Outline

- C preprocessor directives
- C Header Files
- C Typecasting
- C Decision Making

We have some experience with some preprocessor directives. I

- We have used #include and #define.
- Courtesy of the IDE we have used #ifndef and #endif.
- There is a companion to #define, which is #undef.
- There is a relative of #ifndef, which is #ifdef.
- There are #if, #elif, #else.
- Beyond these there are #line, #error and #pragma.

We are becoming familiar with #include and #define.

```
#include "production.h"
```

This include directive, used in a .c file, informs the compiler (which reads the file production.h) about functions that are invoked, and user-defined datatypes that are used, in the file in which the include directive is found.

```
#define FILENAMELENGTHALLOWANCE 50
```

- This define directive informs the compiler that when the all caps name is encountered, the second argument is to be substituted in.
- We use these to avoid "magic numbers" in our code.
- There are other substitutions, we tend to avoid them, as they are error prone.

We have seen #infndef and #endif. I

```
#ifndef PRODUCTION_H_
#define PRODUCTION_H_
    #include <stdio.h>
    #include <stdbool.h>

    #define FILENAMELENGTHALLOWANCE 50
    bool production(int argc, char* argv[]);

#endif /* PRODUCTION_H_ */
```

- We might instruct the compiler to read an include file multiple times,
- such as, if file a includes file b, and both file a and file b include file c.

We have seen #infndef and #endif. II

- The preprocessor would create a file with duplicate information.
- To cut down the size due to this duplication,
- #define X is used to mark the presence of an include file, and
- #ifndef X is used to check for this mark of the presence of the include file.
- By the use of #ifndef X
- we instruct the preprocessor to read the enclosed portion of the file, only if
- the literal X is not already defined.
- We do enclose the region of code guarded by the #ifndef (similarly #ifdef),
- so that we can have the idea of region.
- We delimit the region at its end with #endif.

Imagine writing code that customers with various platforms might buy.

- Suppose (It can happen.) that one end-of-line notation exists for Windows, and
- a different end-of-line notation exists in Mac.
- You might want to write code that can be instructed on building, whether it is being built for a Windows distribution or a Mac distribution.
- In this instance you might choose to include different parts of source code.

```
    char EndOfLine[3] ="\0\0\0";
#if  WINDOWS
    strcpy(EndOfLine, "\012\013";
#elif LINUX
    strcpy(EndOfLine, "\012");
#else
    strcpy(EndOfLine, "\013");
#endif
```

We mention pragma to suggest you do not use it.

- Directives of type pragma are for compiler specific extensions.
- This means when you use them you are making your code dependent upon a specific compiler.

```
#pragma warning (disable : 4018 )   //NOT recommended
```

- This example pragma is inadvisable, because warnings help us fix our code.

Header files are useful. I

- While one could put all header information into a single file,
- it is better for code maintenance, and our understanding as we develop,
- to use multiple header files.
- One useful way to divide up the header information among files is to have a header for every group of functions.
- In the homework, we made groups of functions around concepts including House, Layout, Room.
- We created a .c file and a .h file for each group.
- Using the message arrows from the sequence diagram,
- which are composed of the first, or invocation, arrow, and
- possibly a second, or return, arrow,
- identify the tail of the first arrow.
- This tells us in which .c file the invocation of the function from the message occurs.

Header files are useful. II

- Then use the head of the first arrow to discover where the implementation of the function from the message occurs
- The .c file containing the invocation should #include the .h file containing the prototype of the function from the message.
- The .c file containing the implementation should #include the .h file containing the prototype of the function from the message.
- Using the #include directive helps the compiler help you develop the code.

Typecasting is sometimes necessary, and often useful.

- We will want to obtain memory.
- We will use malloc, which returns a void pointer.
- We will always cast the result of malloc to the kind of pointer we want.
- As in Java, typecasting looks like (target type).
- When the target type is a pointer, we will not be surprised to see an asterisk: (target*).

```
Location* lP = (Location*) malloc (12 * sizeof(Location));
//room for a dozen locations
```

Decision making occurs in code.

- There are several statements in which decision making is expressed.
- These are:
  - if
  - if-else
  - nested if, if else
  - switch/case
  - while
  - for

Decision making with if. I

- Decision making implies having an expression to evaluate.
- In "if" statements, the expression is called a conditional.
- The conditional in an if statement answers a question of type boolean, that is, either true or false.
- The condition expression is enclosed in parentheses.
- In case the conditional evaluates to true, the statement block guarded by the if is executed.

```
if((x%2)==0)
{
     puts("x is even");
}
```

Decision making with if/else. I

- Decision making implies having an expression to evaluate.
- In "if" statements, the expression is called a conditional.
- The conditional in an if statement answers a question of type boolean, that is, either true or false.
- The condition expression is enclosed in parentheses.
- In case the conditional evaluates to true, the statement block guarded by the if is executed,
- otherwise the statement block guarded by the else is executed.

```
if((x%2)==0)
{
    puts("x is even");
}
else
{
    puts("x is odd");
}
```

Decision making with nested if. I

- The outer if of the nested pair is executed as described just previously.
- The inner if is found inside the statement block guarded by the if.
- The inner if is only evaluated if the statement block in which it is found is executed.
- When the conditional of the inner “if” evaluates to true,
- the statement block guarded by the inner “if” is executed.
- Nesting is recursive, that is, a statement block may contain an if, which in turn guards a statement block.

Decision making with nested if. II

```
if((x%2)==0)
{
   if((y%3)==0)
   {
        puts ("y is divisible by 3.");
   }
    puts("x is even");
}
```

Decision making with switch/case. I

- A variable's value is used to determine a case.
- The order of the cases, and the placement of break statements determine which cases are executed in response to the variable's value.

Decision making with switch/case. II

```
81      int result = 0;
82      int answer = 0;
83      switch (day)
84      {
85      case Monday:
86          result = 1;
87      case Tuesday:
88          answer = 2;
89          break;
90      case Wednesday:
91          result = 3;
92          break;
93      case Thursday:
94          answer = 4;
95      case Friday:
96          result = 5;
97          break;
98      case Saturday:
99          answer = 6;
100         break;
101     case Sunday:
102         result = 7;
103     default:
104         puts("Unexpected value for day");
105     }
```

There are decisions in iteration constructs, such as for loops. I

- Loops are iteration constructs.
- The word loop refers to the flow of control returning to a former location, and repeating use of instructions.
- The amount of repetition is controlled.
- The decision is about whether to repeat or instead, obtain instructions from after the repeated block of statements.
- For loops are one kind of control of iteration.
- For loops have places for:
  - initialization of variables
  - condition about repeating
  - steps to take in between repetitions, called update or increment
- each is separated by a semicolon.
- We have seen examples of initialization to 0 and to 1.

There are decisions in iteration constructs, such as for loops. II

- It is also possible to initialize more than one variable, separated by commas.
- We have seen examples of termination at a fixed number, or at a variable.
- Termination conditions can be the result of logical combinations.
- We have seen the update, or increment part of the for loop control, as increment and there are other possibilities.

```
for(int i = 0; i<3; i++)
{
    printf("Be brave.\n");
}
```

There are decisions in iteration constructs, such as for loops. III

- In the example above, we initialize one variable, we check the count of iterations, we update by adding 1.

```
for(int i = 0, int j = 5; i<3; i++)
{
printf("Be brave.\n");
}
```

- In the example above, we initialize two variables.

```
for(int i = 0, int j = 5; (i<3)&&(j<10); i++)
{
printf("Be brave.\n");
}
```

There are decisions in iteration constructs, such as for loops. IV

- In the example above, we initialize two variables, and the termination condition is a logical combination, AND, of two conditions.

```
for(int i = 0, int j = 5; j<0; i++, j=j-3)
{
printf("Be brave.\n");
}
```

- In the example above, we initialize two variables, and the update portion decrements a variable by 3.

There are decisions in iteration constructs, such as while loops. I

- In the while loop control, there is an expression of type bool: evaluating to true or false.

```
bool done = false;
while(!done)
{
    //do something that results in we are done
    done = true;
}
```

- We determine during run time when we are finished with the while loop.
- It is also possible to use a for loop for the same effect.
- The keyword while allows us to express this more directly.

There are decisions in iteration constructs, such as while loops. II

```
bool escape = true;
bool tooManyDays = false;
int manyDays = 1000000;
while(escape && !tooManyDays)
{
   //live another day
   manyDays--;
   if (manyDays< 0)
   {
      tooManyDays = true;
   }
   escape = tryAnEscapade();
}
```

- We can put a logical expression in the while condition.

We can make an analogy between programming and some games.

- There are games where it is easy to learn the legal moves, but
- difficult to become proficient at strategy and tactics.
- Improvement at strategy and tactics can be learned in several ways.
- One way is practice, and another is learning from predecessors,
- including predecessor's comments on previous examples (previous chess games, etc.)

Learning from doing is valuable, but can be slow.

- Lab work
- homework
- problems you set for yourself

Learning from predecessors is possible from books and the web.

- Many types of programs have similar components.
- These similarities have been distilled into Patterns.
- See for example Design Patterns: Elements of Reusable Object-Oriented Software, by Erich Gamma, Ralph Johnson, Richard Helm, and John Vlissides.

Let's apply a commonly used pattern, the Model-View-Controller pattern. I

- In the model-view-controller pattern (MVC), there are three main components.
- The model is a collection of state data, which could easily be held in a database,.
- The view is experienced by the human user.
- The view contains elements of the human computer interface.
- There are visual elements through which input is entered into the program.
- The keyboard any other input modalities are also included.
- There are visual elements which display to the user, as well as auditory, etc.
- The last component is the controller.
- We can appreciate the controller by reflecting on the variety of redundant input methods.

Let's apply a commonly used pattern, the Model-View-Controller pattern. II

- For menu items, there are often keyboard accelerators, such as <control> z for Undo.
- Drop down context menus may contain redundant commands to icons.
- The view provides these many options.
- The controller receives the input data, and "canonicalizes" it, that is,
- converts each of the redundant entry expressions into a single expression.
- That is, a keyboard accelerator, a menu item from a drop down menu and an icon click could all mean the same thing. The controller takes any of these redundant inputs, and converts each into a single form.
- That single form is recognized by the model, and is used to update the system state.