

# **ED1021 - Introduction to computation and visualisation**

## **L10 - Functions in C**

**Ramanathan Muthuganapathy (<https://ed.iitm.ac.in/~raman>)**

**Course web page: <https://ed.iitm.ac.in/~raman/introcomp.html>**

**Moodle page: Available at <https://courses.iitm.ac.in/>**

# Functions (User-defined)

- Functions enable to split the code into different segments.
- Reusability - someone else can use that function for a similar purpose.
- Maintaining/Managing the code becomes easier.

# Fahrenheit to centigrade

## Demo using L10\_function.c

```
float fToc(float c); // Function prototype

float fToc(float f) { // Function definition
    float c;
    c = 5.0/9*(f-32);
    return c;
}

int main() {
    float faren, centi;
    scanf("%f", &faren);
    centi = fToc(faren); // Calling the function
    printf("centigrade = %f\n", centi);
}
```

# Components of a function

- Function prototype
- Function definition
- Calling the function (or function calling)

# Function definition (written outside main( ))

one of the components of a function

```
Return_Type Nameofthefunction(DT fa1, DT fa2....., ) { //DT- datatype, fa - formal argument
```

```
}
```

```
float fToc(float f) { // Function defintion
```

```
}
```

# Function (written outside main())

## Function definition + body of the function

```
Return_Type Nameofthefunction(DT fa1, DT fa2....., ) { //DT- datatype, fa - formal argument
```

```
    St1;
```

```
    St2;
```

```
    // Body of the function
```

```
    return var (or constant/ exp)
```

```
}
```

```
    float fToc(float f) { // Function definition, f-formal arg.
```

```
        float c;
```

```
        c = 5.0/9*(f-32);
```

```
        return c;
```

```
    }
```

# Function Prototype - same as fn. defn. with ; (written above function definition)

Return\_Type Nameofthefunction(DT fa1, DT fa2....., ); // sometime written without fa1, fa2 etc.. (only DT is specified)

```
float fToc(float f); // Prototype
```

# Calling the function (written within another function, main( ))

## one of the components of a function

var = Nameofthefunction(arg1, arg2....., ); // arg1... - actual arguments

`centi = fToc(faren); // Calling the function`

```
int main() {  
    float faren, centi;  
    scanf("%f", &faren);  
    centi = fToc(faren); // Calling the function, faren – actual arg.  
    printf("centigrade = %f\n", centi);  
}
```



HWs (functions) - FP, FD, main( )

- 1) Addition of two numbers
- 2) Subtraction of two numbers
- 3) Multiplication
- 4) Division

Demo using L10\_HW.c

IMP HW: Change the return type to float in all the functions (use another file to do this problem).

# Functions

## some points

Function can return only one value (or const or expression) at the most.

The number of arguments can be zero but can be more (if it is too many, then the code will become difficult to maintain)

Max. of 6 or 7 arguments is okay.

All user-defined functions have to be called for them to operate.

# Recall

## components of a function

- Function prototype
- Function definition
- calling the function

# Function for factorial

## L10\_factorial.c

```
#include <stdio.h>
```

```
long int factorialfn(long int num);
```

```
long int factorialfn(long int num)
{
    long int i, fact = 1;
    for (i = 1; i <= num; i++) {
        fact *= i;
    }
    return fact;
}
```

```
int main(void) {
    long int num1, fact;
    printf("Enter a number: ");
    scanf("%ld", &num1);
    fact = factorialfn(num1);
    printf("The factorial of %ld is %ld\n", num1, fact);
}
```

# Scope of a variable

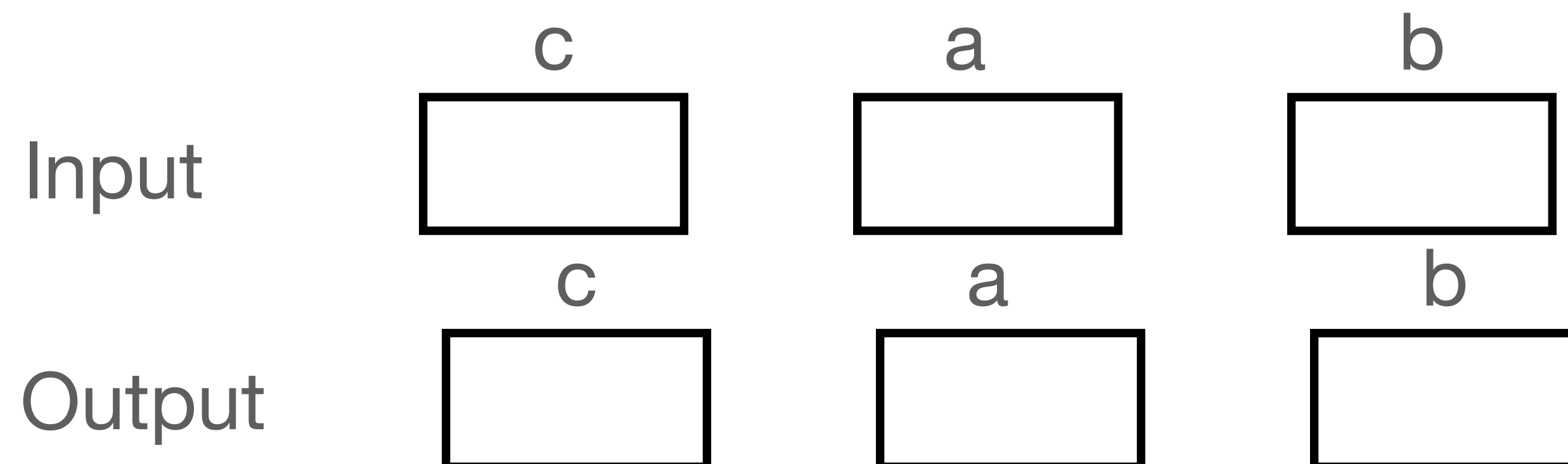
**L10\_ScopeLocal.c, L10\_ScopeGlobal.c**

- Local
  - variable is know only within the function
- Global
  - variable is known to all functions.
  - Any change in inside any the functions will be reflected wherever the variable has been used.
- Static

# Swapping two variables

## scalar ones

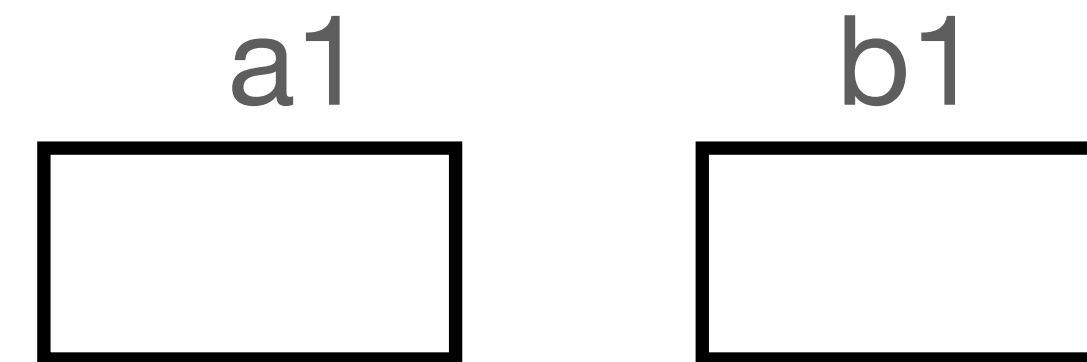
```
int main() {  
    int a, b, c;  
    a = 20; b = 45;  
    printf("Values of a and b before swapping a = %d, b = %d\n", a, b);  
    c = a;  
    a = b;  
    b = c;  
    printf("Values of a and b after swapping a = %d, b = %d\n", a, b);  
}
```



# Swapping using functions

## swap - L10\_Swapfunction.c

```
void swap(int a1, int b1);
```



```
void swap(int a1, int b1) { //a1, b1 – formal arguments
```

```
    int c;
```

```
    c = a1;
```

```
    a1 = b1;
```

```
    b1 = c;
```

```
    printf("Values of a and b after swapping a = %d, b = %d\n", a1, b1);
```

```
    return;
```

```
}
```

```
int main() {
```

```
    int a, b;
```

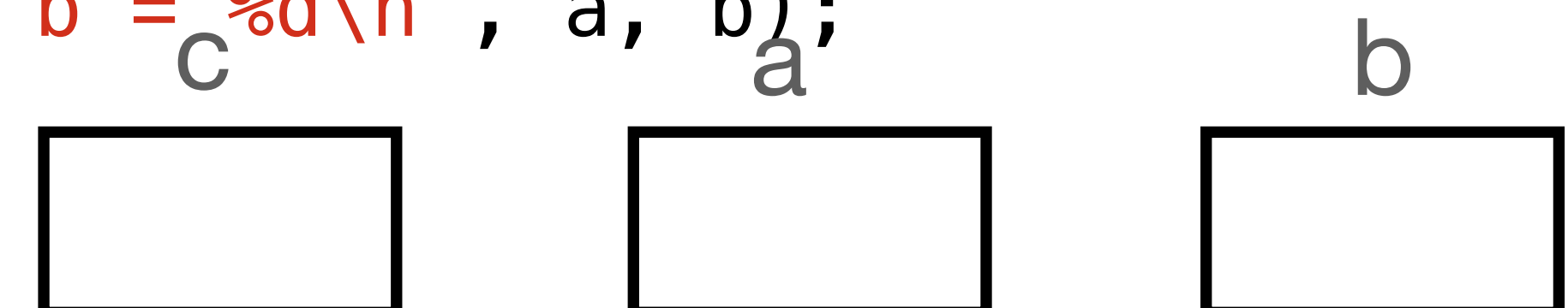
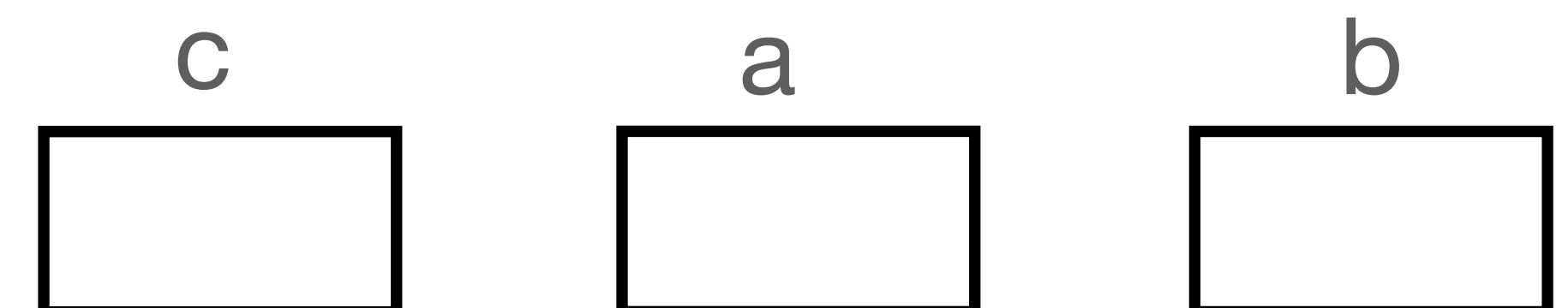
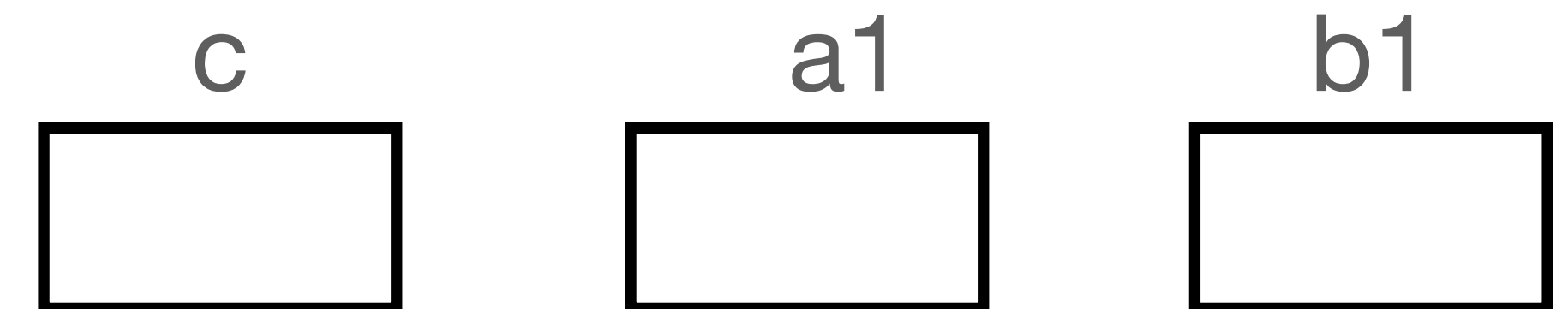
```
    a = 20; b = 45;
```

```
    printf("Values of a and b before swapping a = %d, b = %d\n", a, b);
```

```
    swap(a, b); // a, b – actual arguments
```

```
    printf("Values of a and b after calling a = %d, b = %d\n", a, b);
```

```
}
```



# Swapped valued in main( )

## L10\_GlobalSwap.c

- Global (CW)
  - variable is known to all functions.
  - Any change in inside any the functions will be reflected wherever the variable has been used.
- Combination of local / global (HW)
  - Hint: Pass one variable and use the other as global variable



# Swapping using global variables

## swap - L10\_GlobalSwap.c

```
#include <stdio.h>
```

```
//Function prototype
```

```
void glo_swap(void);
```

```
int g_a, g_b;
```

```
//Function – Function definition with body of the function
```

```
void glo_swap(void) { //a1, b1 are formal arguments
```

```
    int c;
```

```
    c = g_a;
```

```
    g_a = g_b;
```

```
    g_b = c;
```

```
    printf("Values of g_a and g_b after swapping inside the function: g_a = %d, g_b = %d\n", g_a, g_b);
```

```
    return;
```

```
}
```

```
//Main function
```

```
int main() {
```

```
    g_a = 20; g_b = 45;
```

```
    printf("Values of g_a and g_b before swapping g_a = %d, g_b = %d\n", g_a, g_b);
```

```
    glo_swap(); //calling the function, a, b – actual arg.
```

```
    printf("Values of g_a and g_b after swapping, inside the main:, g_a = %d, g_b = %d\n", g_a, g_b);
```

```
}
```

c

g\_a

g\_b

g\_a

g\_b

g\_a

g\_b

# More key points to be noted

- 'Call by value'
- We have been passing only scalar variables!