Introduction to Programming

Labor 08

Arrays

- a data structure, which provides the facility to store a collection of data of same type under single variable name
- the size should be an individual constant
- the index specifies the location of the element in the array
- the array index starts from zero
- the maximum index value will be equal to the size of the array minus one

One dimensional array

- declaration form of one-dimensional array is:

data_type array_name[size];

- **int** a[5];
  a[0] = 1; // set first element
  a[4] = 5; // set last element

One dimensional array

Examples:

```
int a[10];
float b[20];
char c[100];
```

- **Initialization**:

```
int a[10]={1,2,3,4,5,6,7,8,9,10};
float f[100]={3.14, -12, 45};
int b[50]={0};
```

Exercise

- Write a program to input **n** different integer numbers into an array, named **a.**
- After all numbers are input, display the numbers.

Solution

- Input/Output one dimensional array

```
int  i, n;
printf(”n=");
scanf("%d", &n);
int a[n];
for (i=0; i<n; i++)
{
    printf(”a[%d]=", i);
    scanf("%d", &a[i]);
}
for (i=0; i<n; i++)
     printf("%d   ", a[i]);
```

Exercise

What is the result of this code?

```
int a[8] = { 12, 24, 11, 7, 4, 13, 18, 52 };
int b[8] = { 2, 44, 21, 17, 24, 3, 38, 11 };
int c[8], d[8], i;

for (i = 0; i < 8; i++)
    c[i] = a[i] + b[i];

for (i = 0; i < 8; i++)
    d[i] = c[i];

for (i = 0; i < 8; i++)
    printf("%d,  ", d[i]);
```

Exercise

What is the result of this code?

```
int B[40],  i;
for (i=0 ; i < 30; i++)
    B[i] = 2*i+2;
printf("%d  %d\n", B[2], B[i-12]);
```

Exercise

What is the result of this code?

```
int B[25],  i;
for ( i=0; i < 15; i++ )
    B[i] = 4*(2*i+1);
printf("%d  %d\n", B[11], B[i-7]);
```

Exercise

What is the result of this code?

```
int B[50],  i;
for (i=0; i < 34; i++ )
    B[i] = 3*i+2;
printf("%d  %d\n", B[44], B[i-12]);
```

Exercise

What is the result of this code?

```
int B[40],  i;
for (i=0; i <= 27; i++ )
    B[i] = 2*i+2;
printf("%d  %d\n", B[7], B[i-18]);
```

Exercise

- Generate randomly **n** natural numbers within a and b interval [a,b].
- Calculate the followings:
  - sum of the elements
  - count the even elements
  - define the minimum element
  - define the index of the minimum element
  - define the maximum element
  - define the index of the maximum element

  - ***Help:*** store the randomly generated numbers in a one dimensional array:
    - srand(time(NULL));
    - array[i]=(rand()%(b-a+1))+a;

Solution

```
int i, n, a, b, c, sum=0, even=0, min=0, max=0;
printf("n="); scanf("%d",&n);
printf("a="); scanf("%d",&a);
printf("b="); scanf("%d",&b);
if (a>b) {c=a; a=b; b=c;}
int array[n];
srand(time(NULL));
for(i=0;i<n;i++)
        {
  array[i]=(rand()%(b-a+1))+a;
        sum=sum+array[i];
        if (array[i]%2==0)
                even++;
        if (array[i]<array[min])
                min=i;
        if (array[i]>array[max])
                max=i;
        }
for(i=0;i<n;i++)
        printf("%d ",array[i]);
printf("\nSum: %d\nEven: %d\nMin value: %d\nMin index: %d\nMax value: %d\nMax index: %d\n",
sum, even, array[min], min, array[max], max);
```

Subroutines in C

- Subroutines are used to create **functional blocks of code** and provide **good program structure.**
- This makes it easier for the program to be understood, allows a block of code to be **reused** and ultimately allows ready-made library routines to be created for future use.
- **Return value**
  - A C subroutine may return one value (**function**), or "**void**" if there is no return value needed (**procedure**).
  - The "**return**" instruction passes the number specified as the subroutine's output and returns to the calling program.
- **Definition/declaration**
  - The subroutine must be declared in the file that uses it BEFORE any code that calls it.

Functions

```
return_type function_name (type1 arg1, type2
    arg2 ,.., typen argn)
{
    local variables;
    statements;
    return expression;
}
```

Functions

```
int add(int x, int y)
{
      int z;
      z=x+y;
      return z;    //return x+y;
}

double average(int x, int y, int z)
{
      double avg;
      avg=(x+y+z)/3.0;
      return avg;
}
```

Example

```
int add(int x, int y)
{
        return x+y;
}
void display(int x, int y, int z).
{
        printf("Sum of %d and %d is %d", x, y, z);
}
int main()
{
        int a,b,c;
        printf("Enter two numbers:\n");
        scanf("%d %d",&a,&b);
        c=add(a,b);
        display(a,b,c);
        return 0;
}
```

Exercise

- Write a function which calculates the N factorial (N!).

Solution

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

Exercise

- Write a function which calculates the sum of the first N number.

Solution

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

Exercise

- Write a **power** function which calculates x^n.

Solution

```
int power(int x, int n)
{
    int i, pow = 1;

    for(i = 1; i <= n; i++)
        pow = pow * x;

    return pow;
}
```

Exercise

- Write a function which determines if the number is prime or not.

Solution

```
#include <stdio.h>
#include <math.h>
int prime(int n)
{
int i;
if (n==0 || n==1)
        return 0;
for(i=2;i<=sqrt(n);i++)
        if (n%i==0)
                return 0;
return 1;
}
```

```
int main()
{
int n;
printf("n="); scanf("%d",&n);
if (prime(n)==0)
        printf("Prime!\n");
else
        printf("Not prime!\n");

return 0;
}
```

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);
  }
```