FUNCTIONS AND POINTERS

Dr. Johnson

School of Mathematics

Semester 1 2012
Part 2:
- Functions - how and where to use them
- Pointers - dynamic allocation, pass by reference

Aims - week 2:
- Understand precision and errors within a program.
- Use functions, loops and if statement to control a program.
- Understand the concept of a pointer.
**Why Use Functions?**

- Add structure to your code
- Readability of your algorithms
- Code reuse - why write something twice?
**Why Use Functions?**

- Add structure to your code
- Readability of your algorithms
- Code reuse - why write something twice?
- Can be written in different files
- Function can be stored in libraries
### How to Use Functions?

Functions must be declared **before** the main program.

All functions must return a value of the data type specified in the declaration.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>FunctionName</th>
</tr>
</thead>
<tbody>
<tr>
<td>( TYPE</td>
<td>argumentNames )</td>
</tr>
<tr>
<td></td>
<td>function statements</td>
</tr>
<tr>
<td></td>
<td>....</td>
</tr>
<tr>
<td></td>
<td>....</td>
</tr>
<tr>
<td></td>
<td>RETURN value</td>
</tr>
</tbody>
</table>
HOW TO USE FUNCTIONS?

- Functions must be declared **before** the main program.
- All functions must return a value of the data type specified in the declaration.
- Even if this is **void**!
- Use a function as if it is a variable

```markdown
<table>
<thead>
<tr>
<th>TYPE</th>
<th>FunctionName</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( TYPE argumentNames )</td>
</tr>
<tr>
<td></td>
<td>function statements</td>
</tr>
<tr>
<td></td>
<td>....</td>
</tr>
<tr>
<td></td>
<td>....</td>
</tr>
<tr>
<td></td>
<td>RETURN value</td>
</tr>
</tbody>
</table>
```
#include<iostream>
using namespace std

// square an integer
int square(int i)
{
    return i*i;
}

// Main Program
main()
{
    int number=5;
    cout « square(number) « endl;
}
A function must be defined before it can be called.
Use prototypes to declare functions before they are used.

```c
data type function_name(arguments)
```

The main body of the function can be placed somewhere else in the code (or even a separate file)
Simply “include” the header file at the top of your code:

```cpp
#include "myHeaderFile.hpp"
```

- use your IDE to create a project with your header and source file
- individual compilation units must **linked** together
What’s in a name?

- Pointers are an important mechanism in any computer program.
- In many languages pointers are *hidden* from the user.
- Pointers store *location*, rather than value.

---

```
<table>
<thead>
<tr>
<th>Memory</th>
<th>Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xfff8</td>
<td>0xfff4</td>
</tr>
<tr>
<td>6</td>
<td>0xfff8</td>
</tr>
<tr>
<td>a</td>
<td>ptr_a</td>
</tr>
<tr>
<td>0xfff8</td>
<td>0xfff4</td>
</tr>
</tbody>
</table>
```
Using Pointers

- Declare pointers by putting `*` in front of the variable name.
- This means that the data it contains is a *memory address*. 
Declare pointers by putting * in front of the variable name.

This means that the data it contains is a memory address.

There are two operators & and * associated with pointers.

& in front of a variable means return the memory address of the variable.

* in front of a variable means return the value of data at that memory address.
Using Pointers

- Declare pointers by putting \* in front of the variable name.
- This means that the data it contains is a *memory address*.
- There are two operators \& and \* associated with pointers.
- \& in front of a variable means return the *memory address* of the variable.
- \* in front of a variable means return the value of data at that *memory address*.
- We can use pointers like the variable itself by putting a \* in front of it.
#include <iostream>

int main() {
    double a, *ptr_a;
    a = 6;
    ptr_a = &a;
    std::cout << "a = " << a << " address of a = " << &a << std::endl;
    std::cout << "ptr_a = " << ptr_a << " address of ptr_a = " << &ptr_a << std::endl;
    return 0;
}
main()
{
double a,*ptr_a;
a = 6;
ptr_a = &a;
cout << " a = " << a << endl;
*ptr_a = 2; NB. use pointer like it was the variable
cout << " a = " << a << " and ptr_a = " << *ptr_a << endl;
}


**Simple Arrays**

- We can declare an *array* of any data type or object.
- Simply write:

  ```
  data type arrayName[number of elements];
  ```

  ```
  int array[3];
  ```

  ![Diagram of an array with elements](image)
**Simple Arrays**

- We can declare an *array* of any data type or object.
- Simply write:

```
data type arrayName[number of elements];
```

```c
int array[3];
array[0] = 3;
```

```
array[ 0 ]
array[ 1 ]
array[ 2 ]
```

```
3

int
```
Simple Arrays

- We can declare an *array* of any data type or object
- Simply write:

```plaintext
data type arrayName[number of elements];
```

```c
int array[3];
array[0] = 3;
array[1] = 5;
```
We can declare an *array* of any data type or object

Simply write:

```c
data type arrayName[number of elements];
```

```c
int array[3];
array[0] = 3;
array[1] = 5;
array[2] = array[0] + array[1];
```

```
  int
array [ 0 ] 3
array [ 1 ] 5
array [ 2 ] 8
```
An array is just a pointer

- An array name is just a pointer to a block of data.
- We can use pointers instead of arrays for dynamic memory allocation.

```
array
0xfff0
0xfff4
0xfff8
```

Compiler

```
0xfff0
0xfff4
0xfff8
```

Memory

```
0xfff0
0xfff4
0xfff8
```

Array Storage

```
0xfff0
0xfff4
0xfff8
```

```
6
3
1
```

0xfff0
0xfff4
0xfff8

```
array
```

```
0xfdd0
```

0xfdd0

```
0xffff0
```

Dr. Johnson

MATH49111
We use the **new** and **delete** keywords to instigate dynamic storage

**WARNING** - dynamic storage is extremely dangerous for your programs if you don’t know what you’re doing!!!

```cpp
int n,*array;
cin » n;
array = new int[n];
for(int i=0;i<n;i++)
    { array[i]=i;cout « array[i]; } 
delete [] array;
```
**Pointers and Functions**

- When we pass a variable into a function, the stored **value** is copied into the function, not the variable itself.
- To change the value of the variable itself, we must pass the reference to the memory location.

```c
void swap(double &a, double &b)
{
    NB. stuff in here
}
main()
{
    double a=1, b=2;
    swap(a, b);  // NB. passing by reference
    cout « " a " « a « " b " « b;
}
```
Topics:
- Errors and precision
- Flow control - if, else, for, do
- Functions - how and where to use them
- Pointers - dynamic allocation, pass by reference

Aims - week 2:
- Understand precision and errors within a program.
- Use functions, loops and if statement to control a program.
- Understand the concept of a pointer.