem statement to working code
- Systematic loop design and development
- Recognizing and reusing code patterns

Example: Rainfall Data

- General task: Read daily rainfall amounts and print some interesting information about them.
- Input data: Zero or more numbers giving daily rainfall followed by a negative number (sentinel).
- Example input data: 0.2 0.0 0.0 1.5 0.3 0.0 0.1 -1.0
- Empty input sequence: -1.0 [or -17.42 or …]
- What sort of information might we want to print?

Rainfall Analysis

Some possibilities:

- Print the data
- Print number of data values in the input
- Print maximum daily rainfall
- Print number of days with no rain
- Print average daily rainfall
- Print median daily rainfall (half of the days have more, half less)
- Print number of days where rainfall amount is above the average for all days in the input

What's similar about these? Different?

Example: Print Rainfall Data

```
#include <stdio.h>
int main (void) {
  double rain;          /* current rainfall from input */
  /* read rainfall amounts and print until sentinel */
  scanf("%lf", &rain);
  while (rain >= 0.0) {
    printf("%f  ", rain);
    scanf("%lf", &rain);
  }
  return 0;
}
```


Example: # Days in Input

```
#include <stdio.h>
int main (void) {
  double rain;          /* current rainfall from input */
  int ndays;            /* number of days of input */
  /* read rainfall amounts and count number of days */
  ndays = 0;
  scanf("%lf", &rain);
  while (rain >= 0.0) {
    ndays = ndays + 1;
    scanf("%lf", &rain);
  }
  printf("# of days input = %d.\n", ndays);
  return 0;
}
```

Is There a Pattern Here?

```
#include <stdio.h>
int main (void) {
  double rain;   /* current rainfall */

  /* read rainfall amounts */

  scanf("%lf", &rain);
  while (rain >= 0.0) {
    printf("%f  ", rain);
    scanf("%lf", &rain);
  }

  return 0;
}
```

```
#include <stdio.h>
int main (void) {
  double rain;          /* current rainfall */
  int ndays;            /* # input numbers */
  /* read rainfall amounts */
  ndays = 0;
  scanf("%lf", &rain);
  while (rain >= 0.0) {
    ndays = ndays + 1;
    scanf("%lf", &rain);
  }
  printf("# of days input = %d.\n", ndays);
  return 0;
}
```


Program Schema

- A program schema is a pattern of code that solves a general problem.
- Learn patterns through experience, observation.
- If you encounter a similar problem, reuse the pattern.
- Work the problem by hand to gain insight into possible solutions. Ask yourself "what am I doing?"
- Check your code by hand-tracing on simple test data.

Schema: Read until Sentinel

```
#include <stdio.h>
int main (void) {
  double variable;        /* current input */
  declarations;
  initial;
  scanf("%lf", &variable);
  while (variable is not sentinel) {
    process;
    scanf("%lf", &variable);
  }
  final;
  return 0;
}
```


Schema Placeholders

- In the schema, *variable*, *declarations*, *sentinel*, *initial*, *process*, and *final* are placeholders.
- *variable* holds the current data from input. It should be replaced with an appropriately named variable.
- *sentinel* is the value that signals end of input.
- *declarations* are any additional variables needed.
- *initial* is any statements needed to initialize variables before any processing is done.
- *process* is the "processing step" - work done for each input value.
- *final* is any necessary operations needed after all input has been processed.

Schema instance for Rainfall

```
#include <stdio.h>
int main (void) {
  double rain;          /* current rainfall */
  declarations;
  initial;
  scanf("%lf", &rain);
  while (rain >= 0.0) {
    process;
    scanf("%lf", &rain);
  }
  final;
  return 0;
}
```


Loop Development

Some useful ideas

- Do you know an appropriate schema? Use it!
- Declare variables as you discover you need them.
  - When you create a variable, write a comment describing what's in it!
- Often helps to start with
  - What has to be done to *process* one more input value?
  - What information is needed for *final*?
- Often easiest to write *initial* last
  - *initial* is "what's needed so the loop works the 1st time"
  - Often obvious after writing rest of the loop

Print Rainfall Data

```
                #include <stdio.h>
                int main (void) {
                    double rain;        /* current rainfall */
declarations:
    initial:
                    scanf("%lf", &rain);
                    while (rain >= 0.0) {
    process:
                        scanf("%lf", &rain);
                    }
        final:
                    return 0;
                }
```


Print # Days With No Rain

```
                #include <stdio.h>
                int main (void) {
                    double rain;        /* current rainfall */
declarations:
    initial:
                    scanf("%lf", &rain);
                    while (rain >= 0.0) {
    process:
                        scanf("%lf", &rain);
                    }
        final:
                    return 0;
                }
```


Print Largest Daily Rainfall

```
                #include <stdio.h>
                int main (void) {
                    double rain;        /* current rainfall */
declarations:
    initial:
                    scanf("%lf", &rain);
                    while (rain >= 0.0) {
    process:
                        scanf("%lf", &rain);
                    }
        final:
                    return 0;
                }
```


Print Average Daily Rainfall

```
                #include <stdio.h>
                int main (void) {
                    double rain;        /* current rainfall */
declarations:
    initial:
                    scanf("%lf", &rain);
                    while (rain >= 0.0) {
    process:
                        scanf("%lf", &rain);
                    }
        final:
                    return 0;
                }
```


Print Average Daily Rainfall (2)

```
                #include <stdio.h>
                int main (void) {
                    double rain;        /* current rainfall */
declarations:
    initial:
                    scanf("%lf", &rain);
                    while (rain >= 0.0) {
    process:
                        scanf("%lf", &rain);
                    }
        final:
                    return 0;
                }
```


Event-Driven Programming

- Modern programs tend to be "event-driven"
  - Program starts, sets itself up
  - Wait for an event to happen
    - event = mouse click, key press, timer, menu selection, etc.
  - Perform requested operation ("handle" event)
  - Resume waiting for the next event
- The GP142 graphics package we'll use follows this model
- Can also be used with text (console) input

Simple Command Interpreter

Read in "commands" and execute them.

Input - single characters

- a -- execute command A by calling *process_A( )*
- b -- execute command B by calling *process_B( )*
- q -- quit

Pseudocode for main loop:

- get next command
- if a, execute command A
- if b, execute command B
- if q, signal quit

Command Interpreter Loop Control

repeat until quit signal

use variable "done" to indicate when done

```
set done to false
while not done
    body statements
    if quit command, set done to true
```


Command InterpreterProgram

```
#define FALSE 0
#define TRUE  1
int main(void)
{
   char command;     /* current input command */
   int done;         /* == "quit command has been entered" */
   done = FALSE;
   while (! done){
      /* get command from user */
      printf("Input command: ");
      scanf(" %c", &command);
      if (command == 'a' || command == 'A'){
         process_A( );        /* Execute command A */
      } else if (command == 'b' || command == 'B') {
         process_B( );        /* Execute command B */
      } else if (command == 'q' || command == 'Q') {
         done = TRUE;         /* User wants to quit */
      } else }
         printf("Unrecognized command\n");
      }
   }
   return 0;
}
```