Program specifications:
Display many messages on the screen
Display a two rows of asterisks to separate sections of the output

PRINTING BANNERS
One version:
```
cout << "*****************************" << endl;
cout << "*****************************" << endl;
```

Another version:
```
int count;
for (count = 0; count < 30; count++) {
    cout << '*' << endl;
}
cout << endl;
for (count = 0; count < 30; count++) {
    cout << '*' << endl;
}
cout << endl;
```

#include <iostream>
int main() {
    // produce some output
    // print banner lines
    cout << '=================================\n';
    cout << '=================================\n';
    // produce more output
    // print banner lines
    cout << '=================================\n';
    cout << '=================================\n';
    // produce even more output
    // print banner lines
    cout << '=================================\n';
    cout << '=================================\n';
    // produce final output
    ... return 0;
}
WHAT IF WE WANT TO CHANGE THE BANNERS?

Number of rows, number of asterisks per row, use hyphens instead of asterisks, print the date with each row, …

Have to edit every "copy" of the code in the program
Easy to overlook some copies
Hard to find them all

Starting IM program...
Starting IM conversation...
Number of participants = 2
Start typing at any time.

USE A FUNCTION!

Starting IM program...
Starting IM conversation...
Number of participants = 2
Start typing at any time.

#define <iostream>
int main() {
    // produce some output
    PrintBannerLines();
    // produce more output
    PrintBannerLines();
    // produce even more output
    PrintBannerLines();
    // produce final output
    return 0;
}

What do we do now to change the banner?
How many places in the code have to be changed?
What if we want to print two rows of asterisks for something that isn't a banner?

RECALL OTHER FORMS OF CONTROL FLOW

Sequential |
--- |
Conditional |
Loop |

if, else |
while, for |

ANOTHER FORM OF CONTROL FLOW

Control Flow: the order in which statements are executed

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

BIG IDEA FOR CODE: FUNCTIONS

1. Identify the goal: “Print a banner.”
(More abstract than “Print two rows of asterisks.”)

2. Give the function that does that a name: PrintBannerLines

3. Define the solution by writing the code

4. Whenever you want to print a banner, use the function name

BIG IDEA FOR CODE: FUNCTIONS

1. Identify the goal: “Print a banner.”
(More abstract than “Print two rows of asterisks.”)

2. Give the function that does that a name: PrintBannerLines

3. Define the solution by writing the code

4. Whenever you want to print a banner, use the function name
ABSTRACTION

"one name, one definition, many uses"

FUNCTIONS

A FAMILIAR C++ FUNCTION

Function definition for main()

```cpp
int main()
{
    ...
    return 0 ;
}
```

PARAMETERS AND RETURN VALUES

PrintBannerLines

Parameters

(Output)

Return value

PARAMETERS AND RETURN VALUES

The function does not return a value

The function has no parameters

```cpp
// Function to print banner lines
void PrintBannerLines()
{
    cout << "*********************\n";
    cout << "*********************\n";
}
```

PARAMETERS

Suppose we want to change the program:

It should now print 5 rows when it starts and when it finishes, but print the original 2-row banner everywhere else

We could write an additional function that prints 5 rows of asterisks, or…
CAN WE GENERALIZE?

Modify the function so that it will print N rows of asterisks

N is the number of rows that we want “this time” when we call it

N is information that is required to write the code of the function

PASSING ARGUMENTS

```cpp
#include <iostream>

int main() {
    // produce some output
    PrintBannerLines(5);
    // produce more output
    PrintBannerLines(2);
    // produce even more output
    PrintBannerLines(5);
    // produce final output
    return 0;
}
```

---

MATCHING UP THE ARGUMENTS

The function call must include a matching argument for each parameter

When the function is executed, the value of the argument becomes the initial value of the parameter

```cpp
int main (void) {
    // produce some output
    PrintBannerLines(5);
    // produce more output
    PrintBannerLines(2);
    // produce even more output
    PrintBannerLines(5);
    // produce final output
    return 0;
}
```

---

MULTIPLE PARAMETERS

A function may have more than one parameter

Arguments must match parameters in number, order, and type

```cpp
float Avg (float total, int count) {
    return total / (float) count;
}
```
#include <iostream>
int main()
{
    // produce some output
...
    PrintBannerLines(5,'&',21);
    // produce more output
...
    PrintBannerLines(2,'@',15);
    // produce even more output
...
    PrintBannerLines(5,'1',1);
    // produce final output
...
    return 0;
}

void PrintBannerLines(int numLines, char mychar, int numChars)
{
    int i, j;
    for (i = 0; i < numLines; i++)
    {
        for (j = 0; j < numChars; j++)
        {
            cout << mychar;
        }
        cout << endl;
    }
}

MORE ABOUT CALLING FUNCTIONS

The definition is an implicit declaration
Another way is to declare a function prototype:
Declares number and types of parameters, and type of return value

#include <iostream>

#include <cmath>
...

LIBRARY FUNCTIONS

Pre-written functions are commonly packaged in libraries
Every C++ compiler comes with a set of standard libraries

#include <iostream>
#include <string>
...
#include <cmath>
...

SUMMARY: FUNCTIONS

Functions may take several parameters, or none
Functions may return one value, or none

Functions are valuable!
    A tool for program structuring
    Provide abstract services: The caller cares what the function does,
    but not how it does it
    Make programs easier to write, debug, and understand

SYMBOLIC CONSTANTS
BIG IDEA FOR DATA: SYMBOLIC CONSTANTS

One way to compute area:

```c
float radius = 0;
 cin >> radius;
 float area = 3.14159 * radius * radius;
```

An alternative:

```c
const float PI = 3.14159;
 float radius = 0;
 cin >> radius;
 float area = PI * radius * radius;
```

SOUND FAMILIAR?

Functions abstract behavior ("procedural information")
Symbolic constants abstract data

Can use a variable:

```c
float PI = 3.14159;
```

Even better, use a const

```c
const float PI = 3.14159;
```

EXAMPLE: WASHER AREA FUNCTION

Specification:
Write a function to find the area of a washer given the inner radius and outer radius. Assume you already have another function that calculates the area of a circle:

```c
float DiskArea (float r);
```

```c
float WasherArea (float inner, float outer) {
 float innerArea, outerArea, areaOfWasher;
 innerArea = DiskArea (inner); 
 outerArea = DiskArea (outer); 
 areaOfWasher = outerArea - innerArea; 
 return areaOfWasher;
}
```

STATIC CALL GRAPH

Shows which function calls which...

```
main
  WasherArea
    DiskArea
```
THE WHOLE PROGRAM

```cpp
float DiskArea(float r) {
    return PI * r * r;
}

float WasherArea(float inner, float outer) {
    float innerArea, outerArea;
    innerArea = DiskArea(inner);
    outerArea = DiskArea(outer);
    return outerArea - innerArea;
}
```

```cpp
int main() {
    float inner, outer, area;
    cout << "Input inner radius and outer diameter:" << endl;
    cin >> inner;
    cin >> outer;
    area = WasherArea(inner, outer/2.0);
    cout << area << endl;
    return 0;
}
```

SUMMARY

Today's big idea: **Abstraction**

Procedural or behavioral abstraction: **Functions**

Data abstraction: **Constants**

LAB 3 PREP: GEOMETRY REVIEW

What shape is formed between the line and the x axis? What is the area under the line?

TRAPEZOID

\[ A = \frac{h(a+b)}{2} \]