Door-to-door Work and CSE

What should the code for delivering a paper to these five houses look like?

One Solution… Many Times

Go to house #1
Turn up the path
Walk to the door
Put down the paper
Walk back to the road
Go to house #2
Turn up the path
Walk to the door
…

Breaking it Down

Go to house #1
Deliver a paper to #1
Go to house #2
Deliver a paper to #2
Go to house #3
Deliver a paper to #3
…

Delivering a paper to a house:
    Turn up the house's path
    Walk to the door
    Put down a paper
    Walk back to the road

CSE 142
Computer Programming I

Functions I

© 2000 UW CSE

Overview

Concepts this lecture
  Abstraction for behavior:
    "procedural abstraction"
  Functions
  Function control flow
  Two meanings of void
  Pre-written functions

Chapter 3

Read All!
    3.1: Reusing program parts
    3.2: Built-in math functions
    3.3: Top-Down Design
    3.4: Functions with no parameters
    3.5: Functions with parameters

Deceptively Simple Big Idea: Abstraction

One idea ⇒ One definition, many uses
One idea ⇒ One definition, many uses
One idea ⇒ One definition, many uses
One idea ⇒ One definition, many uses
One idea ⇒ One definition, many uses
One idea ⇒ One definition, many uses
One idea ⇒ One definition, many uses
One idea ⇒ One definition, many uses

Big Idea for Concrete Data: Symbolic Constants

One idea ⇒ One definition, many uses

Not this! →
```
if (myMoney > 80.0) {
    myShoes = myShoes + 1;
    myMoney = myMoney – 80.0;
}
```

One idea: cost of shoes
```
int COST_OF_SHOES = 80.0;
..
if (myMoney > COST_OF_SHOES ) {
    myShoes = myShoes + 1;
    myMoney = myMoney – COST_OF_SHOES;
}
```
One Name
Many Uses
One Definition

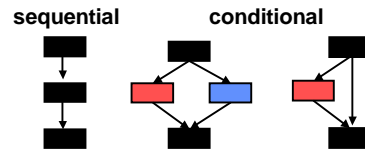## What sorts of *behavior* might we want to give names to?

*We **can't** do this with the tools we've seen so far.*

Control Flow: Review

"The order in which statements are executed."

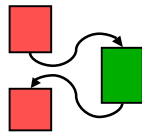**We've discussed two forms of control flow: sequential and conditional**

sequential   **conditional**
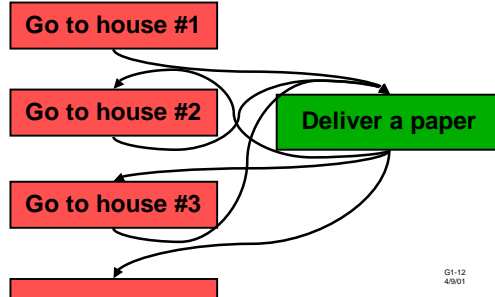

Another Form of Control Flow

"Functions" (or "procedures" or "subroutines") allow you to "visit" a chunk of code and then come back

(The function may be elsewhere in your own program, or may be code in another file altogether.)

Why this isn't just sequential…

Go to house #1

Go to house #2   **Deliver a paper**

Go to house #3

Why Use Functions?

Here's one example:

Suppose we are writing a program that displays many messages on the screen, and…

We'd like to display two rows of asterisks ('*'s) to separate sections of output:

```
********************
********************
```

Moving Toward a Solution

The result we want is this:

```
********************
********************
```

And the basic code needed is this:

```
printf("********************\n");
printf("********************\n");
```

A Full Solution

```c
#include <stdio.h>
int main(void)
{
   /* produce some output */
   ...
   /* print banner lines */
   printf("********************\n");
   printf("********************\n");

   /* produce more output */
   ...
   /* print banner lines */
   printf("********************\n");
   printf("********************\n");

   /* produce even more output */
   ...
   /* print banner lines */
   printf("********************\n");
   printf("********************\n");

   /* produce final output */
   ...
   return 0 ;
}
```

Anything Wrong With This?

It's correct C code

It fulfills the problem specification, i.e., gives the desired result

---

Anything Wrong With This?

It's correct C code

It fulfills the problem specification, i.e., gives the desired result

What's "wrong" has to do with other issues such as:

- How hard it would be change the program in the future
- How much work is it to write the same statements over and over
- ...

What if...

Later on we wants to change...

- The number of rows of asterisks
- The number of asterisks per row
- Use hyphens instead of asterisks
- Print the date and time with each separator
- ...

How much work is involved?

If We Want to Change Anything

… have to edit every "copy" of the code in the program.

… it's easy to overlook some copies.

… it can be hard to find them all (because they might not be written identically).

… it can be hard to find them all because code written identically may not serve the same logical purpose.

Sound familiar?

These are the *same* problems that lead us to use symbolic constants for data!

Functions let us do the same sort of thing for behavior ("procedural information")!

Big Idea for Code: Functions

One idea ⇒ One definition, many uses

One idea → • Identify a "sub-problem" that has to be solved in your program

One Name → • Choose a name to represent "*the solution of that problem by code*"

One definition → • Write that solution code (only once)

Many Uses → • Whenever you see that same sub-problem again, use the function name to say

"*go to that code now to take care of this problem, and don't come back until you're done*"

PrintBannerLines Function

For our print banner program, that idea means this:

• Identify the idea
print a banner
(NOT *print two rows of asterisks*)

• Give the function that does that a name
PrintBannerLines

• Define the solution by writing the code
printf("********************\n");
printf("********************\n");

• Whenever you want to print a banner, use the function name
PrintBannerLines();

```
#include <stdio.h>
int main(void)
{
    /* produce some output */

    PrintBannerLines();

    /* produce more output */

    PrintBannerLines();

    /* produce more output */

    PrintBannerLines();

    /* produce final output */

    return 0 ;
}
```

The code named PrintBannerLines

```
printf("********************\n");
printf("********************\n");
```

Discussion Question

In the new version of the program:

What do we have to do now if we want to change the banner?

How many places in the program have to be changed?

What if we want to print two rows of asterisks for something that isn't a banner?

The Big Picture, So Far

You've now some colossal concepts:

Abstraction

Functions

Function control flow

The motivation for functions

Coming right up...

Syntax for defining a function

Built-in C functions

Syntax for PrintBannerLines

```c
/* write separator line on output */
void PrintBannerLines (void)
{
    printf("***************\n");
    printf("***************\n");
}
```

Two Key Features

1. The name of the function and
2. the function body: code that is to be executed when the function is called.

function name

```c
/* write separator line on output*/
void PrintBannerLines (void)
{
    printf("***************\n");
    printf("***************\n");
}
```

heading comment

function body (statements to be executed).

A function can have ANY number of ANY kind of statements.

Further details: *void*

The keyword **void** has two different roles in this function definition.

indicates that the function does not return a value.

```c
/* write separator line on output*/
void PrintBannerLines (void)
{
    printf("***************\n");
    printf("***************\n");
}
```

indicates that the function has no parameters.

Oops – Two New Concepts

1. **Return values**: we will postpone for now
2. **Parameters**: We will postpone this, too!

Both concepts are very important in general, but not for this particular example

```c
/* write separator line on output*/
void PrintBannerLines (void)
...
```

Using *PrintBannerLines*

```c
#include <stdio.h>
void PrintBannerLines (void)
{
    printf("***************\n");
    printf("***************\n");
}
int main (void)
{
    /* produce some output */
    ...
    PrintBannerLines( );
    ...
    return 0;
}
```

The definition of the function must precede all calls to it in the file.

Empty ( ) is required when a parameter-less (void) function is called.

Some C Functions

We have already seen and used several functions:

int main (void) ← **Function definition for main( )**
{
    return 0;
}

printf ("control", list);
Calls to the functions printf( ) and scanf( )
scanf ("control", &list);

Library functions

- Pre-written functions are commonly packaged in "libraries"
- Every standard C compiler comes with a set of standard libraries
- Remember #include <stdio.h> ?
  - Tells the compiler you intend to use the "standard I/O library" functions
  - printf and scanf are in the standard I/O library
  - So are lots of other I/O related functions
- There are (many) other useful functions in other libraries

Next Time

We'll continue our discussion about functions. We will examine how values are passed to functions, and how values are returned

QOTD: The Better, The Worse, and The Uglier

Remember "the Good, the Bad, and the Ugly"? Well, it's just as important to use good identifiers for functions.

So, for each of the following situations, give a name that's legal and follows The Way, illegal, and legal but strays:

- Our banner lines function
- A function that retrieves the user's meal preferences (e.g., vegetarian, kosher, etc.)
- A function that calculates the least common denominator of two numbers

What might the rules be for function identifiers?