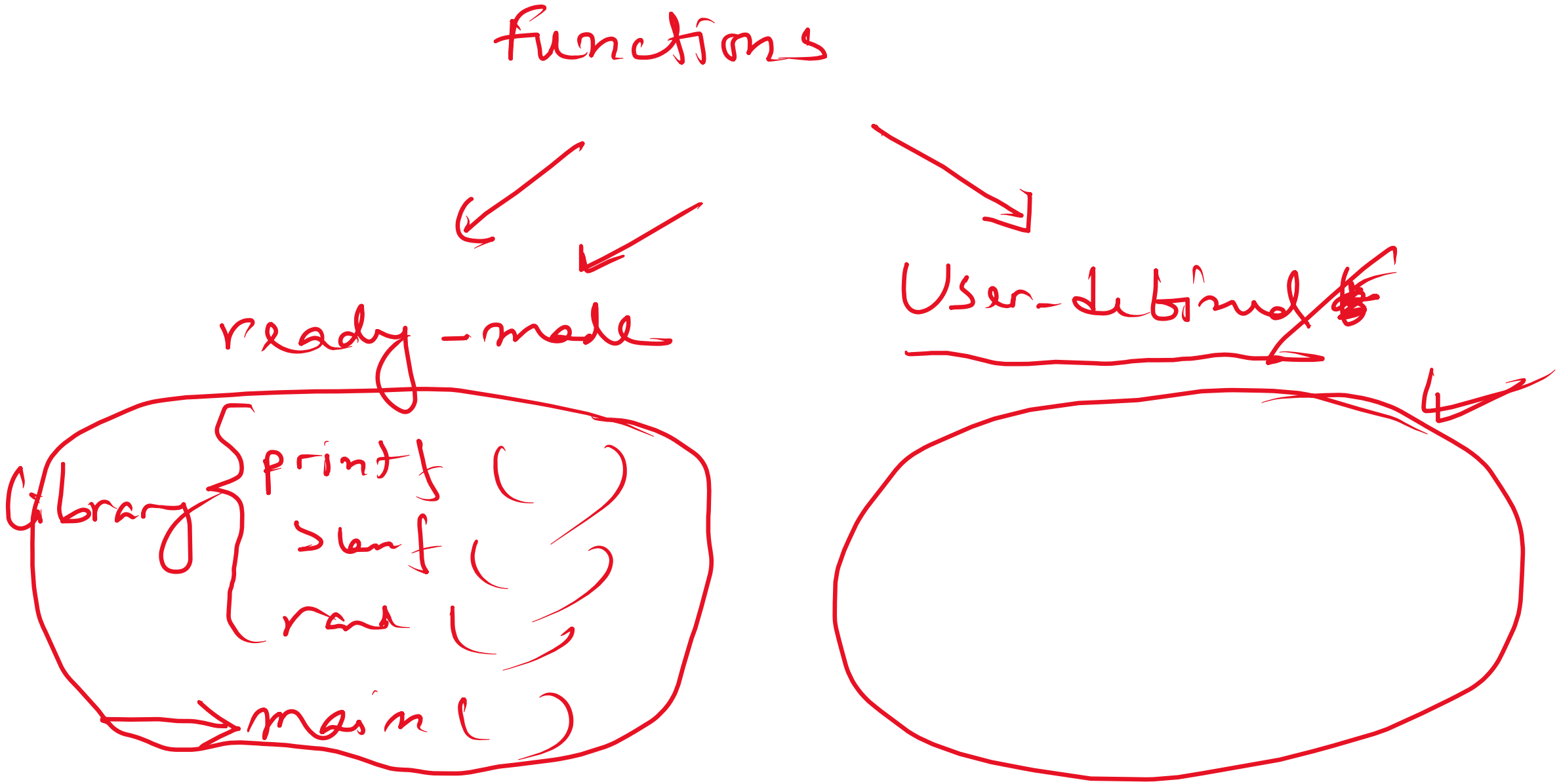
Functions

HK Rana


functions
ready-made
User-defined
library
printf ( )
scanf ( )
rand ( )
main ( )

Why Functions?

- In c, we can divide a large program into the basic building blocks, known as function.
- In other words, we can say that the collection of functions creates a program.
- The function is also known as procedure or subroutine in other programming languages.
- Advantage of functions in C
  - By using functions, we can avoid rewriting same logic/code again and again in a program.
  - We can call C functions any number of times in a program and from any place in a program.
  - We can track a large C program easily when it is divided into multiple functions.
  - Reusability is the main achievement of C functions.
- Limitation
  - However, Function calling is always an overhead in a C program.

Function Definitions

```
return-type function-name ( parameters )
{
    declarations
    statements
}
```

Arguments

- Passed by values
- Demonstrate the example of Factorial

```
average(x, y)
print_count(i)
print_pun()
```

```
num_chars = printf("Hi, Mom!\n");
```

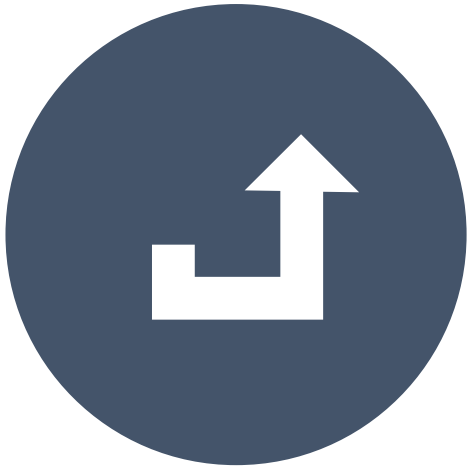
Function Calls

PROGRAM Testing Whether a Number Is Prime

To see how functions can make programs easier to understand, let's write a program that tests whether a number is prime. The program will prompt the user to enter a number, then respond with a message indicating whether or not the number is prime:

```
Enter a number: 34
Not prime
```

Function Declaration



BEFORE

AFTER

Array Arguments

```
int f(int a[])   /* no length specified */
{
  ...
}
```

Although we can use the `sizeof` operator to help determine the length of an array *variable*, it doesn't give the correct answer for an array *parameter:*

```
int f(int a[])
{
  int len = sizeof(a) / sizeof(a[0]);
    /*** WRONG: not the number of elements in a ***/
  ...
}
```

Section 12.3 explains why.

The Return & Others

- The Return Statement
- Program Termination
- The Exit Function

```
exit(0);                    /* normal termination */
```

Since `0` is a bit cryptic, C allows us to pass `EXIT_SUCCESS` instead (the effect is the same):

```
exit(EXIT_SUCCESS);    /* normal termination */
```

Passing `EXIT_FAILURE` indicates abnormal termination:

```
exit(EXIT_FAILURE);    /* abnormal termination */
```