

Functions + Debugger

Section 1D

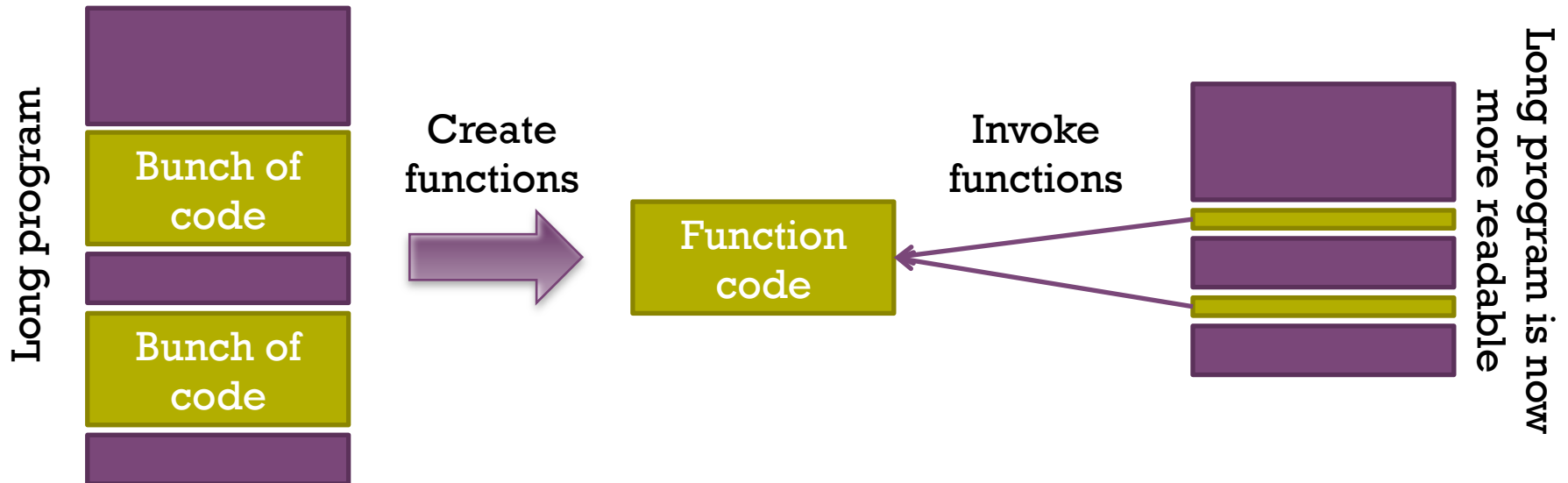
+ Why do we need to write functions?

■ Readability

- Our program is not cluttered with complex code.

■ Reusability

- We can use the same “module” here and there, whenever we have to perform the same thing.



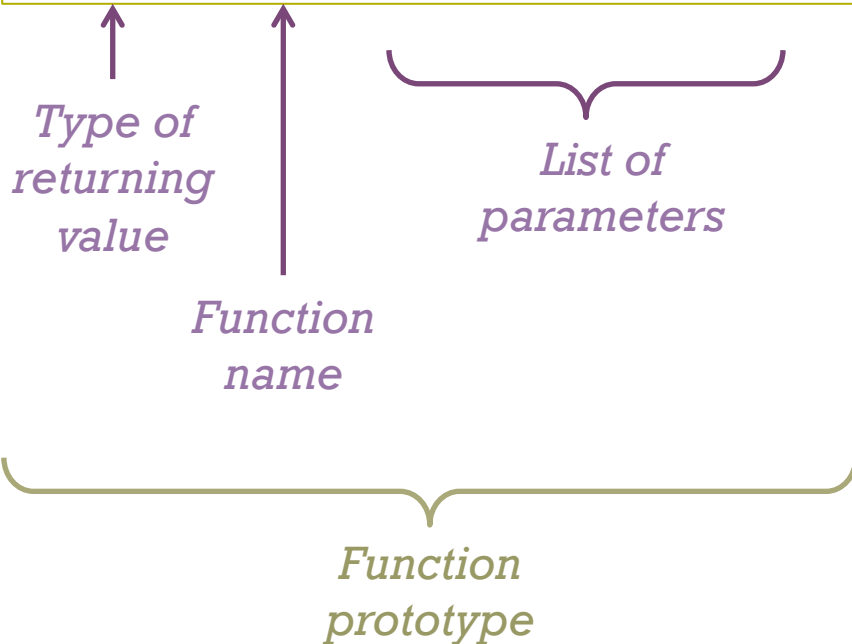


How do we create a function?



✧ Declaration:

```
double power( double, int );
```



✧ Definition:

```
// Function header.
double power( double x, int y )
{
    // Function body.
    double p = 1.0;
    for( int i = 0; i < y; i++ )
        p *= x;

    return p;
}
```



Anything else about declaring a function?



- Declare a function ***before*** it is used.
- If your function does not return anything, use `void`.
 - `void greet(string name);`
- If your function has more than one input parameter, use commas to separate them.
 - `double bodyMassIndex(double height, double weight, string name);`
- You can disregard parameter names ***ONLY*** in the function declaration.
 - `double bodyMassIndex(double, double, string);`





Anything else about defining a function?



- You may define and declare your function at the same time!
- A function has its own block or space of variables, known as “**scope**”
 - Variables within a function are called “**local**.”
 - Variables outside **all** functions are called “**global**” and can be accessed from everywhere.
- The input parameters (if any) are variables declared in the scope of the function (see later for calling functions by value and by reference).
- Use a **return** statement to finish (prematurely) and/or return a value.
 - **return** x * x;
 - **return**; // If function was defined as **void** function()

+ Example

```
double power( double, int );
```

Declaration

```
int main( )  
{  
    double base;  
    cin >> base;  
    int exponent;  
    cin >> exponent;  
  
    cout << power( base, exponent ) << endl;  
}
```

Usage

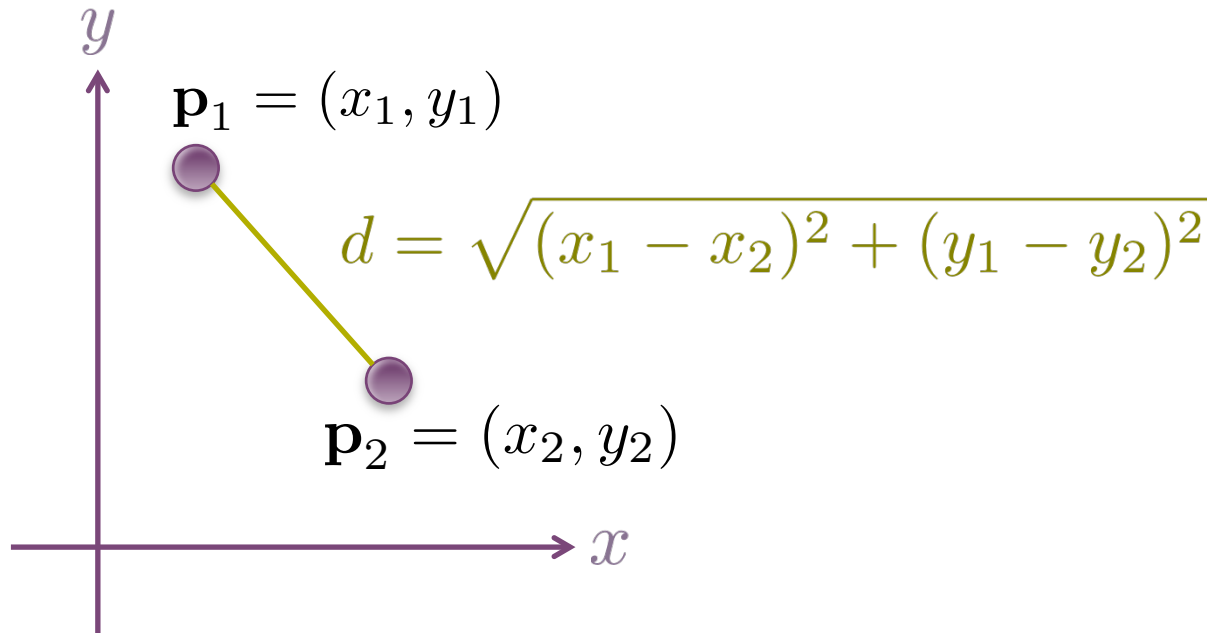
```
double power( double x, int y )  
{  
    double p = 1.0;  
    for( int i = 0; i < y; i++ )  
        p *= x;  
    return p;  
}
```

Definition



Exercise

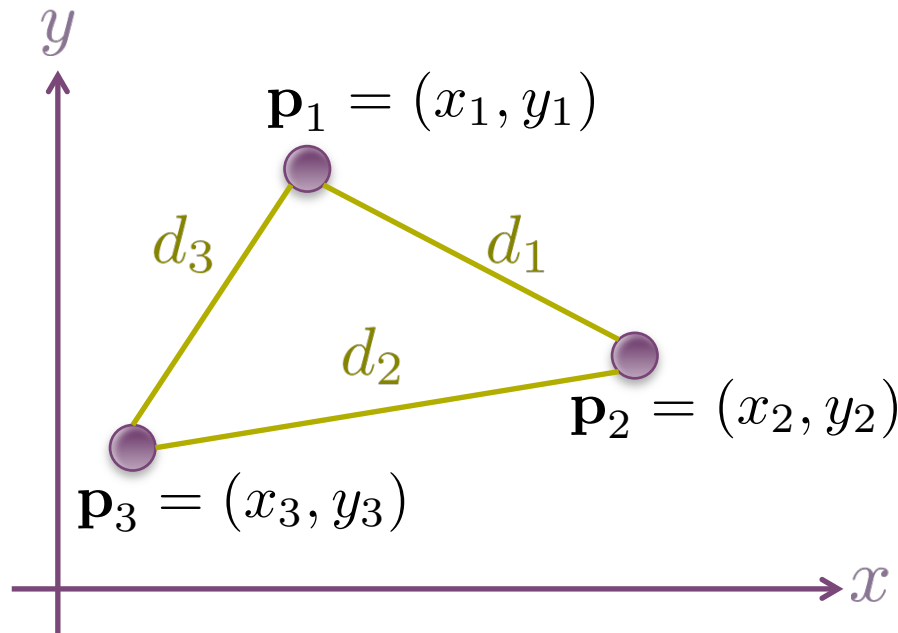
- Write a function that computes the distance between two 2D points.





Exercise

- Write a program that computes the perimeter of a triangle. The user must provide the 2D coordinates of the three points. Use the function you wrote in the previous exercise.



+ Exercise

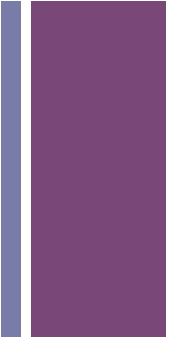
- Is there an error in the following function. If so, how can we fix it? If not, what does the function do?

```
int g( void )
{
    cout << "Inside function g" << endl;

    int h( void )
    {
        cout << "Inside function h" << endl;
    }
}
```



Exercise

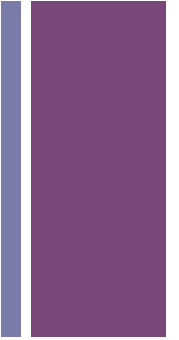


- Is there an error in the following function. If so, how can we fix it? If not, what does the function do?

```
int sum( int x, int y )  
{  
    int result;  
  
    result = x + y;  
}
```



Exercise



- Is there an error in the following function. If so, how can we fix it? If not, what does the function do?

```
void f( string a )  
{  
    string a;  
    cout << a;  
}
```

+ Exercise

- Is there an error in the following function. If so, how can we fix it? If not, what does the function do?

```
void product( void )
{
    int a, b, c, result;

    cout << "Introduce three integers:" << endl;
    cin >> a >> b >> c;
    result = a * b * c;

    cout << "The result is: " << result << endl;

    return result;
}
```



Exercise

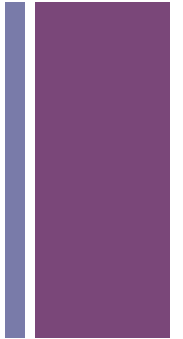


- Write a function that displays a solid square of characters `fillChar`, whose side length is specified by `side`. For instance, if `side = 4`, and `fillChar = '#'`, then, the function outputs:

```
####  
####  
####  
####
```



Exercise



■ The **Fibonacci** series

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

starts with 0 and 1, and has the property that each term is the sum of the its previous two terms. Write a function `int fibonacci(int n)`, that computes the n^{th} Fibonacci term in the series. For example:

`fibonacci(0) = 0`

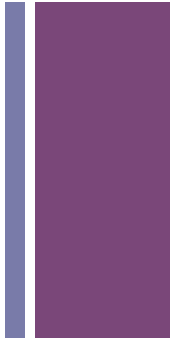
`fibonacci(2) = 1`

`fibonacci(5) = 5`

`fibonacci(8) = 21`



Parameters by value and by reference



✧ By value

Caller

```
double squareRoot( double x )  
{  
    return sqrt( x );  
}
```

```
...  
double v;  
cin >> v;  
  
cout << squareRoot( v );  
...
```

Local **x**



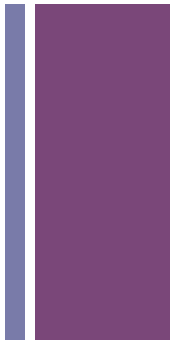
Local **v**



Two different contexts or scopes!



Parameters by value and by reference



✧ By reference

Caller

```
bool squareRoot( double x, double& s )
{
    if( x < 0 )
        return false;

    s = sqrt( x );
    return true;
}
```

```
double r;
double v;
cin >> v;

if(squareRoot( v, r ))
    cout<< r;
else
    cout<< "Imaginary";
```

Local *x*



Local *s*



Local *r*



Local *v*





Exercise



- A **palindrome** is a word, phrase, number, or a sequence of characters which reads the same backward and forward, like:
“Race Car”, “taco cat”, “Was it a car or a cat I saw?”

Write a function `bool isPalindrome(string s, string& r)`, that checks whether string `s` is a palindrome. If so, it should return `true` and leave `r` unchanged. If not, it should return `false` and copy into `r` string `s` but backwards. Assume that `s` contains only letters (no spaces, digits, or other punctuation characters).



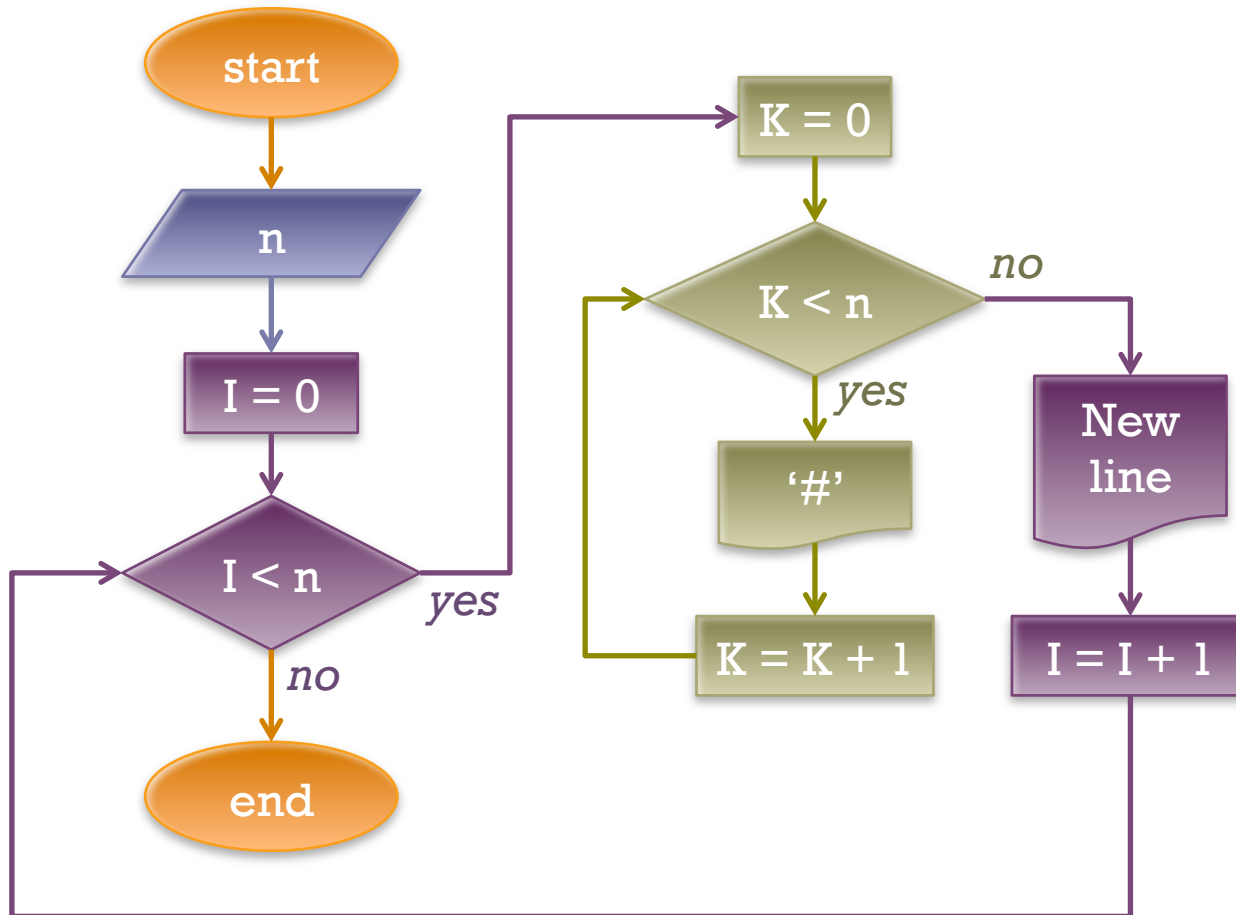
Exercise



- Design a function `bool strToNumber(string s, int& n)` that converts a “numeric” string `s` into an integer `n` if `s` is valid (i.e. contains only digits) and returns `true`. If `s` is invalid, the function returns `false`, and `n` is left unmodified.
- `s = “012345”`, function returns `true`, and `n = 12345`
- `s = “0.12k”`, function returns `false`, and `n` is undefined

+ A word on pseudocode

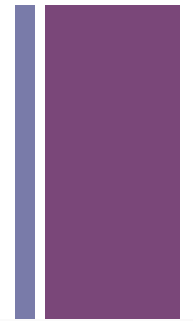
- Translate your (detailed) C++ code to simple words (or the other way around?) - Recall **flow charts**!



Pseudocode

Read a number n from the user.
Iterate n times:
Print n '#' characters starting from the left of a new line.

+ The Debugger (Xcode)



```
37 bool isPalindrome( string s, string& r )
38 {
39     string lowerS = "";
40     string backwardS = "";
41     string lowerBackwardS = "";
42
43     for( size_t I = 0; I < s.size(); I++ )
44     {
45         char c = tolower( s[I] );
46         lowerS += c;
47         backwardS = s[I] + backwardS; // This backward version contains characters in upper/lower case.
48         lowerBackwardS = c + lowerBackwardS; // This backward version contains characters only in lower case.
49     }
50
51     if( lowerS == lowerBackwardS )
52         return true;
53
54     r = backwardS;
55     return false;
56 }
57
```

Thread 1: breakpoint 1.1

0 isPalindrome(std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> >&)

- ▶ **r** = (string &) ""
- ▶ **s** = (std::__1::string) "HelloWorld"
- ▶ **lowerBackwardS** = (std::__1::string) ""
- ▶ **I** = (size_t) 1
- ▶ **backwardS** = (std::__1::string) ""
- ▶ **lowerS** = (std::__1::string) ""

(lldb)



The Debugger (Visual Studio)

The screenshot shows the Visual Studio IDE with the following components:

- Code Editor:** Displays the `isPalindrome` function in `palindrome.cpp`. The function takes a `string s` and a `string& r` as arguments. It initializes `lowerS`, `backwardS`, and `lowerBackwardS` to empty strings. A `for` loop iterates over the characters of `s`, converting them to lowercase and building the `backwardS` and `lowerBackwardS` strings. The function returns `true` if `lowerS` equals `lowerBackwardS`, otherwise `false`. A breakpoint is set at line 11, and the execution has stopped at line 12.
- Autos Window:** Shows the current state of the variables in the scope of the function call.

Name	Value	Type
I	9	unsigned int
backwardS	"dlroWolleH"	std::basic_string<char, std::char_traits<char>, std::allocator<char>>
lowerBackwardS	"dlrowolleh"	std::basic_string<char, std::char_traits<char>, std::allocator<char>>
s	"HelloWorld"	std::basic_string<char, std::char_traits<char>, std::allocator<char>>
- Call Stack Window:** Shows the sequence of function calls that led to the current state.

Name	Lang
Session 4.exe!isPalindrome(std::basic_string<char, std::char_traits<char>, std::allocator<char>>& s, std::basic_string<char, std::char_traits<char>, std::allocator<char>>& r)	C++
Session 4.exe!main() Line 23	C++
[External Code]	
[Frames below may be incorrect and/or missing, no symbols loaded]	

The bottom of the window shows the **Autos** tab selected, with other tabs like **Locals**, **Threads**, **Modules**, and **Watch 1** visible. The **Call Stack**, **Breakpoints**, and **Output** tabs are also present at the bottom right.