Functions

Modular Programming with
Modularity

- How do you solve a complex problem? *very carefully*
- Divide it into small tasks and try to solve each task and then combine them
- In C we use functions also referred to as *modules* to perform the tasks we determined in our solution
- So, a module is a set of statements that performs a task or computes a value
Advantages of using modules

- Promotes the concept of abstraction
- Readable
- Reduces length of programs, making it more
- Large projects can be developed in parallel
- Modules can be reused
- Modules can be written and tested separately
Sequence of steps in the program

NOTE: It does NOT indicate the tasks.

tasks and which tasks reference other shows how the program separated into

Structure Charts
Structure Charts (cont'd)
Functions

- Change the value of the function arguments (call by reference)
- Return a single value to the calling function
- Take arguments and perform a specific task

Functions

- Programmer-defined functions (e.g., myPrint, area)
- Pre-defined library functions (e.g., printf, sin, tan)

Functions could be

- Program encounters function names
- Additional functions are called or invoked when the
  function starts with main()
```c
{ return 0;
printf("The sine of the angle is \%f\n", sin(\angle));
scanf("\%f", \angle);
printf("Input angle in radians: \n");
double angle;
}

int main(void)
{
#include <math.h>
#include <stdio.h>
So far, we used several pre-defined functions!

Pre-defined Functions Example
```
Programmer-defined Functions Example
But this might cause some errors (double -> int) so you need to be careful.

- If the type is not the same, type conversion will be applied (conversion or arguments).
  - Type
  - Formal parameters must match with actual parameters in order, number and data
  - Actual parameter
  - Formal parameters

Function Prototype describes how a function is called

```
result = function_name(5, distance, x);
```

Function Call

```
the function_name(int a, double b, int c);
```

(Function Prototype-declarations)

```
{ statements; declarations; }
```

(Function definition)

```
return type function_name (parameter-declarations)
```

Programmer-Defined Functions Terminology
Value Returning Functions

- Function *returns* a single value to the calling program.
- A *return* expression; statement is *required* in the function definition.
- The value returned by a function can be assigned to a variable, printed, or used in an expression.
A void function does not return a value to the calling program.

- If a return statement is used (no return value), a program may be called to perform input and output.
- Modify data.
- Perform a particular task (clear the screen).
Example - factorial function

```c
int factorial(int n) {
    int facts = 1;
    while(n > 1) {
        facts *= n;
        n--;
    }
    return facts;
}
```


```c
#include "stdio.h"

int main(void)
{
    int n;

    printf("Enter a positive integer\n");

    scanf("%d", &n);

    printf("\n\nFactors: \n\nreturn 0;
    printf("\n\nif(n<=0) exit(0);

    scanf("%d", &n);

    printf("Enter a positive integer\n");

    return(nfactres);

    {
    --n;

    if(n>0)
    {
        int factors = 1;

        while(n>1)
        {
            int factres = 1;

            (int (int n))
        }
    }
}

return 0;
```

```c
#include <stdio.h>

int main()
{
    int n, factorial = 1, n_factorial;
    printf("Enter positive integer\n");
    scanf("%d", &n);

    n_factorial = factorial = 1;
    if (n < 0)
        return 0;

    for (n_factorial = 1; n_factorial <= n; n_factorial++)
        factorial *= n_factorial;

    printf("%d! = %d\n", n, factorial);
    return 0;
}
```

Variable, printed, or used in an expression can be assigned to a

The value returned by a function can be assigned to a
Example – use fact()
Example

```java
int n = 5;
int t = 5;
factres = 1;

while (n > 1) {
    t = fact (int n);
    factres = factres * t;
    n--;}

return (factres);
```
Example

```
int fact(int n)
{
    int factres = 1;
    while(n>1)
    {
        factres = factres * n;
        n--;
    }
    return (factres);
}
```
```c
#include <stdio.h>

int fact(int n) {
    int s, t = 5;
    s = 120 + fact(t+1);
    printf("Result is %d\n", s);
    return 0;
}

int main(void) {
    printf("\nExample\n");
    return 0;
}
```
```java
public int fact(int n)
{
    return (n <= 1) ? 1 : n * fact(n - 1);
}
```

Example
Example
Example
\[
\gamma = \frac{p - i\lambda}{\zeta \cdot i\zeta + i\lambda} = \lambda
\]

Write a statement to compute.
\[
\int \text{select} (\int, n, \int, k) = \binom{n}{k}
\]

Write a select function that takes \( n \) and \( k \) and computes \( \int \text{select} \).
```c
{
    int maximum;
    maximum = a;
    else
    maximum = b;
    if (a < b)
        int maximum;
    } int max(int a, int b)
```

Write a function to compute maximum of two numbers.

---

**Exercise**
Exercise


```c
{ 
    temp = max(x, max(y, z));
    temp = max(4, 4+3*2);
    temp = max(4, 6);
    temp = max(x, y);
    int x = 2, y = 3, z = 7, temp;
}

int main() {
    Are following calls to max function valid?
    
    Exercise
    
```
Exercise

Write a function that takes score as input and computes and returns letter grade based on the scale below.

- 80-100: A
- 60-79: B
- 40-59: C
- 0-39: D
Solution

```c
char get_grade(int score)
{
    int grade = 'A';
    if (score >= 80) grade = 'B';
    else if (score >= 70) grade = 'C';
    else if (score >= 60) grade = 'D';
    else if (score >= 39) grade = 'E';
    return grade;
}
```
What is the output of the following program?

Exercise
What is the output of the following program?

Exercise
If a return statement is used (no return value)

A program

A void function does not return a value to the calling

perform input and output

modify data

perform a particular task (clear the screen)

A void function may be called to

Void Functions