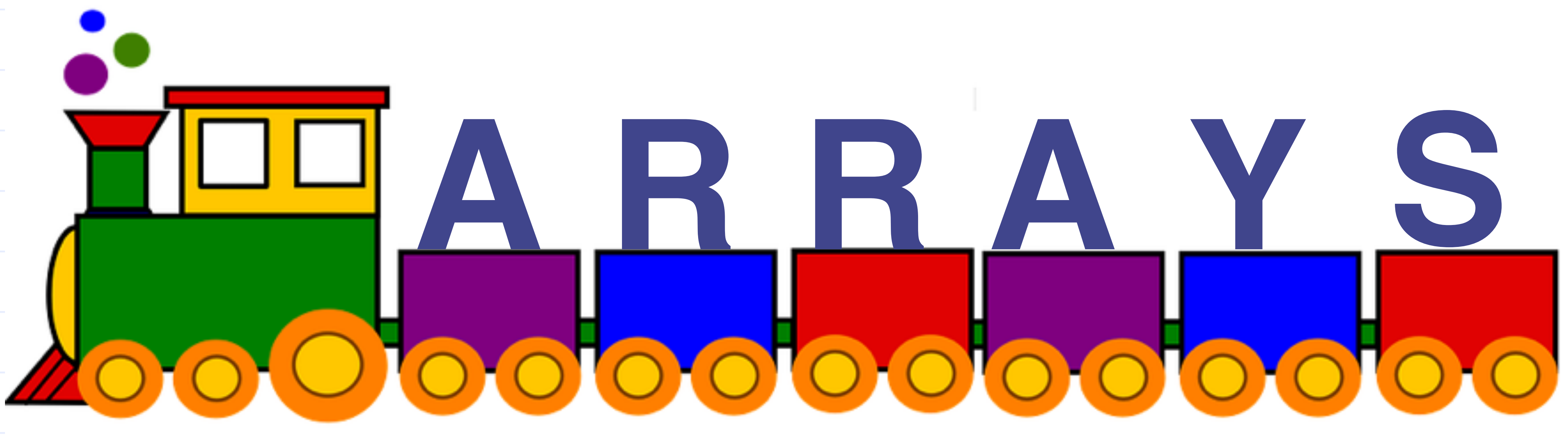
ESC101: Introduction to Computing



String Copy

- Two char arrays src[]and dest[].
- Copy contents of src into dest.
- We assume that dest is declared with size at least as large as src.
- Note the use of ‘\0’ for loop termination

```
// declare and initialise char src[]
// declare char dest[]
   int i;
   for (i = 0; src[i] != '\0’; i++){
          dest[i] = src[i];
}
   dest[i] = '\0';
```

Comparing Two Strings

- Lexicographical Ordering
  - A string str1 is said to be lexicographically smaller than another string str2 if the first character, where the strings differ, is smaller in str1.
- Order of words in a Dictionary

- Examples:
  - "cap" is lexicographically smaller than "cat".
  - "mat" is lexicographically smaller than "matter".

String Comparison

- Given two strings str1 and str2, we want to set the value of a variable flag such that:
  - flag = 0 if the strings are equal,
  - flag = -1 if str1 is lexicographically smaller,
  - flag = 1 if str2 is lexicographically smaller.
- Assumption: The strings contain letters of one case (either capital or small).

Code for string comparison

```
// Declare and initialise two arrays:
// char str1[] char str2[]
  int i=0;int flag;
  while (str1[i]==str2[i]){//skip over same chars
    if (str1[i]=='\0')
      break;
    i++;
  }
  if (str1[i] == str2[i])
    flag=0;
  else if (str1[i] < str2[i])
    flag=-1;
  else          //str2 < str1
    flag=1;
}
```

When can this happen?

At this point, since the first differing characters are such that str1[i] < str2[i], => str1 is smaller

Other operations on strings

- Return length of a string.
- Concatenates one string with another.
- Search for a substring in a given string.
- Reverse a string
- Find first/last/k-th occurrence of a character in a string
  - … and more
- Case sensitive/insensitive versions

Practice Problem

- We are provided with 5 names in input. We have to write a program to read in each name and check its length. The output should be the length of the longest name
- Input:

Amlan

Bhuvesh

Harpreet

Nishant

Prabuddha

- Output: The length of longest name is 9

Solution for Practice problem

```
#include <stdio.h>
int main()
{
   int max=0;
   char name[100];
  return 0;
}
```

Solution for Practice problem

```
#include <stdio.h>
int main()
{
   int max=0;
   char name[100];
   for( int i=0; i<5; i++)
   {
      scanf("%s",name);
  }
   return 0;
}
```

Solution for Practice problem

```
#include <stdio.h>
int main()
{
   int max=0;
   char name[100];
   for( int i=0; i<5; i++)
   {
      scanf("%s",name);
      int j=0;
      while(name[j]!='\0')
         j++;
  }
  return 0;
}
```

Solution for Practice problem

```
#include <stdio.h>
int main()
{
   int max=0;
   char name[100];
   for( int i=0; i<5; i++)
   {
      scanf("%s",name);
      int j=0;
      while(name[j]!='\0')
         j++;
      if(j > max)
         max=j;
   }
  return 0;
}
```

Solution for Practice problem

```
#include <stdio.h>
int main()
{
   int max=0;
   char name[100];
   for( int i=0; i<5; i++)
   {
      scanf("%s",name);
      int j=0;
      while(name[j]!='\0')
         j++;
      if(j > max)
         max=j;
   }
   printf("Longest name length is %d\n",max);
   return 0;
}
```

ESC101: Introduction to Computing

f(unction)

A Modern Smartphone

- Surf the net
  - Input: Web address
  - Output: Desired page
- Book tickets
  - Input: userid, password, booking info, bank info
  - Output: Ticket
- Send email
  - Input: email address of receiver, mail text
  - Output: --
- Take photos
  - Input: --
  - Output: Picture
- Talk (we can do that too!!)
  - Input: Phone number
  - Output: Conversation (if lucky)
- …

Lots of related/unrelated task to perform

- Divide and Conquer
  - Create well defined sub tasks
  - Work on each task independently
    - Development, Enhancements, Debugging
- Reuse of tasks.
  - Email and Chat apps can share spell checker.
  - Phone and SMS apps can share dialer
- C facilitates this using Functions

Function

- An independent, self-contained entity of a C program that performs a well-defined task.
- It has
  - Name: for identification
  - Arguments: to pass information from outside world (rest of the program)
  - Body: processes the arguments do something useful
  - Return value: To communicate back to outside world
    - Sometimes not required

Why use functions?

Example : Maximum of 3 numbers

```
int main(){
   int a, b, c, m;

   /* code to read
    * a, b, c */

   if (a>b){
     if (a>c) m = a;
     else m = c;
   }
   else{
     if (b>c) m = b;
     else m = c;
   }

   /* print or use m */

   return 0;
}
```

```
int max(int a, int b){
   if (a>b)
     return a;
   else
     return b;
}

int main() {
   int a, b, c, m;

   /* code to read
    * a, b, c */

   m = max(a, b);
   m = max(m, c);
   /* print or use m */

   return 0;
}
```

Why use functions?

- Break up complex problem into small sub-problems.
- Solve each of the sub-problems separately as a function, and combine them together in another function.
- The main tool in C for modular programming.

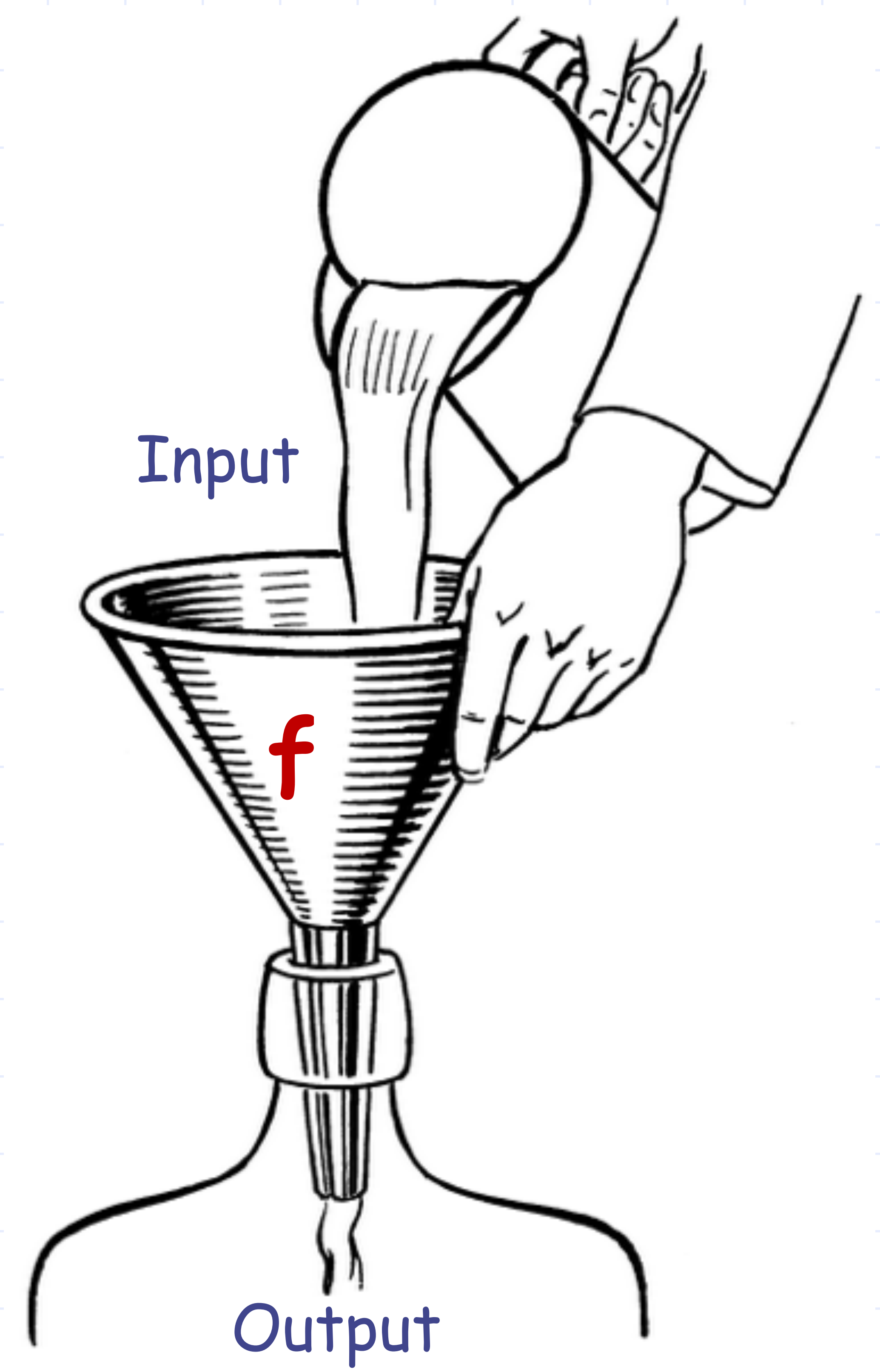
Advantages of using functions

- **Code Reuse**: Allows us to reuse a piece of code as many times as we want, without having to write it.
  - Think of the `printf` function!
- **Procedural Abstraction**: Different pieces of your algorithm can be implemented using different functions.
- **Distribution of Tasks**: A large project can be broken into components and distributed to multiple people.
- **Easier to debug**: If your task is divided into smaller subtasks, it is easier to find errors.
- **Easier to understand**: Code is better organized and hence easier for an outsider to understand it.

We have seen functions before

- main() is a special function. Execution of program starts from the beginning of main().
- scanf(…), printf(…) are standard input-output library functions.
- sqrt(…), pow(…) are math functions in math.h

Parts of a function



```
int max (int a, int b) {
    if (a > b)
        return a;
    else
        return b;
}
```

Return Type

Function Name

2 arguments
a and b,
both of type int.
(formal args)

```
int main () {
    int x;
    x = max(6, 4);
    printf("%d",x);
    return 0;
}
```

Body of the function, enclosed inside { and } (mandatory) returns an int.

Call to the function.
Actual args are 6 and 4.

Function Call

- A function call is an *expression*
  - feeds the necessary values to the function arguments,
  - directs a function to perform its task, and
  - receives the return value of the function.
- Similar to operator application

5 + 3 is an expression of type integer that evaluates to 8

max(5, 3) is an expression of type integer that evaluates to 5

Function Call

- Since a function call is an *expression*
  - it can be used anywhere an expression can be used
  - subject to type restrictions

`printf(“%d”, max(5,3));`	prints 5
`max(5,3) – min(5,3)`	evaluates to 2
`max(x, max(y, z)) == z`	checks if z is max of x, y, z
`if (max(a, b)) printf(“Y”);`	prints Y if max of a and b is not 0.

Returning from a function: Type

- Return type of a function tells the type of the result of function call
- Any valid C type
  - int, char, float, double, …
  - **void**
- Return type is void if the function is not supposed to return any value

```
void print_one_int(int n) {
    printf("%d", n);
}
```

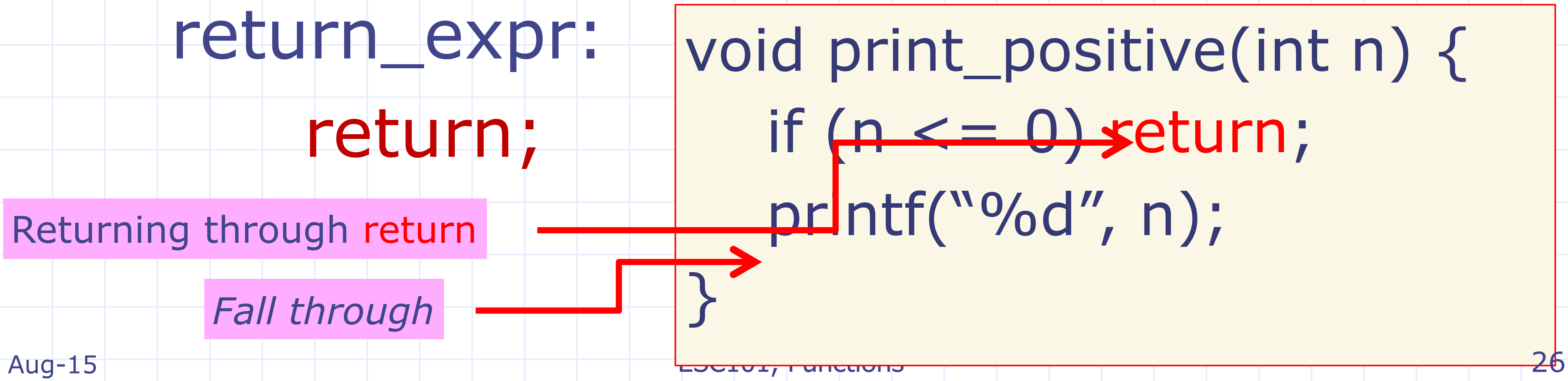
Returning from a function: return statement

- If return type is not void, then the function should return a value:

  return return_expr;

- If return type is void, the function may *fall through* at the end of the body or use a return without return_expr:

  return;



```
void print_positive(int n) {
    if (n <= 0) return;
    printf("%d", n);
}
```

Returning from a function: return statement

- When a return statement is encountered in a function definition
  - control is immediately transferred back to the statement making the function call in the parent function.
- A function in C can return only ONE value or NONE.
  - Only one return type (including void)