CS1100 - Lecture 25

Instructor : Jasine Babu

Teaching Assistants : Nikhila K N, Veena Prabhakaran

How to handle arrays?

In the last few classes, we learned how to pass variables as parameters to functions. In this lecture, we will discuss how to pass arrays as parameters to functions, so that the array elements can be modified using the functions. In this lecture we will concentrate on passing one-dimensional arrays as parameters to functions.

Let us first consider a common situation where there is a function which reads in values to the elements of an array, another function which manipulates some array elements and a third function that is used to print the array elements. Consider a sample `main()` function declaration as given below.

```
int main()
{
    int a[20], n;
    printf("enter size of the array (<=20) \n");
    scanf("%d",&n);
    if(n>20)
    {
       printf("wrong input \n");
    }
    else
    {
        input_array(a, n);
        printf("\n array is \n");
        print_array(a, n);
        swap(&a[0],&a[n-1]);
        printf("\n modifiled array is \n");
        print_array(a, n);
    }
    return 0;
}
```

The above `main()` function invokes three functions; `input_array()`, `print_array()` and `swap()`. The function call `input_array(a,n)` is used to read `n` values into the array `a` and the function call `print_array(a,n)` is used to print `n` elements of the array `a`.

In both these function calls, we have passed the name of an array of integers and the length of the array as the parameters. Since the name of an array of integers stands for the address of the first element of the array, the expression `a` is equivalent to the expression `&a[0]`. Since `a[0]` is an integer, `&a[0]` is a pointer to an integer. Since we are passing `a` which is a pointer to an integer as the first parameter and `n` which is an integer as the second parameter while invoking the function calls `print_array(a,n)` and `input_array(a,n)`, the function headers of these functions have to be declared as follows.

```
void input_array(int *, int);
void print_array(int *, int);
```

The function call `swap(&a[0],&a[n-1])` is intended to exchange the values of `a[0]` and `a[n-1]`. Since the invocation of the `swap()` function takes of addresses of two integers as parameters, the declaration of this function should be as follows.

```
void swap(int *, int *);
```

The definition of the function `input_array()` can be done as follows.

```
void input_array(int *x, int k)
/*input n elements to the array with starting address x*/
{
       int i;
       printf("enter array elements \n");
       for(i=0; i < k; i++)
       {
          scanf("%d",&x[i]);
       }

}
```

Similarly, the definition of the function `print_array()` can be done as follows.

```
void print_array(int *x, int k)
/*prints elements of an n element array with starting address x*/
{
     int i;
     for(i=0; i<k; i++)
     {
          printf("%d ",x[i]);

     }
     printf("\n");
}
```

The `swap()` function can be defined exactly the same way as we did it in the last lecture.

```
void swap(int *x, int *y)
{
   int temp;
   temp=*x;
   *x=*y;
   *y=temp;
}
```

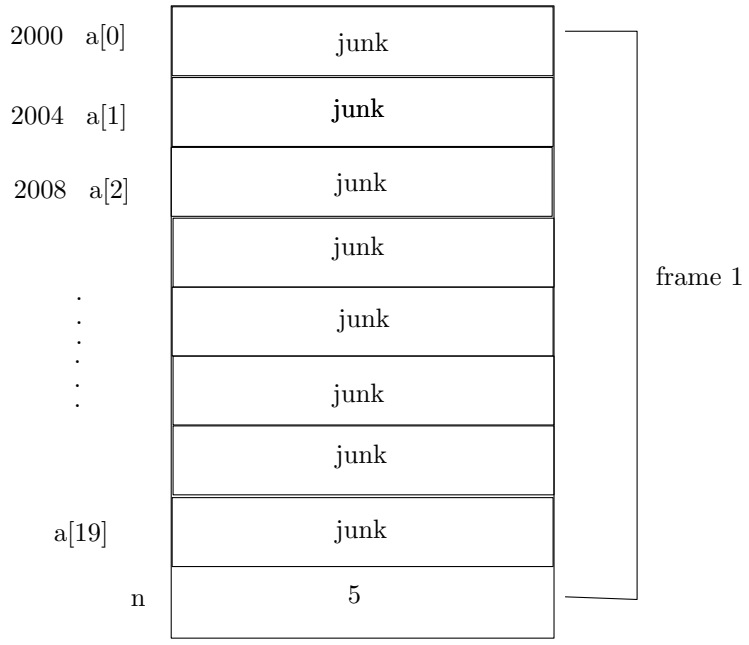
We can combine all the function definitions given above and complete the program as follows.

```
#include <stdio.h>
void input_array(int *, int);
void print_array(int *, int);
void swap(int *, int *);
int main()
{
    int a[20], n;
    printf("enter size of the array (<=20) \n");
    scanf("%d",&n);
    if(n>20)
    {
       printf("wrong input \n");
    }
    else
    {
        input_array(a, n);
        printf("\n array is \n");
        print_array(a, n);
        swap(&a[0],&a[n-1]);
        printf("\n modifiled array is \n");
        print_array(a, n);
    }
    return 0;
}
void input_array(int *x, int k)
/*input n elements to the array with starting address x*/
{
       int i;
       printf("enter array elements \n");
       for(i=0; i < k; i++)
       {
          scanf("%d",&x[i]);
       }
}
```

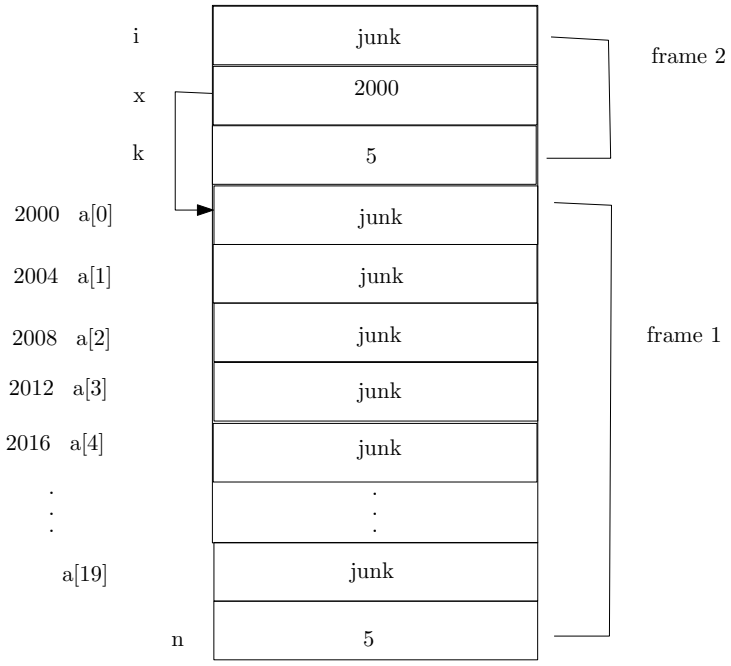
```
void print_array(int *x, int k)
/*prints elements of an n element array with starting address x*/
{
     int i;
     for(i=0; i<k; i++)
     {
          printf("%d ",x[i]);

     }
     printf("\n");
}

void swap(int *x, int *y)
{
   int temp;
   temp=*x;
   *x=*y;
   *y=temp;
}
```

Let us consider the execution of this program with the input values `n=5` and the array `a={10, 9, 8, 7, 6}`.

The following figure shows the memory state diagram just after executing the `scanf` instruction in `main()`.



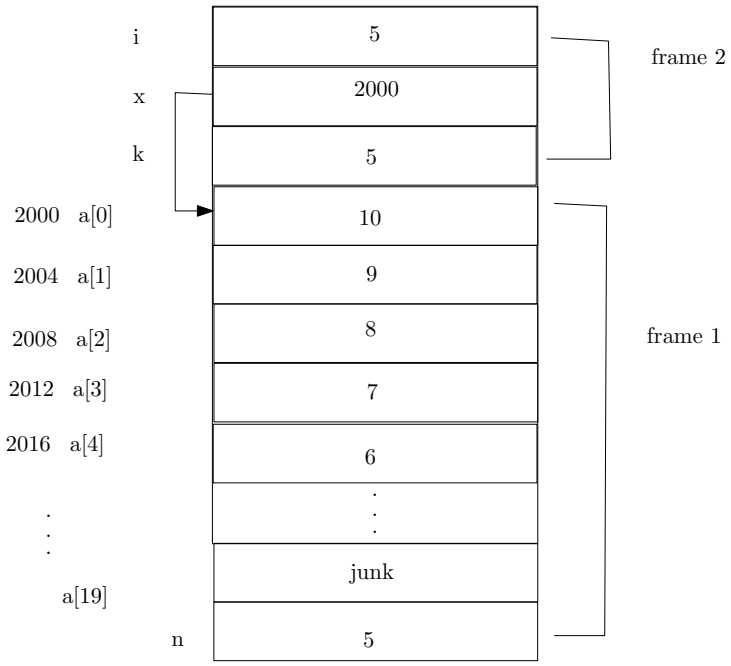
After this, the execution of `main()` continues and when the function `input_array(a,n)` is invoked, a new stack frame is created in memory. The values of the parameters `a` and `n` in the function call are copied to the locations of the formal parameters `x` and `k`. Note that, the value of `a` is `&a[0]` which is `2000` as per the earlier figure and this is the value that is getting copied as the value of the integer pointer `x` in the new frame. The memory state diagram when the function `input_array()` starts to execute is given below.



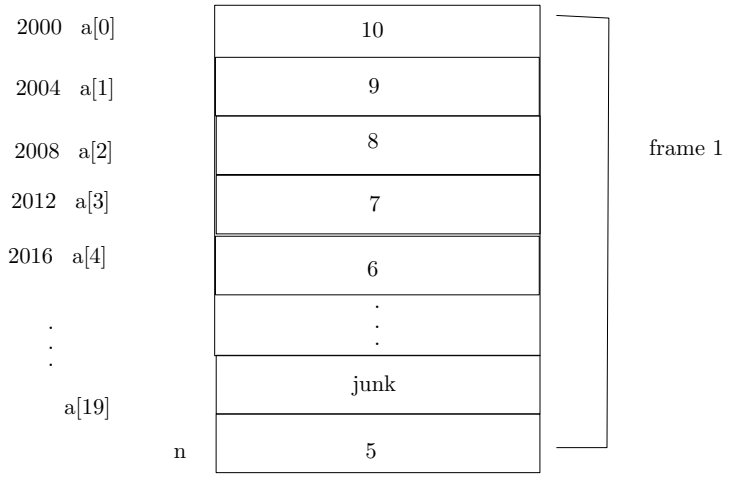
Since the value of `x` is `2000`, `x[0]` (which is the same as `*(x+0)`) refers to the variable stored in location `2000`. In general, `x[i]` (which is same as `*(x+i)`) refers to the variable stored in location `2000+i`$\times$ `size of int`. Note that, this variable is nothing but `a[i]`. Therefore, the expressions `&x[i]` and `&a[i]` denotes the same address.

When the instruction `scanf("%d",&x[i]);` is executed, input value is taken to the location whose address is `&x[i]` which is the same as the location whose address is `&a[i]`. Thus, when this instruction is executed, input value is getting stored in the location of `a[i]`. When the `for` loop repeats for values of `i` from `0` to `k-1` (which is 4), input values {`10, 9, 8, 7, 6`} are respectively getting stored in locations of `a[0]` to `a[4]`.

The contents of the memory location after completing the execution of the `for` loop in the `input_array()` function is as follows.

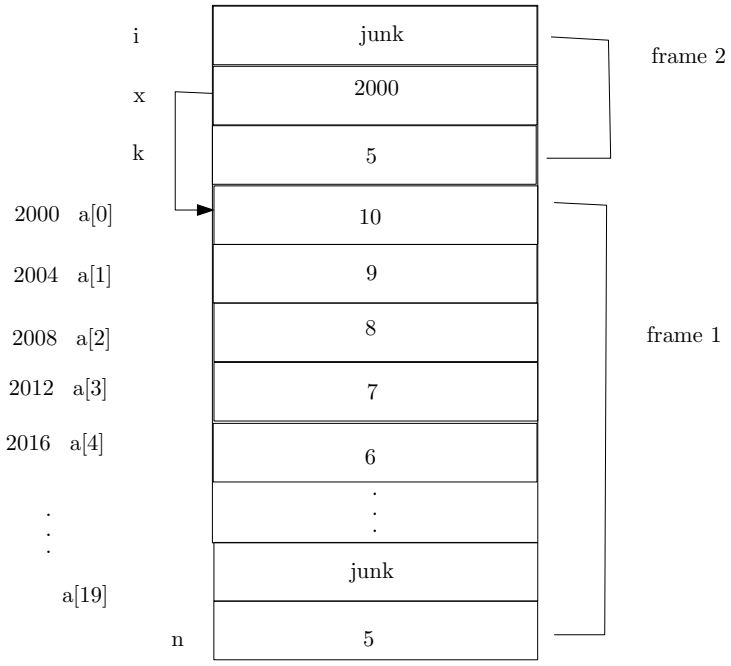


After this, the function `input_array` finishes its execution and program control transfers back to `main()` to the line after the instruction `input_array(a, n);` . The memory state diagram at this point is shown in the figure below.



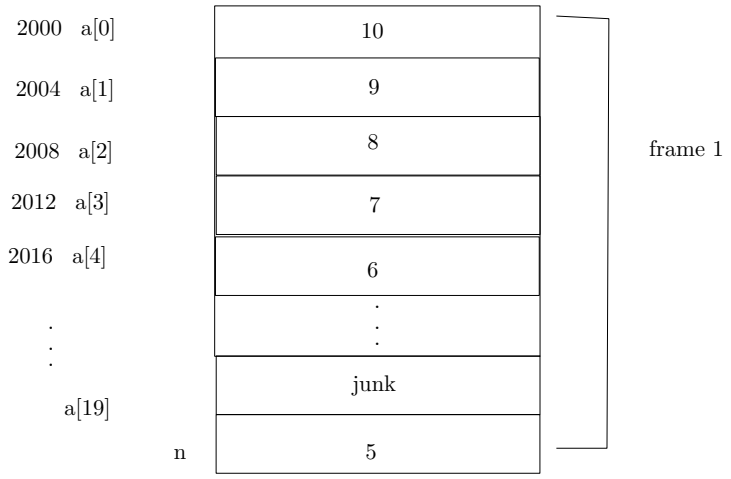
After executing the next `printf` statement, the function `print_array(a,n)` is invoked and again a new stack frame is created in memory. The values of the parameters `a` and `n` in the function call are copied to the locations of the formal parameters `x` and `k` as it was done during the function call `input_array(a,n)`.

The memory state diagram when the function `print_array()` starts to execute is given below.

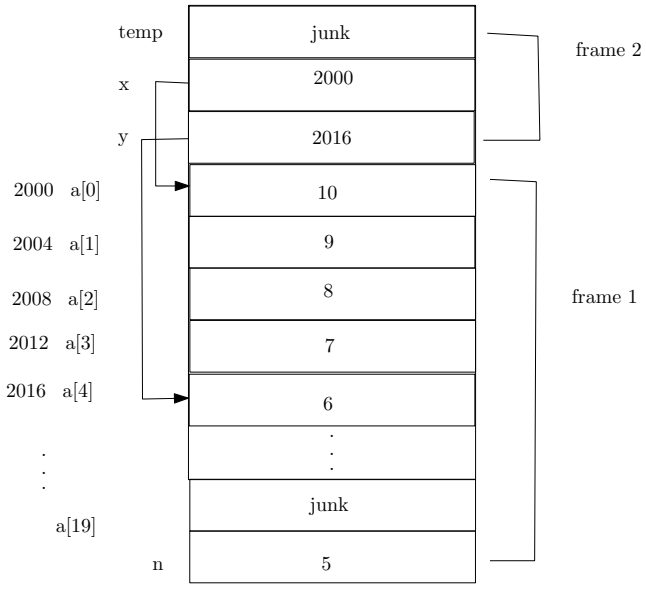


As discussed in the case of the `input_array()` function, the expression `x[i]` denotes the same locations as the `a[i]`. Therefore, when the `for` loop in the `print_array()` function executes for `i=0` to `4`, the values of `a[0]` to `a[4]` are printed. After this, `print_array()` function finishes its execution and the program control returns to the `main()` function.

At this point, the content of the memory location is as shown below.

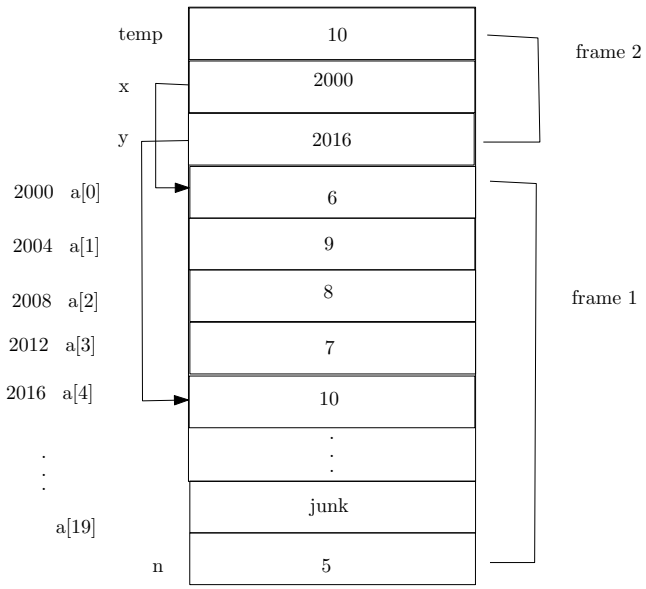


The execution continues in `main()` and the function call `swap(&a[0],&a[n-1])` is invoked. A new stack frame for `swap()` is created and the values of the parameters `&a[0],&a[n-1]` are copied as the values of the pointer variables `x` and `y`. At this point, the contents of memory locations are as shown below.

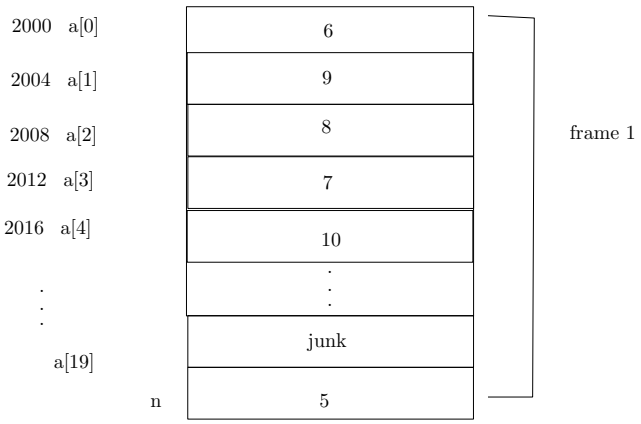


In the previous lecture, we have already seen how the `swap()` function works.

The memory state diagram just before the function `swap()` finishes its execution is shown in the figure below.



The following figure shows the contents of the memory locations after the function `swap()` finishes its execution and the program control returns to `main()` to the next line after the function call `swap()`.



Finally, the `print_array()` function is executed once again as earlier and the modified array elements are displayed. The output of the above program is given below.

```
  array is
10 9 8 7 6
 modifiled array is
6 9 8 7 10
```

Another way of declaring functions with array parameters

Recall that, the functions `input_array()` and `print_array()` in the previous program were declared as:

```
void input_array(int *, int);
void print_array(int *, int);
```

The function headers in their definition were as:

```
void input_array(int *x, int k)
void print_array(int *x, int k)
```

Instead of the above method, there is another equivalent way of declaring and defining these functions.

Declaration

```
void input_array(int [], int);
void print_array(int [], int);
```

The function headers in the definition

```
void input_array(int x[], int k)
void print_array(int x[], int k)
```

This new method of writing works exactly the same way as our earlier method. The entire program can be re-written in the new format as given below.

```
#include <stdio.h>
void input_array(int [], int);
void print_array(int [], int);
void swap(int *, int *);
int main()
{
    int a[20], n;
    printf("enter size of the array (<=20) \n");
    scanf("%d",&n);
    if(n>20)
    {
       printf("wrong input \n");
    }
    else
    {
        input_array(a, n);
        printf("\n array is \n");
        print_array(a, n);
        swap(&a[0],&a[n-1]);
        printf("\n modifiled array is \n");
        print_array(a, n);
    }
    return 0;
}
void input_array(int x[], int k)
/*input n elements to the array with starting address x*/
{
       int i;
       printf("enter array elements \n");
       for(i=0; i < k; i++)
       {
          scanf("%d",&x[i]);
       }
}
void print_array(int x[], int k)
/*prints elements of an n element array with starting address x*/
{
     int i;
     for(i=0; i<n; i++)
     {
          printf("%d ",x[i]);

     }
     printf("\n");
}
```

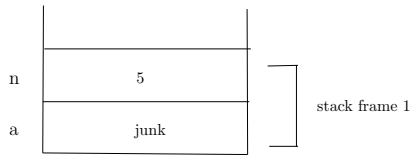
```
void swap(int *x, int *y)
{
   int temp;
   temp=*x;
   *x=*y;
   *y=temp;
}
```

An incorrect way of reading values to an array

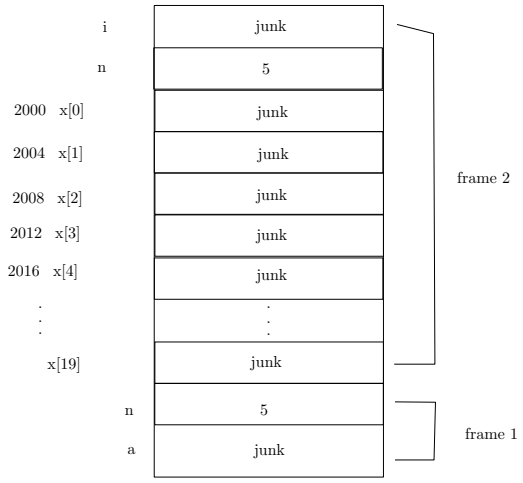
Consider the following program in which the function `input_array()` declares a local variable `int x[20]`, reads the values to elements of the array `x` and returns `x` to the `main()` function and assigns this return value to a variable of type `int *`.

```
#include <stdio.h>
int * input_array(int);
int main()
{
    int *a, n;
    printf("enter size of the array (<=20) \n");
    scanf("%d",&n);
    if(n>20)
    {
       printf("wrong input \n");
    }
    else
    {
        a=input_array(n);
    }
    return 0;
}
int * input_array(int k)
/*input n elements to the array with starting address x, n<=20*/
{
       int x[20],i;
       if(n<=20)
       {
          printf("enter array elements \n");
          for(i=0; i < n; i++)
          {
            scanf("%d",&x[i]);
          }

       }
       return x;
}
```

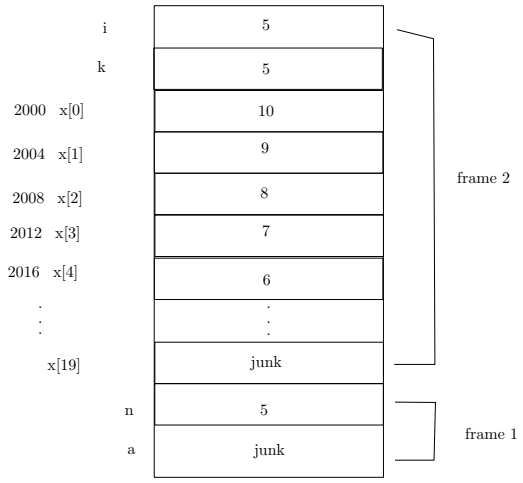
Consider the execution of the above program for `n=5` and `a={10, 9, 8, 7, 6}`. Just before invoking `input_array()` function from `main()` , the memory state diagram is as follows.



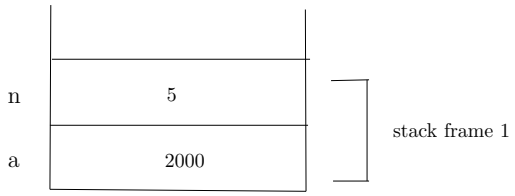
When the function `input_array()` is invoked, a new stack frame is created and it will allocate space for the input array `x`, `n` and `i`. At this point, the contents of the stack frame in the memory is as follows.



After this, values for the array elements are read from the terminal one by one, while executing the `for` loop. The contents of the memory locations just before returning from `input_array()` function is as follows.



When the function `input_array()` finishes execution, the stack frame of `input_array()` is deleted from memory and the return value `x` which is the address of `x[0]`, which is `2000` is returned to `main()` and this return value gets assigned to `a`, because `a` is on the left hand side of the function call `a=input_array(n);`. The following figure shows the contents of the stack frame after finishing the execution of the function `input_array()`.



The problem now is that, the pointer `a` stores the address `2000`, whose contents no longer exist. The location with address `2000` may get allocated for some other purpose and if we try to access any element of the array, say try to display the value of `a[0]`, it might result in a segmentation fault.

Rewriting selection sort algorithm

Recall the sorting algorithm that we discussed in the Lecture 9. Suppose we have `n` numbers for sorting which are stored in an array `a`. The basic algorithm for selection sorting is given below.

```
1. for i=1 to n-1
2. {
3.    find out maxposn, the position of the maximum among a[0] to a[n-i]
4.    swap(a[n-i],a[maxposn])
5. }
```

To implement the selection sort program using functions, we will define a `sort_array()` function, which takes an array of integers `x` and its size `n` as parameters and performs the steps of the algorithm given above. The declaration of this function can be made as follows.

```
void sort_array(int *,int);
```

For defining the `sort_array()` function, we will make use of the following functions:

- For implementing line 3 of the algorithm define a function `getMaxpos()` which takes an array `x` and an index `k` as parameters and returns the position of the maximum element among `x[0]` to `x[k]`.

  The declaration of this function can be made as follows.
  `int getMaxpos(int *,int);`

- For implementing line 4 of the algorithm, we can use the `swap()` function which we have defined earlier.

The function `getMaxpos()` is easy to implement. One way to do this is as follows.

```
int getMaxpos(int *x, int upper)
{
  int i, max_pos=0;
  for(i=0;i<=upper;i++)
  {
    if(x[i] > x[max_pos])
          max_pos=i;
  }
  return max_pos;
}
```

Apart from these functions, we will also need to use the functions `input_array()` and `print_array()` which we have defined earlier.

The complete program for implementing selection sort is given below.

```
#include<stdio.h>
void input_array(int *,int);
void sort_array(int *,int);
int getMaxpos(int *,int);
void swap(int *, int *);
void print_array(int *, int);
int main()
{
  int a[20],n;
  printf("Enter the size of the array (<=20) : ");
  scanf(" %d",&n);
  if(n<=20)
  {
     input_array(a,n);
     sort_array(a,n);
     printf("\n Sorted list is : \n");
     print_array(a,n);
  }
  return (0);
}

void sort_array(int *x, int n)
/* sorting an array of n elements with
starting address x using selection sort*/
{
  int posn, i;
  for(i=1;i<n;i++)
  {
     posn=getMaxpos(x,n-i);
     swap(&x[posn], &x[n-i]);
  }
}
```

```
int getMaxpos(int *x, int upper)
/* returns index of the maximum element among
x[0] to x[upper] */
{
  int i, max_pos=0;
  for(i=1;i<=upper;i++)
  {
    if(x[i] > x[max_pos])
         max_pos=i;
  }
  return max_pos;
}

void swap(int *x, int *y)
{
  int temp;
  temp=*x;
  *x=*y;
  *y=temp;
}

void input_array(int *x, int k)
/*input n elements to the array with starting address x*/
{
       int i;
       printf("enter array elements \n");
       for(i=0; i < k; i++)
       {
          scanf("%d",&x[i]);
       }
}

void print_array(int *x, int n)
/*prints elements of an n element array with starting address x*/
{
     int i;
     for(i=0; i<n; i++)
     {
          printf("%d ",x[i]);
     }
     printf("\n");
}
```