Functions in C

Function Definition

```
return-value-type function_name (paramter-list) {
        declarations
        statements
}
```

- Function name is an **identifier**.
- Return type is any valid C type or `void`.
- Parameters to function specified by a comma separated list.
- The declaration of a function which excludes its body is called a **function prototype**

Functions in C

Return from Function

- Only one function with a given name is allowed
- A function returns (control restored back to caller) when either
  - A `return` is encounter during execution of the function code, or
  - The control reaches end of the function body

Argument Conversion

- C allows function calls even if argument types do n't match with corresponding parameter types.
- The rules for type conversion of arguments depends on whether function prototype is encountered not prior to call or not.

Functions in C

Argument Conversion

Prototype seen prior to call

- Each argument implicitly converted into type of corresponding parameter as if by **assignment**
- Eg., `int` type converted to `double` if function expects `double`.

Prototype not seen priort to call

- Default argument promotion is performed.
- `float` coverted to `double`
- Integral promotion are performed, causing `short` or `char` converted to `int`

Functions in C

- Relying on default promotion is dangerous.

```
#include <stdio.h>
int main() {
    double x = 5.0;

    /* No protype seen. Not known if int is expected. *
     * Default promotion to double is of no effect    *
     * Result is undefined                            */

       printf("squre␣of␣%.5f␣=␣%.5f\n", x, square(x));
}
int square(int n) {
    return n * n
}
```

Functions in C

Without Parameters & Return Values

```c
#include <stdio.h>
void factorial() {
   int n, result = 1;
   printf("Enter n: ");
   scanf("%d", &n);
   printf("%d! = ", n);
   while ( n > 0 ) {
      result = result * n;
      n--;
   }
   printf("%d\n", result);
}

int main() {
   factorial();
}
```

Functions in C

Void Type

- `Void` is not a type.
- Syntactically it is used where a **type name** is expected.
- When used for `parameter-list`: function does require any parameters.
  - Eg., `int function(void)` is same as `int function()`.
- If return type not mentioned function always returns `int`, i.e. `function()` same as `int function()`

Functions in C

Void Type

- Void type introduced so that compiler can issue a warning when a function does not return an integer type if that is supposed to return one.
- A variable declared `void` is useless.
- But, `void *` defines generic pointer. (Any pointer can be cast to void pointer and back without loss of information)

Functions in C

Function with a Return Value

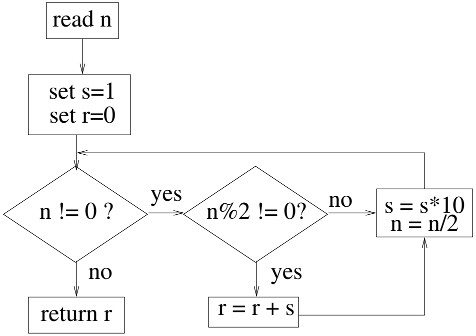
```
#include <stdio.h>
int factorial(int n) {
    int result = 1;
    while ( n > 0 ) {
        result = result * n;
        n--;
    }
}

int main() {
    int n;
    printf("Enter n: ");
    scanf("%d", &n);
    printf("%d! = %d\n", factorial(n));
}
```

Functions in C

Function with a Return Value

Problem: convert a decimal number n to its binary equivalent n_b.

- Partition n into sum of powers of 2, eg., $67 = 2^6 + 2^1 + 2^0$.
- Binary equivalent of 67 will have 1's in 5th, 1st and 0th bit positions.



Functions in C

Decimal to Binary

```
#include <stdio.h>
int binaryNumber(int n) {
   int r = 0, s =1;

   while (n!=0) {
      if (n % 2 != 0)
          r = r + s;
      s=s*10;
      n=n/2;
  }
  return r;
}

int main() {
   printf("Binary␣of␣%d␣is␣%d\n", 58, binaryNumber(58));
}
```