

Introduction to Programming

Recursive functions

- ▶ **call itself** within that function
- ▶ recursive function must have the following type of statements
 - a statement to test and determine whether the function is calling itself again.
 - a statement that calls the function itself and must be argument.
 - a conditional statement (if-else)
 - a **return** statement.
- ▶ Example: factorial of a number.

Exercise

- ▶ Write a recursive function which calculates the n factorial.

Functions

`/*iterative solution*/`

```
int fact(int n)
{
    int f=1, i;
    for (i=1; i<=n; i++)
        f=f*i;
    return f;
}
```

`/*recursive solution*/`

```
int fact(int n)
{
    if (n==1)
        return 1;
    else
        return n * fact(n-1);
}
```

```
int fact(int n)
{
    return n>1? n*fact(n-1) : 1;
}
```

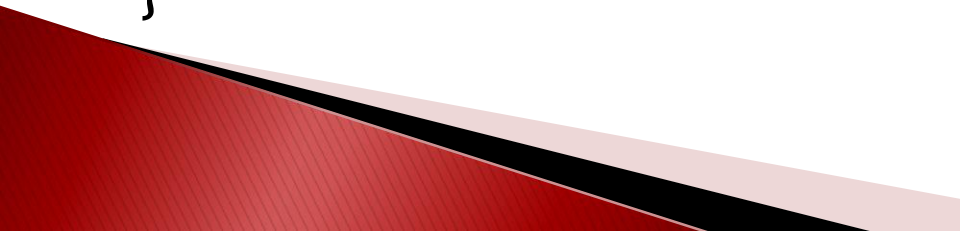
Exercise

- ▶ Write a recursive function which calculates the sum of the first n number.

Solutions

```
int sum(int n)
{
    if (n==1)
        return 1;
    else
        return n + sum(n-1);
}
```

```
int sum(int n)
{
    return n>1 ? n+sum(n-1) : 1;
}
```



Exercise

- ▶ Write a recursive function which calculates the following sum:

$$\sum_{i=1}^n i^2$$

Solution

`/*iterative solution */`

```
int sum1(int n)  
{  
int i, sum=0;  
for (i=1;i<=n;i++)  
    sum=sum+i*i;  
  
return sum;  
}
```

`/*recursive solution */`

```
int sum1(int n)  
{  
if (n==1)  
    return 1;  
else  
    return n*n+sum1(n-1);  
}  
  
int sum1(int n)  
{  
    return n>1 ? n*n+sum1(n-1) : 1;  
}
```


Exercise

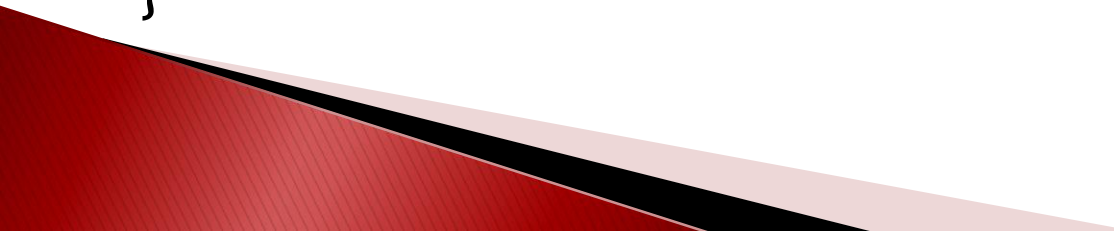
- ▶ Write a recursive function which calculates the following sum:

$$\sum_{i=1}^n i \cdot (i + 1)$$

Solutions

```
int sum2(int n)
{
    if (n==1)
        return 2;
    else
        return n*(n+1)+sum2(n-1);
}
```

```
int sum2(int n)
{
    return n>1 ? n*(n+1)+sum2(n-1) : 2;
}
```



Exercise

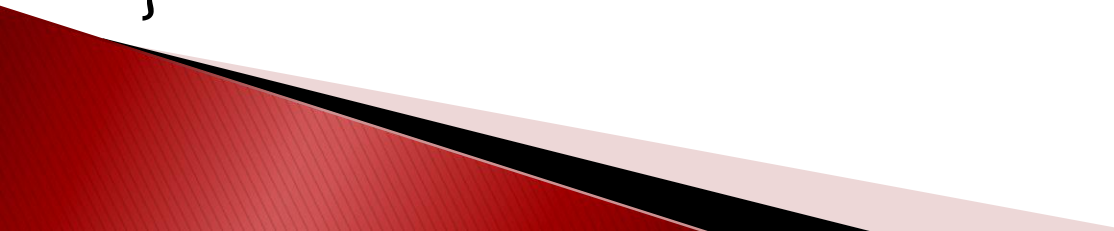
- ▶ Write a recursive function which calculates the following sum:

$$\sum_{i=3}^n (i^3 - 5)$$

Solutions

```
int sum3(int n)
{
    if (n==3)
        return 22;
    else
        return n*n*n-5+sum3(n-1);
}
```

```
int sum3(int n)
{
    return n>3 ? n*n*n-5+sum3(n-1) : 22;
}
```



Exercise

- ▶ Write a function which returns the n^{th} Fibonacci numbers.

$$f(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ f(n-1) + f(n-2), & n > 1 \end{cases}$$

Solutions

*/*iterative*/*

```
int fibonacci (int n)
{
    int i, f0=0, f1=1, f2;
    for (i=2;i<=n;i++)
    {
        f2=f1+f0;
        f0=f1;
        f1=f2;
    }
    return f2;
}
```

*/*recursive*/*

```
int fibonacci (int n)
{
    if (n==0)
        return 0;
    else if (n==1)
        return 1;
    else
        return fibonacci(n-1) + fibonacci(n-2);
}
```

Pointers

- ▶ address of the variable

Definition:

```
type *pointer;
```

Example:

- `int i, *pi;`
- `float f, *pf;`

Reference

With **&** operator we can refer to the address of the variable.

Using the ***** operator we can refer to the variable value.

Syntax:

- **&**variable

Example:

- `int a , *p;`
- `p = &a;`

We say that the **p** pointer points at the **a** variable.

Exercise

What is the result after the execution of this code?

```
int a = 5, *p;  
p=&a;  
*p=a+*p-2;  
printf("%d\n",a);
```

Exercise

What is the result after the execution of this code?

```
int k = 4, j=10, *p, *q;  
p=&k; q=&j;  
*p+=*p+55+k-*q;  
printf("%d %d\n",k,*p+*q);
```

Exercise

What is the result after the execution of this code?

```
int a=10, b=20, *p, *q;  
p=&a; q=&b;  
*p +=(*q)++-15;  
printf("%d %d\n", a,*q);  
*q+=*p;  
printf("%d %d\n", a,*q);
```

Pointers and arrays

`int a[10];`

`a=&a[0]` – 0. element address

`a+1=&a[1]` – 1. element address

`a+i=&a[i]` – i. element address

`*a=a[0]` – 0. element

`*(a+1)=a[1]` – 1. element

`*(a+i)=a[i]` – i. element

`a++;` – point to the next element

`*(a+i)++;` – increases the value of i. element with 1

Exercise

What is the result after the execution of this code?

```
int A[45], i;  
for (i=0; i < 45; ++i )  
    A[i] = 2*(i-3);  
printf("%d  %d\n", *(A+25), A[i-3]);
```

Exercise

What is the result after the execution of this code?

```
int B[25], i;  
for (i=0; i < 20; ++i )  
    B[i] = 4*(2*i+1);  
printf("%d  %d\n", *(B+23), B[i-7]);
```

Passing one-dimensional array to function

```
int function_name1(int a[], int n)
```

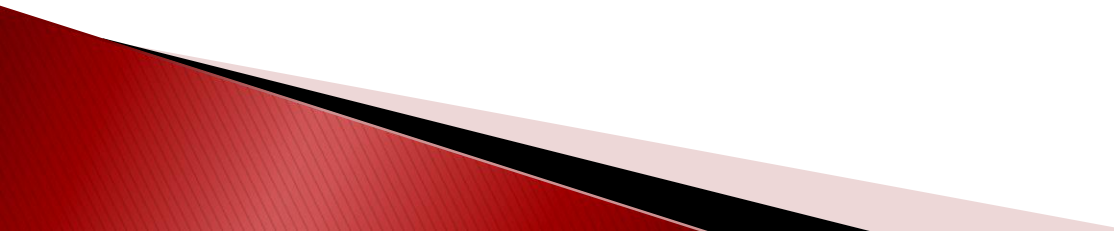
```
int function_name2(int *a, int n)
```

```
    a[i] -> *(a+i)
```

- ▶ Write a **procedure** which displays the elements of the one-dimensional array.
- ▶ Write a **function** which returns the average of the elements of the array.
- ▶ Write a **function** which returns the maximum element of the array.

Solution

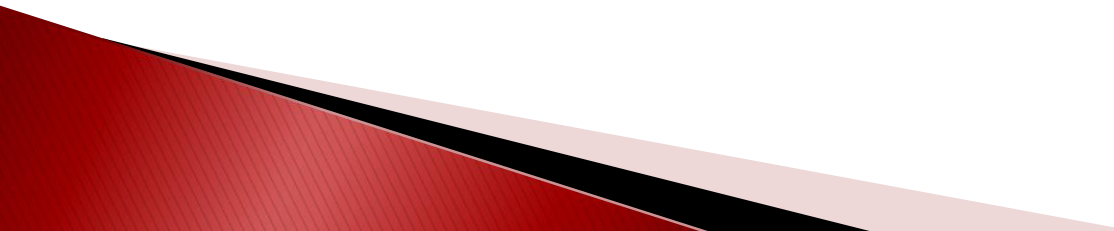
```
void print_out(int a[], int n)
//void print_out(int *a, int n)
{
    int i;
    for (i=0; i<n; i++)
        printf("%d ",a[i]); // printf("%d ", *(a+i));
    printf("\n");
}
```



Solution – Average

```
float average(int a[], int n)
{
    int i;
    float sum=0;
    for (i=0; i<n; i++)
        sum+=a[i]; //sum+=*(a+i);

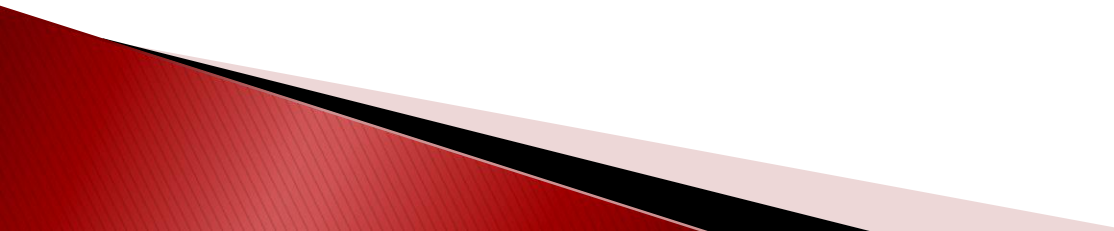
    return sum/n;
}
```



Solution – Maximum value

```
int maximum(int a[], int n)
{
    int i, max=a[0];
    for (i=1; i<n; i++)
        if (a[i]>max)
            max=a[i];

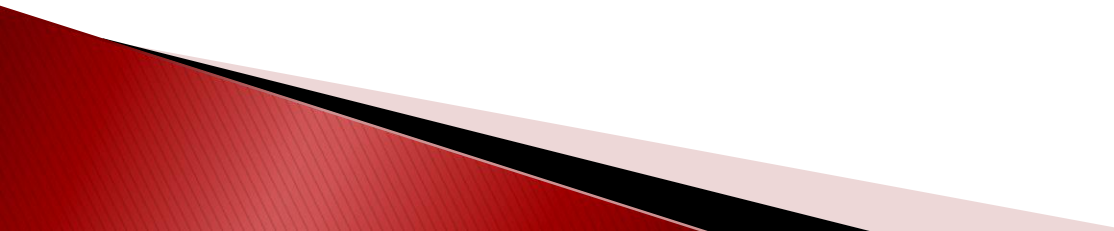
    return max;
}
```



Solution – Maximum index

```
int maximum_index(int a[], int n)
{
    int i, max=0;
    for (i=1; i<n; i++)
        if (a[i]>a[max])
            max=i;

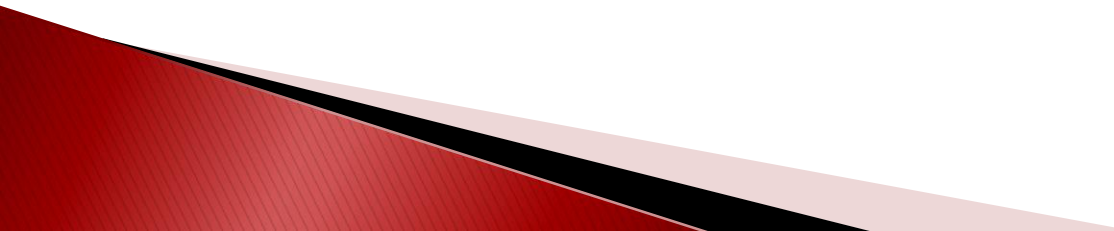
    return max;
}
```



Solution – Minimum value

```
int minimum(int a[], int n)
{
    int i, min=a[0];
    for (i=1; i<n; i++)
        if (a[i]<min)
            min=a[i];

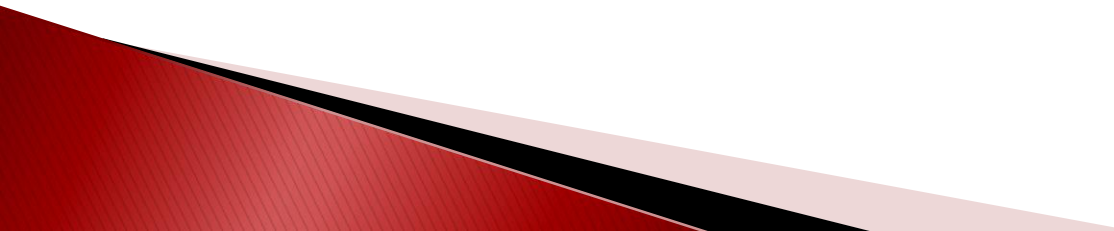
    return min;
}
```



Solution – Minimum index

```
int minimum_index(int a[], int n)
{
    int i, min=0;
    for (i=1; i<n; i++)
        if (a[i]<a[min])
            min=i;

    return min;
}
```



Number of the even elements

```
int even(int *a, int n)
{
    int i, count=0;
    for (i=0; i<n; i++)
        if (a[i]%2==0)
            count++;

    return count;
}
```

