CS1100 - Lecture 9

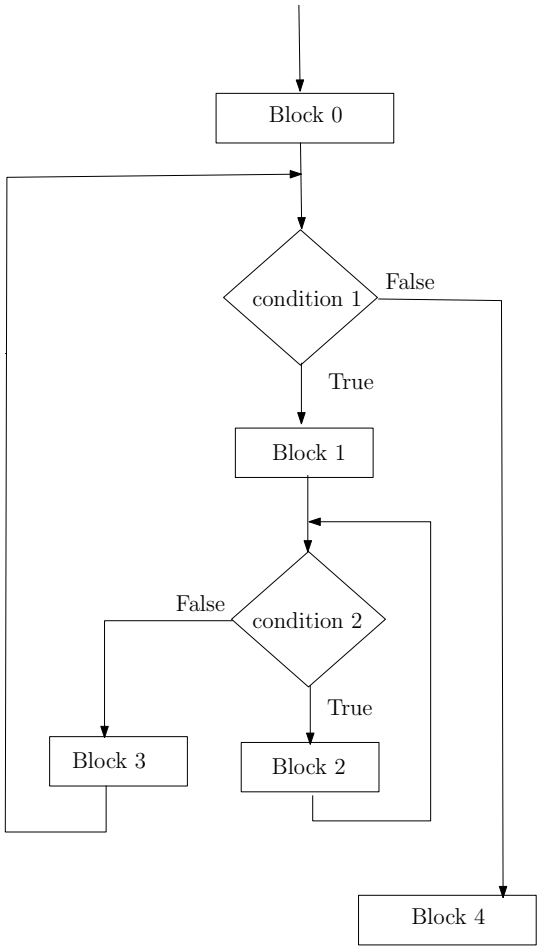
Instructor : Jasine Babu

Teaching Assistants : Nikhila K N, Veena Prabhakaran

Nested Loops

We have already seen some example programs, that uses nested loops in the lab. Nested loops are those in which, one loop appears within another. The genral structure of a nested `while` loop and its corresponding control flow is shown in the figure below. In this figure, we assume that there are no `break` or `continue` statements in any of the program blocks.

```
Block 0
while( condition 1)
{
      Block 1
      while( condition 2)
      {
          Block 2
      }
      Block 3
}
Block 4
```



If `condition 1` evaluates to true, the control will enter the outer loop and start executing Block 1. After executing Block 1, `condition 2` is evaluated. If `condition 2` is true, Block 2 is executed and control comes back to checking `condition 2` again. This is repeated until `condition 2` becomes false when it is getting evaluated while executing the `while(condition 2)` instruction. When `condition 2` becomes false at the time of execution of `while(condition 2)` instruction program control is transferred to Block 3.

After executing the Block 3 program control will go back to `condition 1`. When the execution of Block 3 is completed, one iteration of the outer loop and some k iterations of the inner loop is completed. Again the program control go back to the `condition 1` and the whole process starts repeating. This is done until `condition 1` becomes false when it is getting evaluated while executing the `while(condition 1)` instruction. When `condition 1` becomes false at the time of execution of `while(condition 1)` instruction program control is transferred to Block 4.

Selection Sort

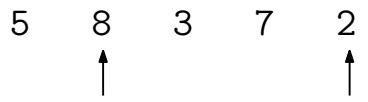
Let us learn an algorithm for sorting a list of `n` numbers to understand nested while loops. Sorting means arranging the given set of numbers in ascending (or descending) order.

In Exercise Problems 4, Question no. 3, an outline of a program for selecting the maximum element in a list of `n` numbers and exchanging it with the element in the last position was given. Let us call this as a *Selection* step. In the sorting program we are going to describe, we will repeatedly make use of the *Selection* step. Our program has many rounds.

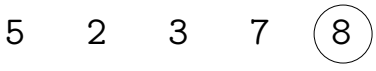
Round 0: Find the maximum among the all numbers in the list and exchange it with the last element.
Round 1: This round operates on the resultant list of the round 0. Find the maximum number among all numbers in the list except last element. Exchange this element with the element in second last position.
Round i: This round operates on the resultant list of the i-1 round. Find the maximum number among the first `n-i` numbers in the list. Exchange this element with the $(n-i)^{th}$ element in the list. Notice that, after the i^{th} round, the largest `i+1` elements in the list are arranged in proper order and they are at the end of the list. After `n-2` rounds, the largest `n-1` elements in the list are arranged in proper order and they are at the end of the list. But this means that, the first element is already the smallest and thus the list has become sorted. Altogether we had `n-1` rounds (round 0 to round `n-2`).

Let us consider an example. Suppose n=5, and the list to sort is 5, 8, 3, 7, 2.

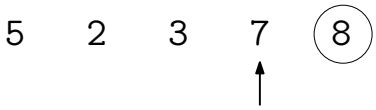
In round 0, the positions of maximum number and last number are shown below.



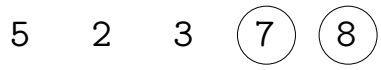
After round 0, the result is as follows.



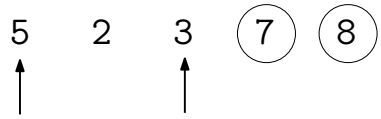
In round 1, find the maximum among 5, 2, 3, 7. The maximum number 7 is already in second last position in our list. Therefore, the list remains the same after round 1.



After these two rounds, the last two elements are in proper order.

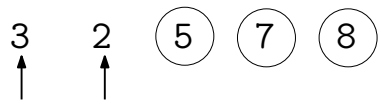


In round 2, the maximum of 5, 2, 3 is 5. Therefore, 5 will be swapped with the value 3. The positions of the elements to be exchanged are shown in the figure below.



After round 2, the list looks as follows.

In round 3, element 3 is the maximum value and it will be swapped with 2. The positions of the elements to be exchanged are shown in the figure below.



After round 3, the list looks as follows.

After this round the list becomes sorted. We had four rounds in total. The final list is as shown below.

This sorting algorithm is called *selection* sort. We know from Exercise Problems 4, Question no. 3, that for finding the position of the maximum number from a list of `n` numbers and exchanging it with the element in the last position, we require one loop (selection loop). But this step is repeatedly done in our algorithm. Therefore, we need an outer loop, which contains the selection loop inside it. In our program, we will assume that, the `n` elements to be sorted are stored in an array `a`. Assume that, the elements are `a[0], a[1], ..., a[n-1]`.

```
Round 0:
1. Find out maxposn, the position of the maximum among values
   from a[0] to a[n-1].
2. swap(a[n-1],a[maxposn]).
```

After round 0, the position of the maximum number is stored in maxposn and the values in `a[n-1]` and `a[maxposn]` are exxchanged. The next step is to find the position of the maximum number among values from `a[0]` to `a[n-2]` and exchange `a[maxposn]` with `a[n-2]`.

```
Round 1:
1. Find out maxposn, the position of the maximum among values
   from a[0] to a[n-2].
2. swap(a[n-2],a[maxposn]).
```

```
Round i:
1. Find out maxposn, the position of the maximum among values
   from a[0] to a[n-(i+1)].
2. swap(a[n-(i+1)],a[maxposn]).
```

We can see that, the task to be done in each round is similar. From this, we can deduce the general algorithm as given below.

```
i=0
while( i <= n-2 )
{
 find out maxposn, the position of the maximum among a[0] to a[n-(i+1)]
 swap(a[n-(i+1)],a[maxposn])
 i=i+1
}
```

The following program implements selection sort.

```
#include<stdio.h>
int main()
{
    int a[20], n, i, j, temp, maxposn;

    printf("enter n (<=20) \n");
    scanf("%d",&n);
    if(n>20)
    {
       printf("wrong input \n");
    }
    else
    {
       /*input array elements*/
       i=0;
       while (i < n)
       {
          printf("enter the next element \n");
          scanf("%d",&a[i]);
          i = i +1;
       }
       i=0;
       /* after one iteration of the loop, maximum among
          a[0] to a[n-(i+1)] is exchanged with a[n-(i+1)]*/

       while(i<=n-2)
       {
          maxposn=0;
          j=1;

          /*when the loop is completed, maxposn will be the index of the
            maximum element among a[0] to a[n-(i+1)]*/
```

```
            while(j<=n-(i+1))
            {
               if(a[j] > a[maxposn])
               {
                   maxposn=j;
               }
               j=j+1;
            }

            /*exchange a[maxposn] and a[n-(i+1)] */

            temp=a[n-(i+1)];
            a[n-(i+1)]=a[maxposn];
            a[maxposn]=temp;
            i=i+1;
         }
         i=0;
         printf("sorted list \n");
         while(i<n)
         {
            printf("%d ",a[i]);
            i=i+1;
         }
         printf("\n");
      }
}
```

This sorting is known as selection sort, because it will select the maximum value and place it to the last position. The selection step can also be done in a different way, in which the minimum number among list of numbers is selected and placed in the first position.