

PROGRAMMING LANGUAGES 1

Functions



FUNCTIONS

- Allow code modularity
- Each C/C++ program has at least one function (the main function)
- A function:
 - Performs certain task
 - Takes zero or more **arguments** as input
 - Returns one output
 - Called from another function to be executed
- Functions have:
 - Declaration
 - Definition
 - Call

FUNCTION DECLARATION

```
returnType functionName(arguments) ;
```

- Declares the function existence
- Typically in header file (.h, .hpp)
- Cannot be within the body of another function
- Function name follows the same conventions as variable names
- Argument types are required, if any
- Argument names can be provided, but not required
- Return type and input types can be any data type or void
- Return type cannot be empty

FUNCTION DECLARATION - EXAMPLES

```
float circleArea(float);
```

```
float rectangleArea(float, float);
```

```
int sum(int[], int);
```

```
void print(int [], int);
```

FUNCTION DEFINITION

```
returnType functionName(arguments) // Function header
{
    // Function body
}
```

- Defines the function behavior
- Typically in source file (.c, .cpp)
- Cannot be within the body of another function
- Must have the same function prototype
- Argument names must be provided
- Function *declaration* is optional if *definition* is provided before call

FUNCTION CALL

```
int main() // or any other function
{
    ...
    returnType variable = functionName(inputs);
    ...
}
```

- Calls the function to be executed, i.e. code not executed until called
- Must be within the body of another function, e.g. the main function
- Takes as input variables of the specified argument types
- Returns a value or void as an output of the specified return type

AVERAGE CALCULATION - REVISITED

```
float average(float [], int); // declaration
```

```
float average(float [] x, int n) // definition
{
    float average = 0;
    for(int i 0; i < n; i++) {
        average += x[i];
    }
    average /= n;
    return average;
}
```

AVERAGE CALCULATION - CONTINUED

```
int main() {  
    float array[100];  
    for(int i 0; i < 100; i++) {  
        cin >> array[i];  
    }  
    float a = average(array); // Call  
    cout << "Average = " << a << endl;  
}
```

IDEALLY

average.h

```
#ifndef AVERAGE_H
#define AVERAGE_H

// declaration
float
average(float[]);

#endif
```

average.cpp

```
#include "average.h"

// definition
float average(float[]
x)
{

}
```

main.cpp

```
#include "average.h"

int main( )
{
    // Call
    average(x)
}
```

#ifndef

- Short for: If not defined
- Uses any **unique** name, e.g. AVERAGE_H
- Followed by #define
 - Using the exact **same** name (copy & paste)
- Guarantees function is not included more than once

```
#ifndef AVERAGE_H  
#define AVERAGE_H  
  
// declaration  
float  
average(float[]);  
  
#endif
```

VARIABLE SCOPE

- Variable scope defines the area where the variable is accessible
- A **local** variable defined in a function is only accessible within that function
- A **global** variable defined outside functions is accessible to all functions

VARIABLE SCOPE - EXAMPLE

```
int n; // global
float average(int x[])
{
    float a; // local to average()
    // Only n and a are visible here
}
int main()
{
    int x[]; // local to main()
    // Only n and x are visible here
}
```

VARIABLE LIFETIME

- Variable lifetime define how long a variable exists (is alive)
- A global variable has the lifetime of the program
- A local variable is alive during the execution of the function
- A pointer is alive until delete is called on that pointer

STATIC VARIABLES

- Defined using the static keyword
- Should be initialized, otherwise, takes default value of 0
- Scope can be local or global
- A static variable (local or global) has the lifetime of the program

STATIC VARIABLES - EXAMPLE

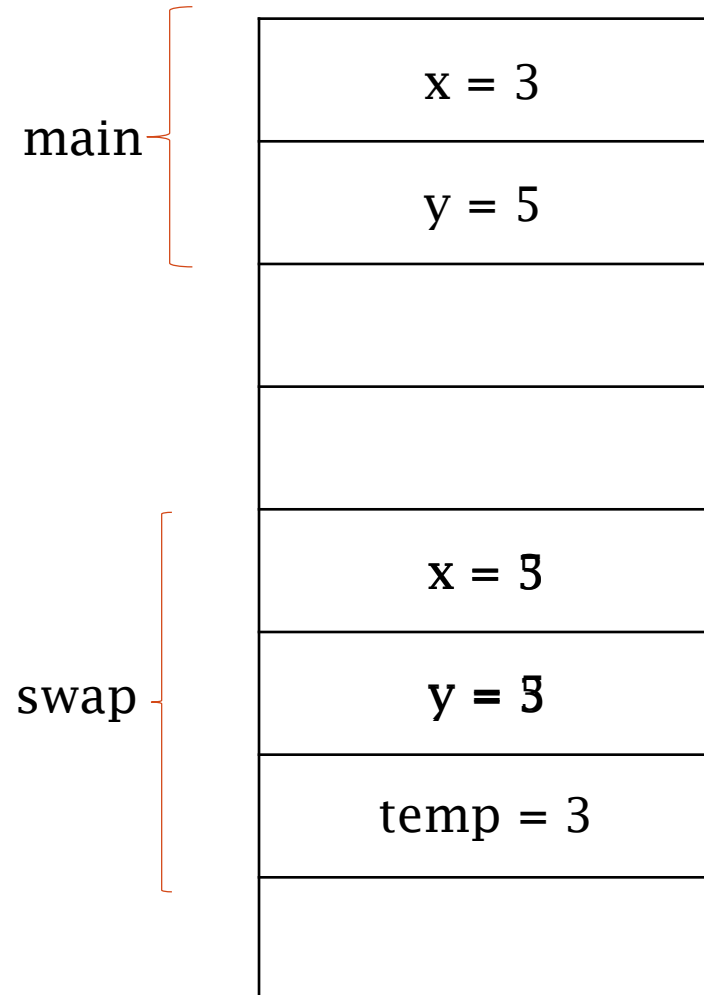
```
// Count the number a function is called
void f() {
    static int count = 0; // first time only
    // Do something
    count++; // count is incremented everytime
}
```

PASS BY VALUE

- By default, variables are passed to functions by value in C and C++
 - A copy is created and passed to the function
 - Any change to the function copy does not affect the original variable

SWAP - WRONG WAY

```
void swap(int x, int y) {  
    int temp = x;  
    x = y;  
    y = temp;  
}  
  
int main ( ) {  
    int x = 3, y = 5;  
    swap(x, y);  
    cout << x << ", " << y;  // 3,5 !!  
}
```

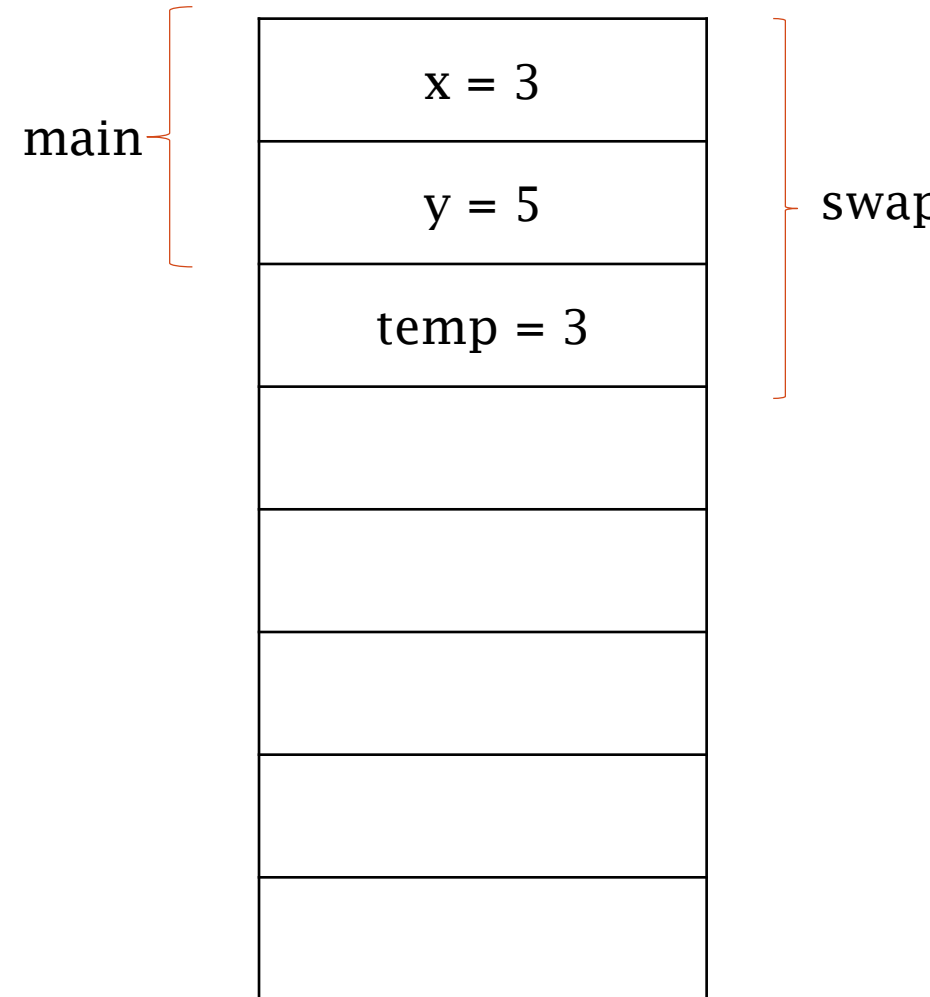


PASS BY REFERENCE

- A reference to the variable can be passed to the function
- Helps save space in memory
- In C, pass by reference was done using pointers

SWAP - DONE RIGHT

```
void swap(int & x, int & y) {  
    int temp = x;  
    x = y;  
    y = temp;  
}  
  
int main ( ) {  
    int x = 3, y = 5;  
    swap(x, y);  
    cout << x << ", " << y;  // 5,3  
}
```



SWAP IN C - OLD WAY

```
void swap(int * x, int * y) { // Using pointers
    int temp = *x;
    *x = *y;
    *y = temp;
}
```

```
int main ( ) {
    int x = 3, y = 5;
    swap(&x, &y);
    cout << x << " , " << y; // 5,3
}
```

main

swap

x = 3
y = 5
&x = 0x1234
&y = 0x1238
temp = 3

CONSTANT ARGUMENTS

- Constant argument cannot be changed by the function
- A constant reference should be passed if variable is not altered by function
 - Helps prevent unintended modification

```
int max(const int & x, const int & y) {  
    return x > y ? x : y;  
}
```

RETURNING MULTIPLE VALUES

- Use an array as the return type
 - Function caller access needed values by index
- Pass empty result variables by reference
 - Set by the function

MIN & MAX – USING ARRAY

// Find min and max of 3 numbers

```
int[ ] minMax(int x, int y, int z)
{
    int m[2]; // m[0] is min and m[1] is max
    m[0] = m[1] = x;
    if(y < m[0]) m[0] = y;
    else if(y > m[1]) m[1] = y;
    if(z < m[0]) m[0] = z;
    else if(z > m[1]) m[1] = z;
    return m;
}
```

```
int main()
{
    int x,y,z, m[2];
    cin >> x >> y >> z;
    m = minMax(x,y,z);
    cout << "Min = " << m[0] << endl;
    cout << "Max = " << m[1] << endl;
}
```

MIN & MAX – USING REFERENCES

```
void minMax(int x, int y, int z, int &min, int &max)
```

```
{
```

```
    min = max = x;
```

```
    if(y < min) min = y;
```

```
    else if(y > max) max = y;
```

```
    if(z < min) min = z;
```

```
    else if(z > max) max = z;
```

```
}
```

```
int main()
```

```
{
```

```
    int x,y,z, min, max;
```

```
    cin >> x >> y >> z;
```

```
    minMax(x, y, z, min, max);
```

```
    cout << "Min = " << min << endl;
```

```
    cout << "Max = " << max << endl;
```

```
}
```

PASSING ARRAYS

- Arrays are always passed by reference, i.e. not copied
- An array can be passed as an array [] or as a pointer *
- Array length should also be passed to the function
- A 2-D array should be passed as a pointer to pointer

```
void print(int []x, int n);
```

```
void print(int * x, int n);
```

```
void print(int **x, int n, int m);
```

RECURSIVE FUNCTIONS

- A function that calls itself, directly or indirectly
- Must have a stopping condition
 - Otherwise, it causes stack overflow error
- More computationally expensive than iterative approach

RECURSIVE FUNCTIONS - EXAMPLE

// Return element n of Fibonacci series 1,1,2,3,5,8,...

```
int fib(int n) {  
    if (n < 2) return 1; // Stopping condition  
    return fib(n-1) + fib(n-2);  
}
```

FUNCTION OVERLOADING

- Defining different functions using the same name
- Each definition must differ in number or type of arguments

FUNCTION DEFAULT ARGUMENTS

- A default value for function argument can be provided
- If an argument has a default value, all following arguments must have defaults

TEMPLATES

Allow for generic-type functions or classes, i.e. allow any type

TEMPLATES - EXMAPLE

```
template <class T>
T sum(T [] x, int n) // T can be any data type
{
    T sum = 0;
    for(int i = 0; i < n; i++) sum+= x[i];
    return sum;
}
```

TEMPLATES - EXMAPLE

```
int main( ) {  
    int x[] = {1,2,3,4};  
    cout << sum(x, 4) << endl; // T is int  
    float y[] = {1.4,2.3,4.7};  
    cout << sum(y, 3) << endl; // T is float  
    ...  
}
```

SUMMARY

- Functions
- Variable scope
- Variable lifetime
- Static variables
- Pass by value & Pass by reference
- Constant arguments
- Returning multiple values
- Recursive Functions
- Function Overloading
- Default Arguments
- Templates

NON-ACADEMIC READING

