Functions

- Why functions?
  - Break longer jobs into conceptually smaller jobs which are precisely defined.
  - C program generally consists of many small functions.
  - A piece of a code which repeats at many places can be written only once and used again and again.
  - Debugging and maintenance is easy.

Functions: Example

- Sine Function
  - Sine function is included in the standard library and is declared in math.h
  - The example here uses Taylor series expansion. The successive terms are added till the term becomes smaller than the machine epsilon.

```
#include <limits.h>

double sin(double x)
{
  double sinValue = 0.0;
  double xSquared;
  double term;
  int n = 1;

  xSquared = x * x;
  term = x;

  while ( term > DBL_EPSILON ) {
    sinValue += term;
    n += 2;
    term *= (- xSquared)/(n - 1)/n;
  }

  return sinValue;
}
```

Functions: Example

- Prime Numbers
  - Print a table as follows

```
1*   2*   3*   4    5* … 10
11*  12   13*  14   15 … 20
```

upto 100. All primes are starred.

  - The main function prints the nice looking table. And it is not about finding primes.
  - The functinon isPrime(n) is expected to return 1(true) if n is a prime and 0(false) otherwise.

```
#include <stdio.h>

main()
{
  int i;

  for( i = 1; i <= 100; i++ ) {
    prinf("%d", i);
    if ( isPrime(i) )
      printf("*");
    if ( i % 10 == 0 )
      printf("\n");
    else
      printf("\t");
  }
}
```

Functions: Example

- Prime Numbers
  - A rather simple implementation of isPrime function is given here.
  - The function has same structure as main function we saw before, except that int before the name of the function and return statement.

```
int isPrime(int num)
{
  int    i;

  if ( num < 3 ) return 1;

  for (i=2; i<=(int) sqrt(num);i++ )
    if ( num % i == 0 ) break;

   if ( num % i ) return 1;

   return 0;
}
```

Functions

- The syntax of a function is

```
return-type function-name(argument-list)
{
    Declarations and statements
}
```

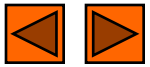
- Smallest function is

```
emptyFunction() {}
```

- Rules for the function name are same as that of the variables.
- Return-type must be one of valid data-types or `void`. If it is not mentioned then `int` is assumed. It is not mandatory to return a value. The calling program is free to ignore the return value.
- The form of argument list is

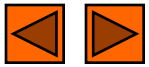
```
type1 arg1, type2 arg2,..., typen argn
```

- it is understood that the arguments have been declared.



Arguments

- The term argument (actual argument, actual parameter) is used for the variables that are passed to the function by a calling program. The term parameter is used for the variables received by the function as described in the function declaration.
- The arguments are passed by value. This means a separate copy of the variable is passed to the function. The values of the variables in the calling program are not affected by the changes to the parameters in the function definition.
- The types of the arguments passed and the parameters declared must match. If they don't match, automatic type casting is applied. If these still donnot match, there is an error.



Arguments

This program will output three lines:

```
4
5
4
```

```
#include <stdio.h>

newI (int I) {
   I = 5;
   printf("%d\n", I);
}

main() {
   int  I;

   I = 4;
   printf("%d\n", I);

   newI(I);
   printf("%d\n", I);

}
```