Functions

“Organizational unit”

organizational unit of a program: function

- We break up a program into functions so that (i.e. ***Modular Programming***)
  - different people can write different parts
  - by putting a piece of a code into a function, we give it a name, and state what it takes as input and what it generates.
  - This makes the overall program easier to understand.

Elements of user defined functions

1. Function declaration / prototype
2. Function definition
3. Function call

1. A **function declaration/prototype** is essentially the definition without the body.

   – The return type, name, parameter types and optionally parameter names.

   – Example: declaration of gcd function:

```
int gcd(int m, int n);
int gcd(int, int);    // also acceptable.
```

Elements of user defined functions

2. Function Definition (Function header & function body):

```
function-type    function-name   ( parameter list)
{
        local var declaration;
        .....
}
```

Types of Functions in C

1. Function No Arguments & No Return Value
2. Function Arguments & No Return Value
3. Function No Arguments & Return Value
4. Function Arguments & Return Value

1. Function No Arguments & No Return Value

e.g.

```
#include<stdio.h>
void fun1(void);                    //function prototype
int main()
{
        fun1();                     //function call
}
void fun1(void)                             //function definition
{
int i;
        for(i=0;i<10;i++)
                printf(" * ");
}
```

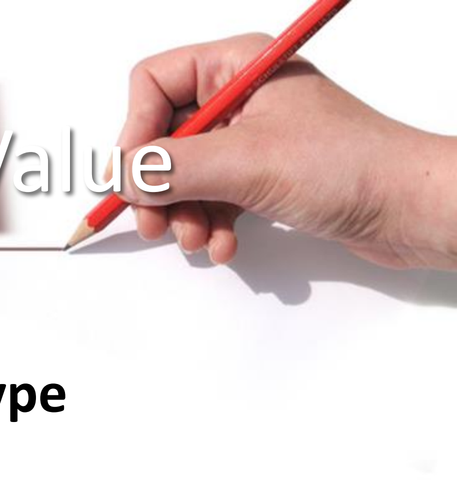
2. Function Arguments & No Return Value

```
#include<stdio.h>
void fun1(int,int);                    //function prototype
int main()
{
int x,y;
        fun1(x,y);                     //function call
}
void fun1(int a,int b)                 //function definition
{
int r;
        r=a+b;
        printf("%d",r);
}
```

3. Function No Arguments & Return Value

```
#include<stdio.h>
int fun1(void);                              //function prototype
int main()
{
int x;
          x = fun1();                       //function call
}
int fun1(void)                              //function definition
{
int i=5;
          i=i*10;
          return (i);
}
```

4. Function Arguments & Return Value



```
#include<stdio.h>
int add (int,int);                    //function prototype
int main()
{
int x=10,y=5;
        r = add(x,y);                   //function call
        printf("%d",r);
}
int add(int a,int b)                    //function definition
{
int r;
        r=a+b;
        return(r);
}
```

Variable Storage classes

Each var has a data type and storage class in C language

1. Automatic var (Local / internal var)
2. External var (Global var)
3. Static var
4. Register var

Local-global var

```
#include<stdio.h>

int a;
int b;
int r;
int main()
{
int add();

        a=10;
        b=20;
        add();
        printf("%d",r);
}
```

```
Void add()
{
f1();

        r=a+b;

}

void f1()
{
        printf("%d %d",a,b);
}
```

Register variables

- We can tell compiler that a var should be kept in one of the machine’s registers for faster execution.

- Syntax:

```
register int count;
```

- Though, there is an upper limit for no. of registers. Once reached, it is automatically treated as non-registered var.

Static Variable

Static var is initialized only once when program is compiled

```
void start(void);
int main()
{
int i;

        for( i=1; i<=5; i++ )
        {
                start ( );
        }
}
```

```
void start(void)
{
static int r=0;

        r=r+1;
        printf("%d",r);
}
```

Static Variable

Static var is initialized only once when program is compiled

```
void sum();
int main()
{
int c;
do
{
        printf("1: add 2:exit“);
        scanf(“%d”,&c);
        switch(c)
        {
                case 1: sum();
                break;
                case 2: exit(0);
        }
}while(1);
```

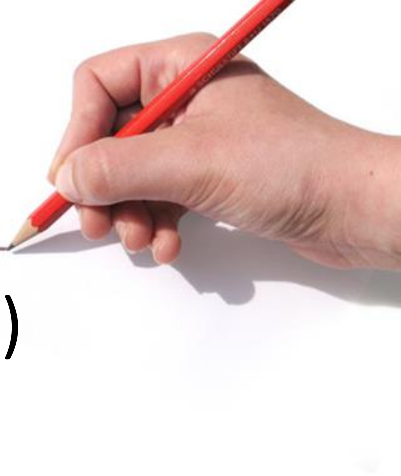
```
void sum()
{
static int s=0;
int n;

printf("enter num \n”);
scanf(“%d”,&n);

s = s+n;

printf("Current sum = %d“,s);
}
```

Passing 1-D array to a function

```
#include<stdio.h>
const int S=5;
void disp(int[]);
main_program
{
int a[S];

for(int i=0;i<S;i++)
    scanf(“%d”,&a[i]);

    disp(a);
}
```

```
void disp(int a[])
{
for(int i=0;i<S;i++)
    printf(“%d”,a[i]);

}
```

Passing 2-D array to a function



```
#include<stdio.h>

const int S=5;
void disp(int[][2]);

int main()
{
int a[S][2];

for(int i=0;i<S;i++)
for(int j=0;j<2;j++)
   scanf(“%d”,&a[i][j]);

   disp(a);
}
```

```
void disp ( int a[ ][2] )
{

for(int i=0;i<S;i++)
   for(int j=0;j<2;j++)
         printf(“%d”,a[i][j]);


}
```

Recursion

What is recursion?

- Methods can call other methods
- Can a method call itself?
- Yes! This is called a recursive method (function)
- “A method within a method”

A recursive method is any method that calls itself

```
#include<stdio.h>
int a=10;
int main()
{
     a=a+1;
    printf("%d ",a);
    main();
}
return 0;
}
```

// Program would enter in infinite loop

```
#include<stdio.h>
int a=10;
int main()
{
if(a<15)
{
    a=a+1;
    printf("%d ",a);
    main();

}
return 0;
}
```

//so, we need a termination condition

Calculate Sum using Recursion

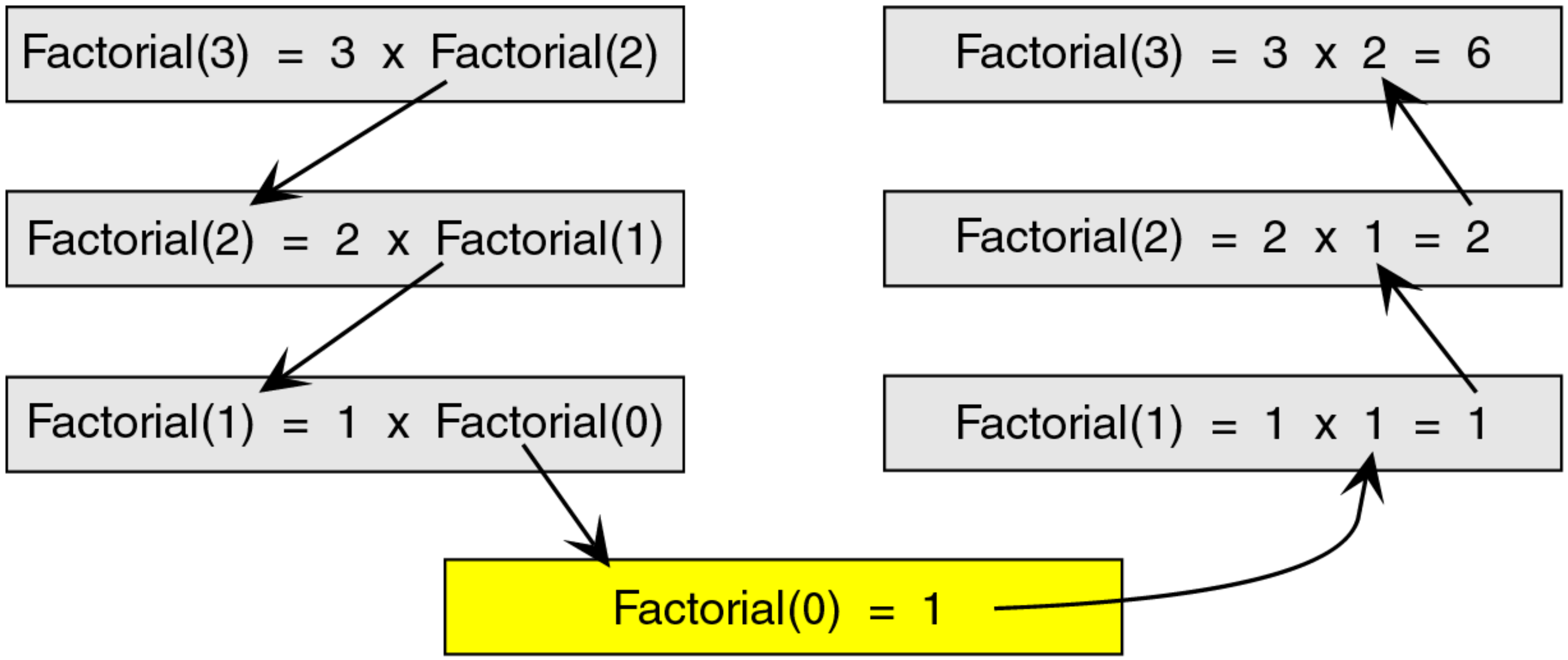
```
#include <stdio.h>
int  sum(int n)
{
   if (n==0) return (0);
   else
   return (n + sum(n-1));
}
int main()
{
int a,res;
          printf( "Enter number:");
          scanf("%d",&a);
          res = sum(a);
          printf(“Sum =  %d", res);
}
```

Factorial of a Number using Recursion

```
#include <stdio.h>
int fact(int n)
{
    if (n==0) return 1;
    else
    return n*fact(n-1);
}
int main()
{
int a,res;
printf( "Enter number:");
scanf("%d",&a);
res=fact(a);
printf("Factorial : %d", res);
}
```

Activation frames keep getting created in memory

- If a=3


Factorial(3) = 3 x Factorial(2)
Factorial(2) = 2 x Factorial(1)
Factorial(1) = 1 x Factorial(0)
Factorial(0) = 1
Factorial(1) = 1 x 1 = 1
Factorial(2) = 2 x 1 = 2
Factorial(3) = 3 x 2 = 6

THANK YOU