CS1100 - Lecture 5

Instructor : Jasine Babu

Teaching Assistants : Nikhila K N, Veena Prabhakaran

1 Introduction

In previous classes, we studied about various types of *if* statement and an iterative construct *while*.

Remark 1 *Any integer expression can replace the condition in while or if statements. For example, we can write* `while(x)`, `while(a+b)`. *These statements are also valid. If the expression has a non zero value, then it is considered as true(i.e. 1). If expression evaluates to 0, then it is considered as false (i.e. 0).*

Eg:

```
while(x)
{
 ...
 ...
 ...
}
```

The following *while* has the same logical meaning as the above.

```
while(x!=0)
{
 ...
 ...
 ...
}
```

Now consider the following examples.

```
while(1)
{
 ...
 ...
 ...
}
```

```
while(2)
{
 ...
 ...
 ...
}
```

These loops execute forever since `1` and `2` are expressions that never evaluate to `0`.

Arrays

Suppose we want to write a program that takes a number n, followed by a set of n numbers from user and gives this set of n numbers back to the user in reverse order. For example, for `n=5`, and the set of numbers 10,15,20,25,30 the program has to be print 30,25,20,15,10 as output. Each time this program is executed, the value of n and the n input values change. Can we write a program for this using only the tools that we have learned so far? The answer is no, because we have to "store" an unknown number of values in variables, which is not possible.

Now, consider another question. Suppose we know that the value of user is going to give the value of n to be at most 1000. Now, is it possible to write the list reversal program using only the tools that we have studied so far? Yes, it is possible. But such a program will be too long and loops alone cannot be used effectively handle this.

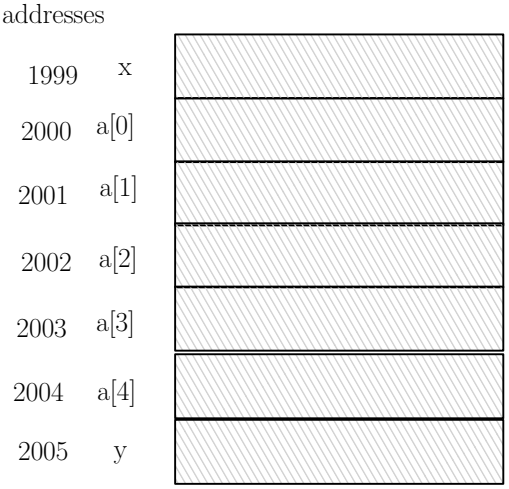
We should have a concise way of representing many variables together. A natural way of solving this problem is by using *arrays*.

An *array* is a common name given to a list of variables together. For example `int a[100]` means a is a name (or a label) given to a list of 100 integer variables. Note that, `a` itself is not a variable here.

The members of this array are denoted as `a[0], a[1], ... a[99]`. Here `a[0]` represents first element, `a[1]` represents second element and so on and `a[99]` represents last element. The number written within the square brackets (`[ ]`) is known as the index. Here 0,1 ... 99 are the indices of the array. In general, if we have a variable `counter`, which takes a value between 0 and 99, then writing `a[counter]` is valid. Moreover each of the elements `a[0], a[1], ... a[99]` have addresses and associated locations in memory as any other variables. This also means an expression like `a[5]` or `a[counter]` can appear in the left side of an assignment statement. However since `a` is not a variable `a` cannot appear in the left side of an assignment statement. The instruction `a[0]=5` assigns value 5 to the array's first memory location. If the value of the variable `counter` is 4, the instruction `a[counter]=50` assigns value 50 to the fifth element of the array.

Memory locations of array elements are always contiguous in memory. An example is shown below.

int x , y , a[5];



The following program reverses a list of `n` numbers where `n` is at most 1000.

```
int n,a[1000],counter
Input n
counter <-- 0
while(counter < n)
{
   Input a[counter]
   counter <-- counter+1
}
counter <-- n-1
while( counter >=0)
{
   Output a[counter]
   counter <-- counter-1
}
```

The memory state diagram of this program with inputs `n=4`, and list elements `10,20,30,40` is shown below.

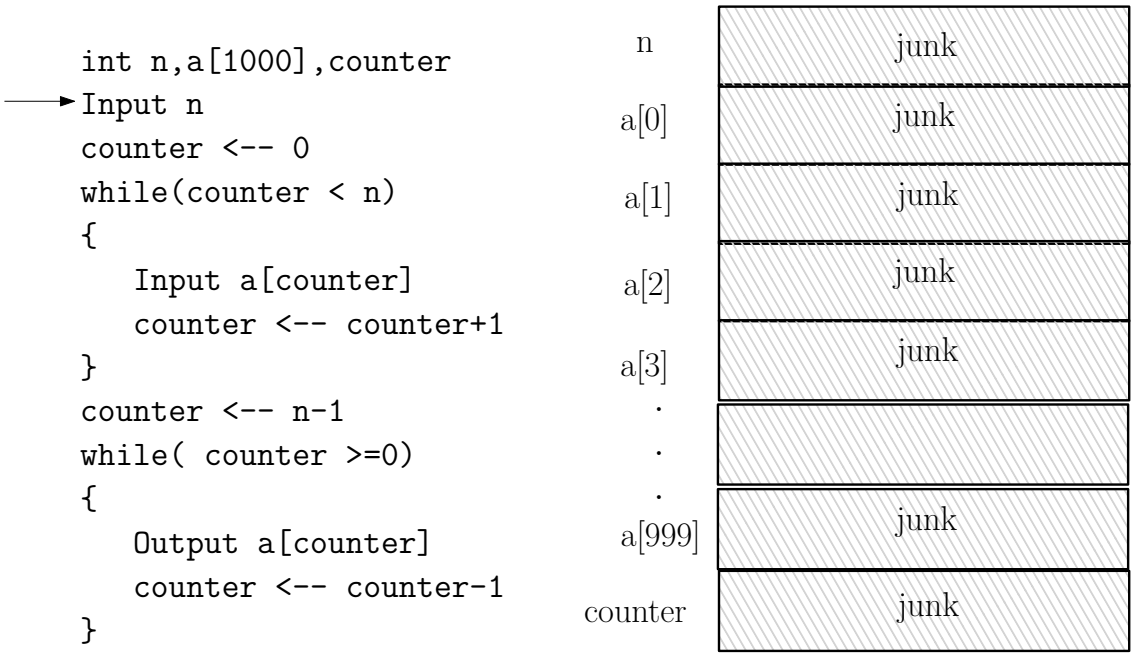


Figure 1: i

```
int n,a[1000],counter
Input n
counter <-- 0
while(counter < n)
{
   Input a[counter]
   counter <-- counter+1
}
counter <-- n-1
while( counter >=0)
{
   Output a[counter]
   counter <-- counter-1
}
```

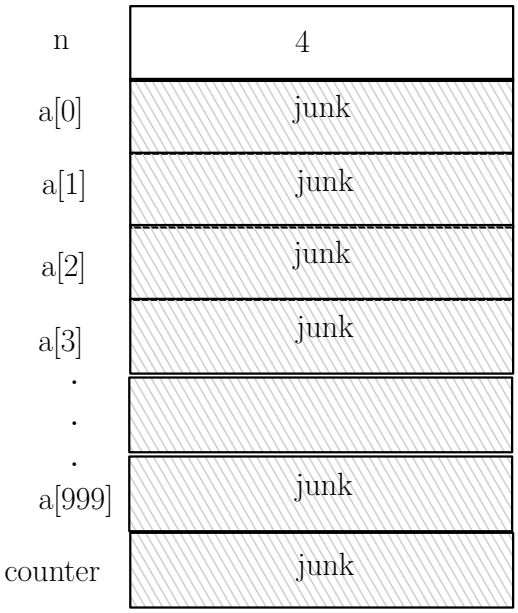


Figure 2: ii

```
int n,a[1000],counter
Input n
counter <-- 0
while(counter < n)
{
   Input a[counter]
   counter <-- counter+1
}
counter <-- n-1
while( counter >=0)
{
   Output a[counter]
   counter <-- counter-1
}
```

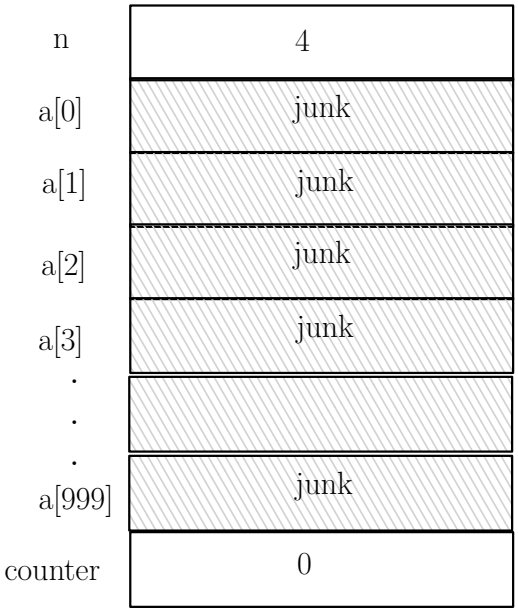


Figure 3: iii

```
int n,a[1000],counter
Input n
counter <-- 0
while(counter < n)
{
   Input a[counter]
   counter <-- counter+1
}
counter <-- n-1
while( counter >=0)
{
   Output a[counter]
   counter <-- counter-1
}
```

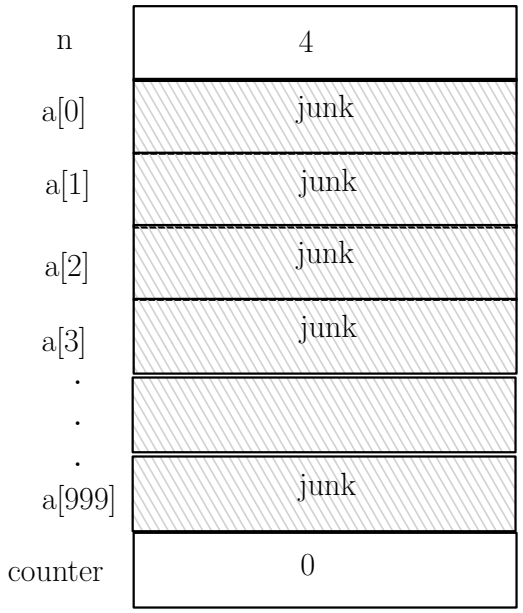


Figure 4: iv

```
int n,a[1000],counter
Input n
counter <-- 0
while(counter < n)
{
   Input a[counter]
   counter <-- counter+1
}
counter <-- n-1
while( counter >=0)
{
   Output a[counter]
   counter <-- counter-1
}
```

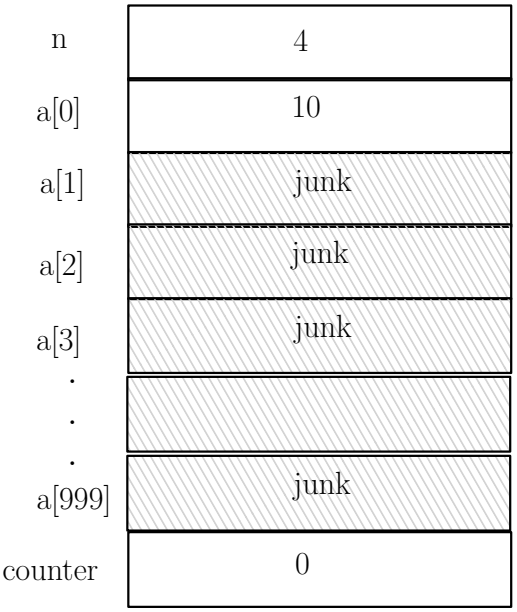


Figure 5: v

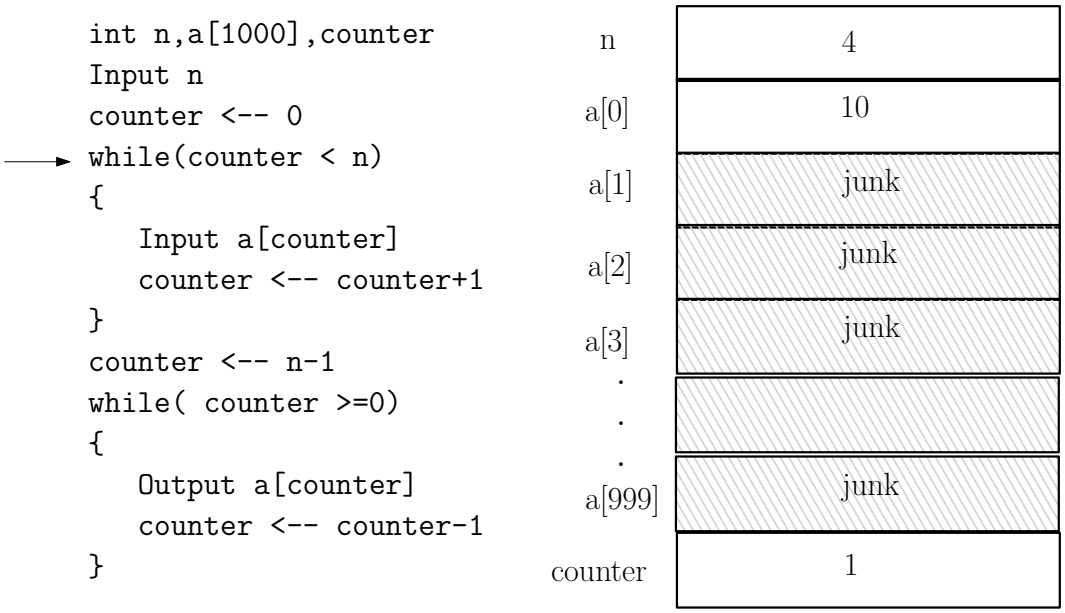


Figure 6: vi

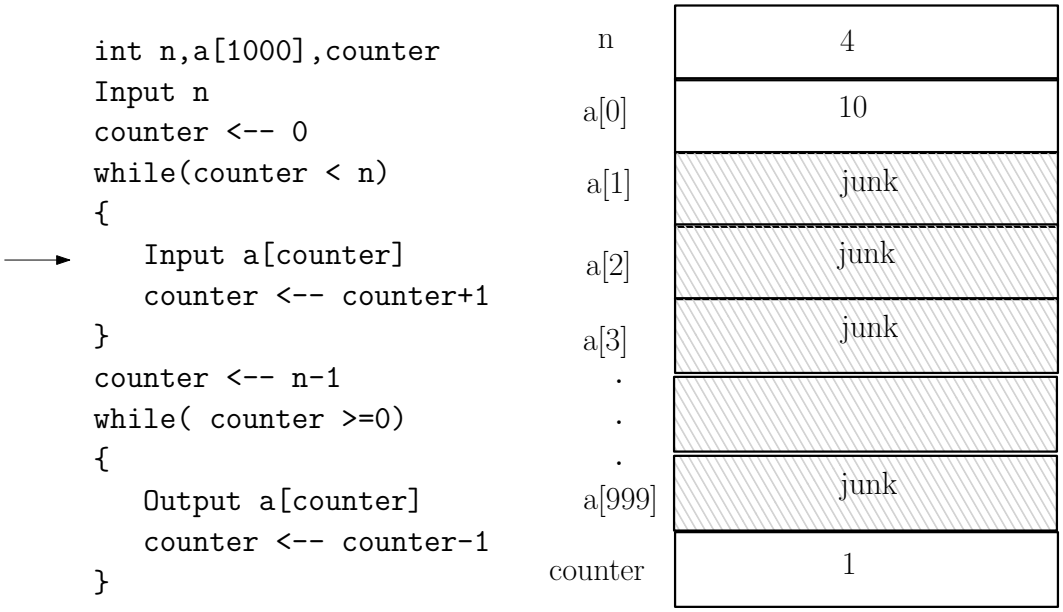


Figure 7: vii

```
int n,a[1000],counter
Input n
counter <-- 0
while(counter < n)
{
   Input a[counter]
   counter <-- counter+1
}
counter <-- n-1
while( counter >=0)
{
   Output a[counter]
   counter <-- counter-1
}
```

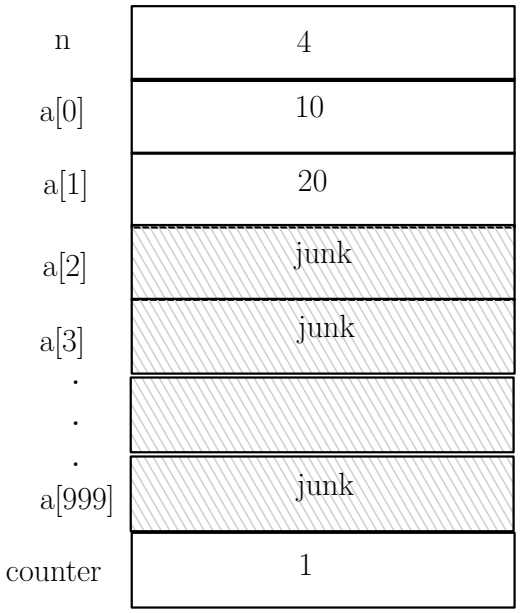


Figure 8: viii

```
int n,a[1000],counter
Input n
counter <-- 0
while(counter < n)
{
   Input a[counter]
   counter <-- counter+1
}
counter <-- n-1
while( counter >=0)
{
   Output a[counter]
   counter <-- counter-1
}
```

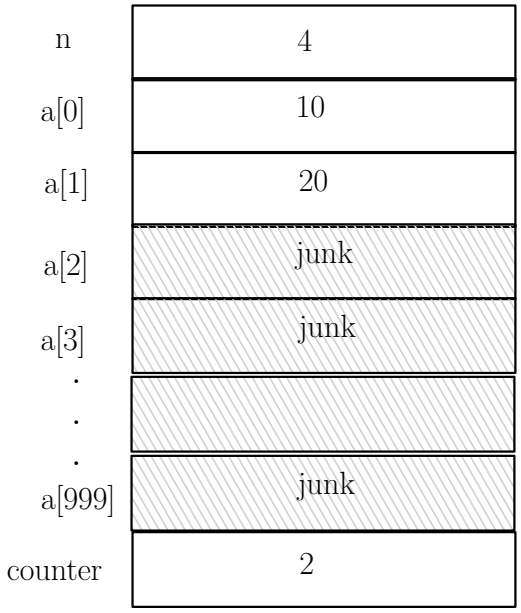


Figure 9: ix

```
int n,a[1000],counter
Input n
counter <-- 0
while(counter < n)
{
   Input a[counter]
   counter <-- counter+1
}
counter <-- n-1
while( counter >=0)
{
   Output a[counter]
   counter <-- counter-1
}
```

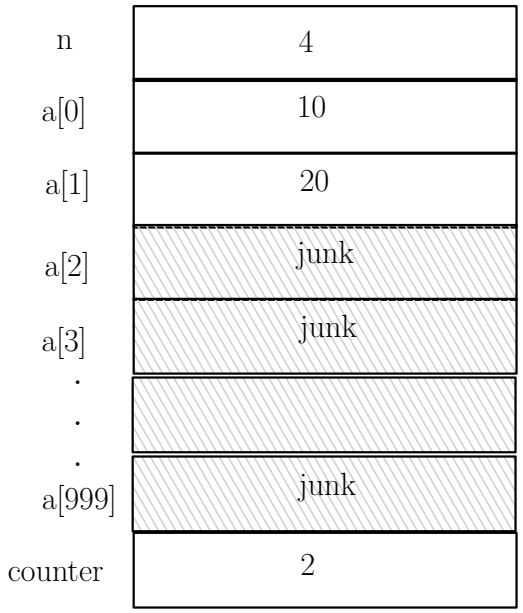


Figure 10: x

```
int n,a[1000],counter
Input n
counter <-- 0
while(counter < n)
{
   Input a[counter]
   counter <-- counter+1
}
counter <-- n-1
while( counter >=0)
{
   Output a[counter]
   counter <-- counter-1
}
```

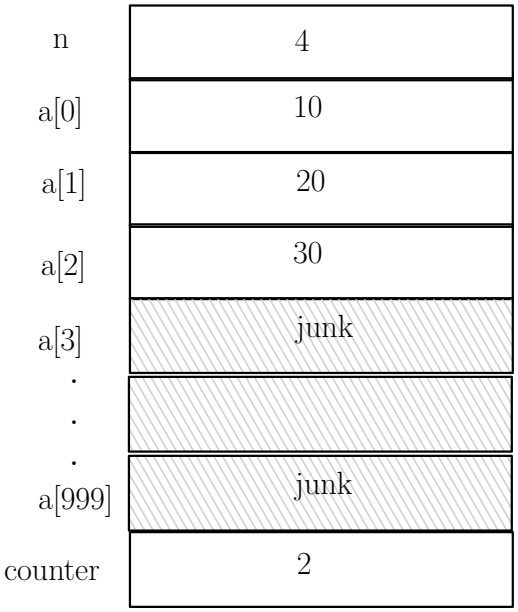


Figure 11: xi

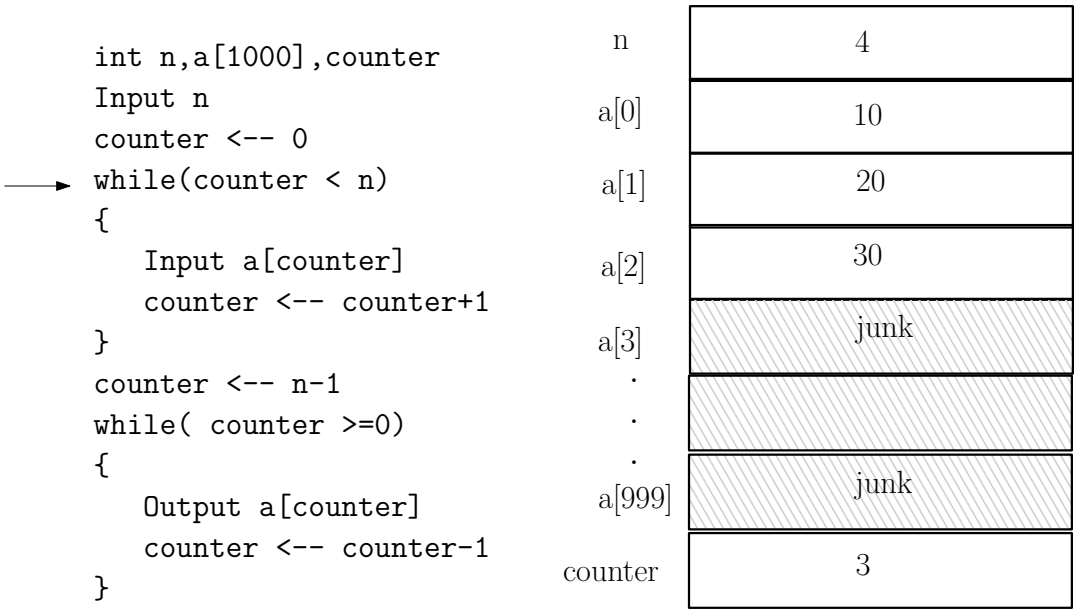


Figure 12: xii

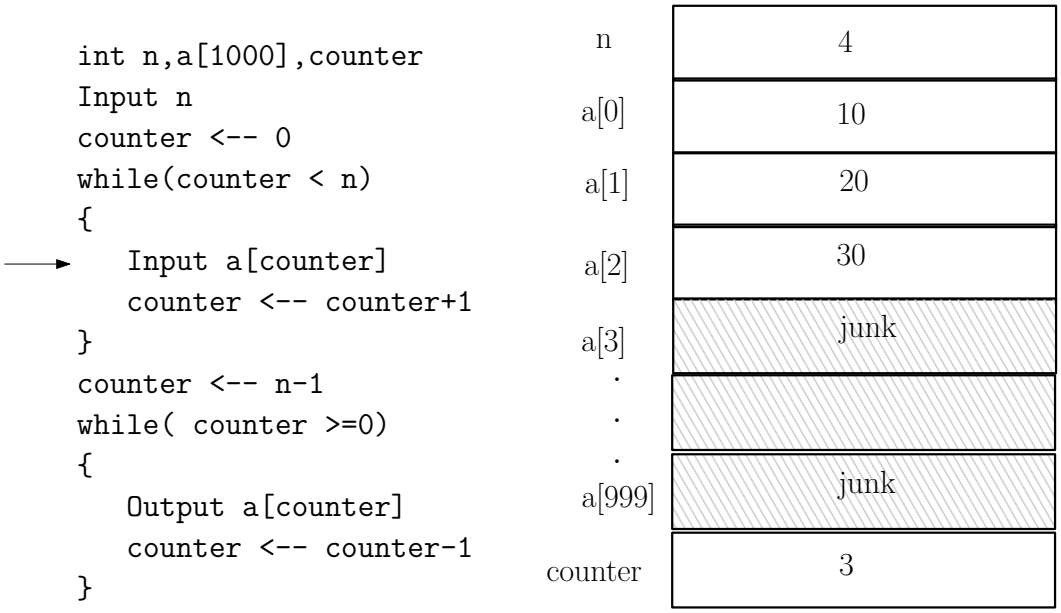


Figure 13: xiii

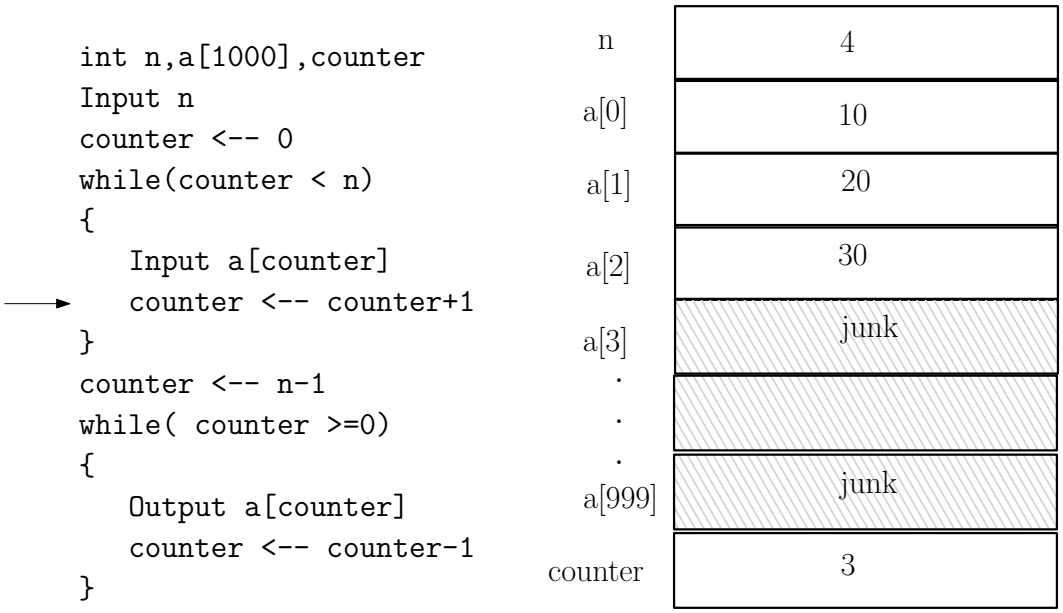


Figure 14: xiv

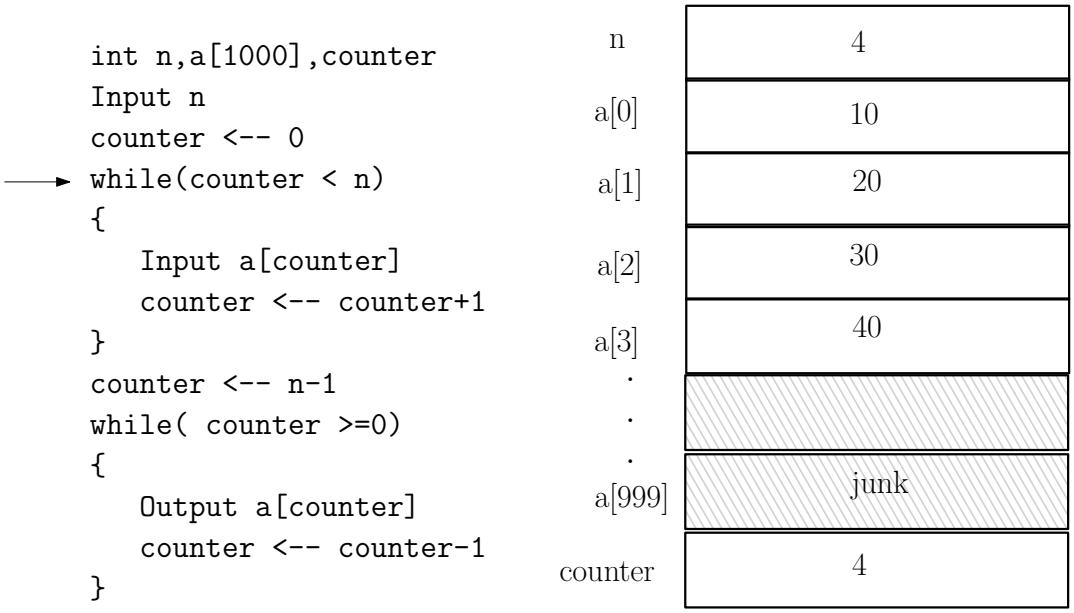


Figure 15: xv

Note that, the program control goes to the line `while(counter < n)` 5 times and when the control comes out of this loop, the value of `counter` is 4 (the value of `n`). If we try to print `a[counter]` at this point, it will print a junk value. Therefore, the statement `counter <-- n-1` after the first `while` loop is necessary.

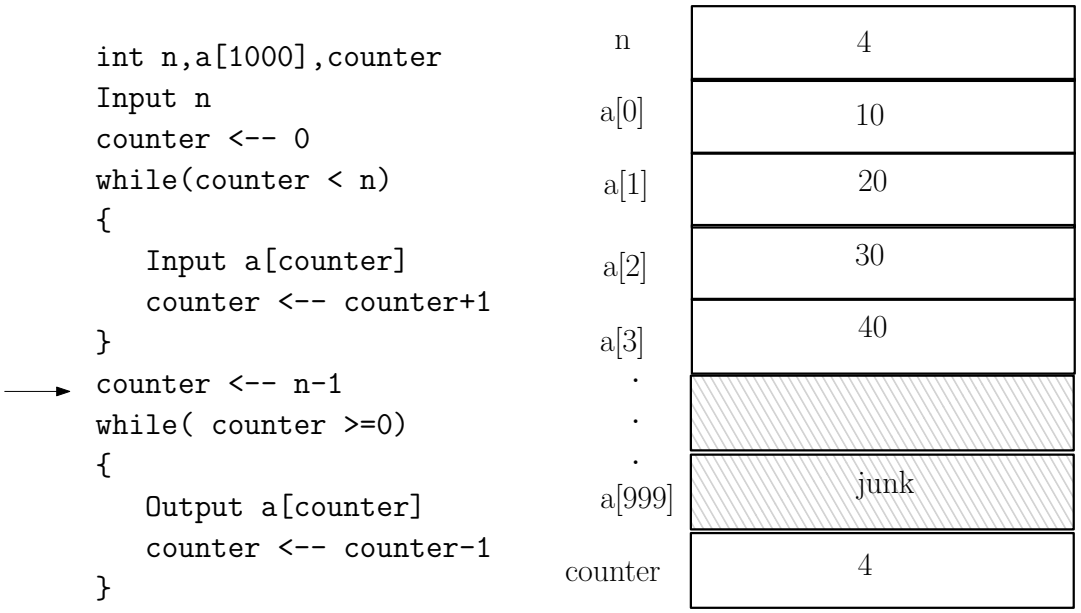


Figure 16: xvi

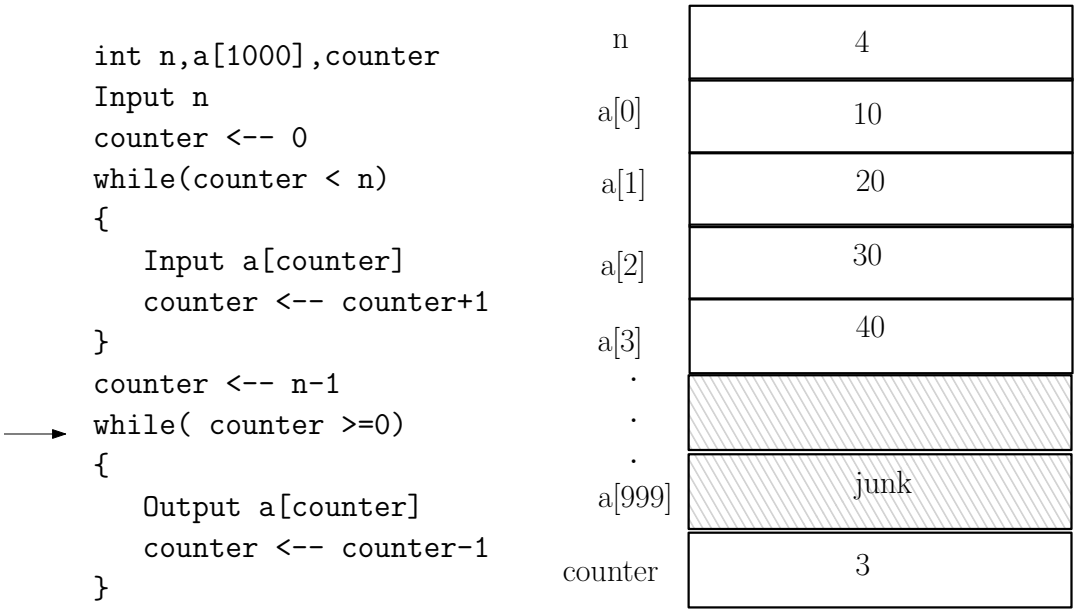


Figure 17: xvii

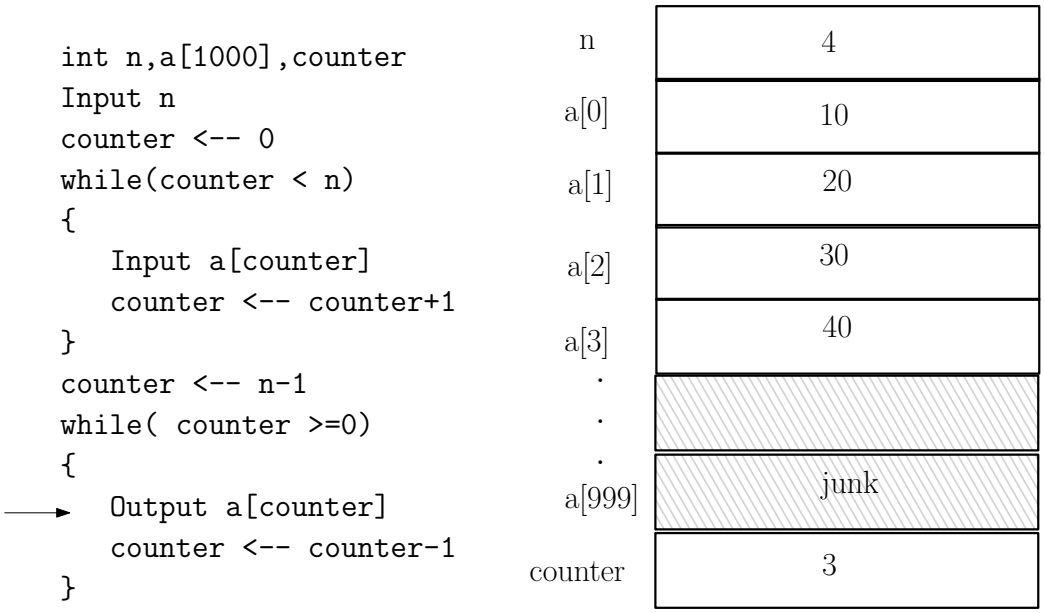


Figure 18: xviii

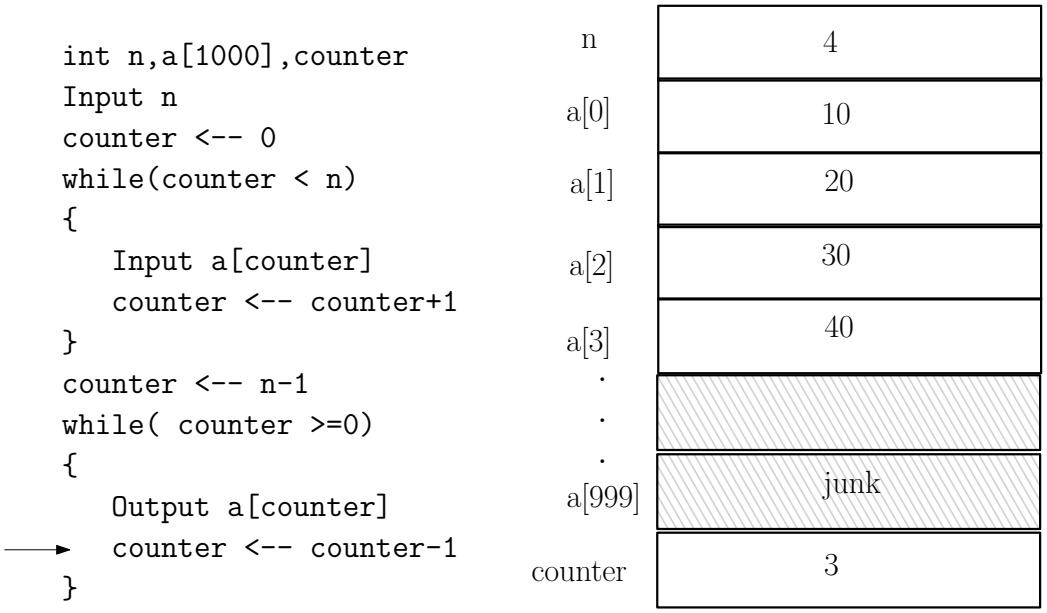


Figure 19: xix

Now, output `40` is printed on screen.

```
int n,a[1000],counter
Input n
counter <-- 0
while(counter < n)
{
   Input a[counter]
   counter <-- counter+1
}
counter <-- n-1
while( counter >=0)
{
   Output a[counter]
   counter <-- counter-1
}
```

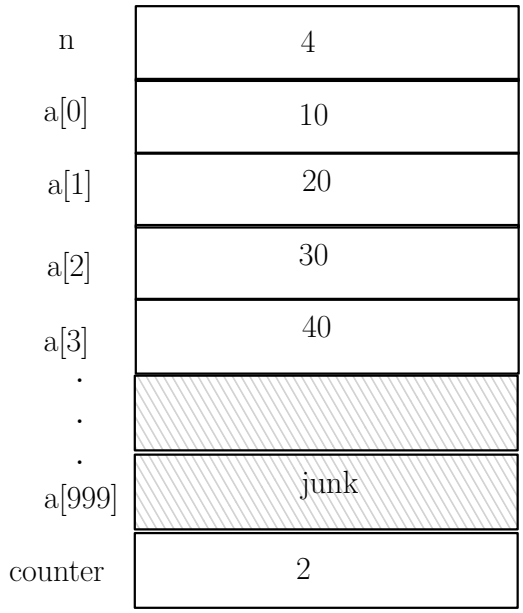


Figure 20: xx

```
int n,a[1000],counter
Input n
counter <-- 0
while(counter < n)
{
   Input a[counter]
   counter <-- counter+1
}
counter <-- n-1
while( counter >=0)
{
   Output a[counter]
   counter <-- counter-1
}
```

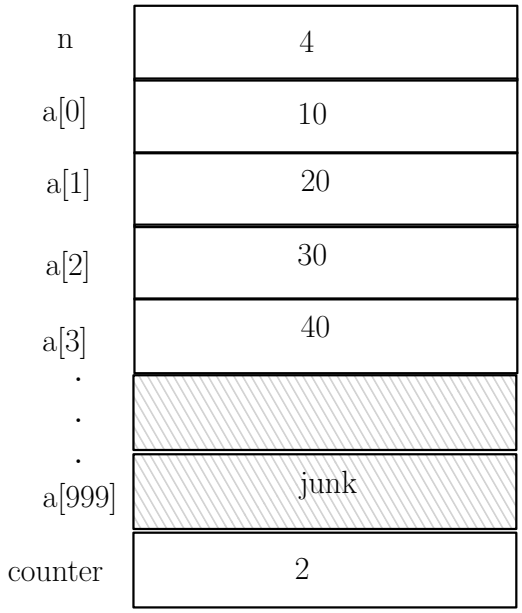


Figure 21: xxi

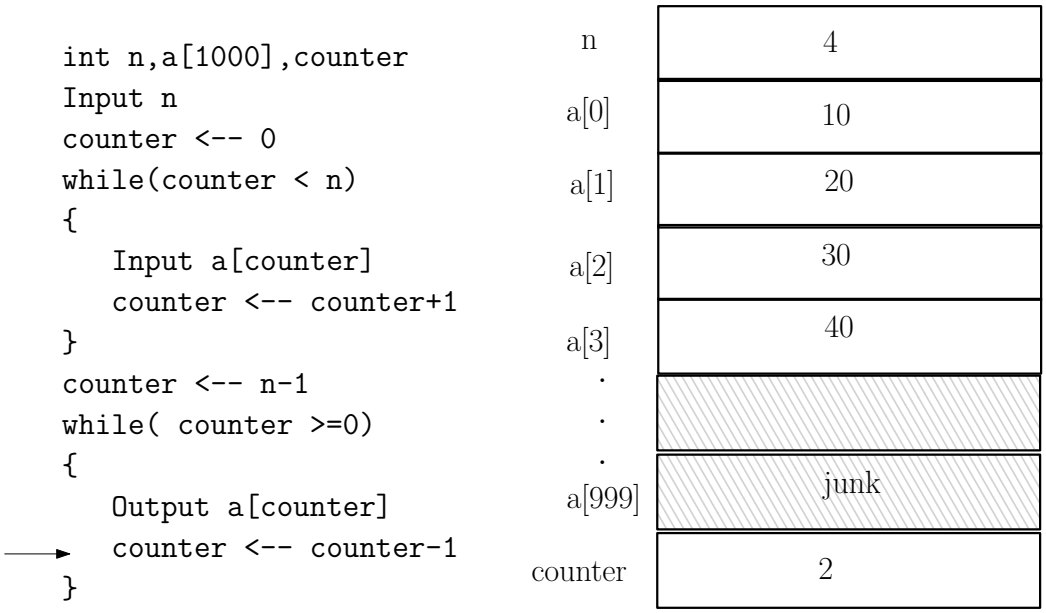


Figure 22: xxii

At this point `40 30` is printed on screen.

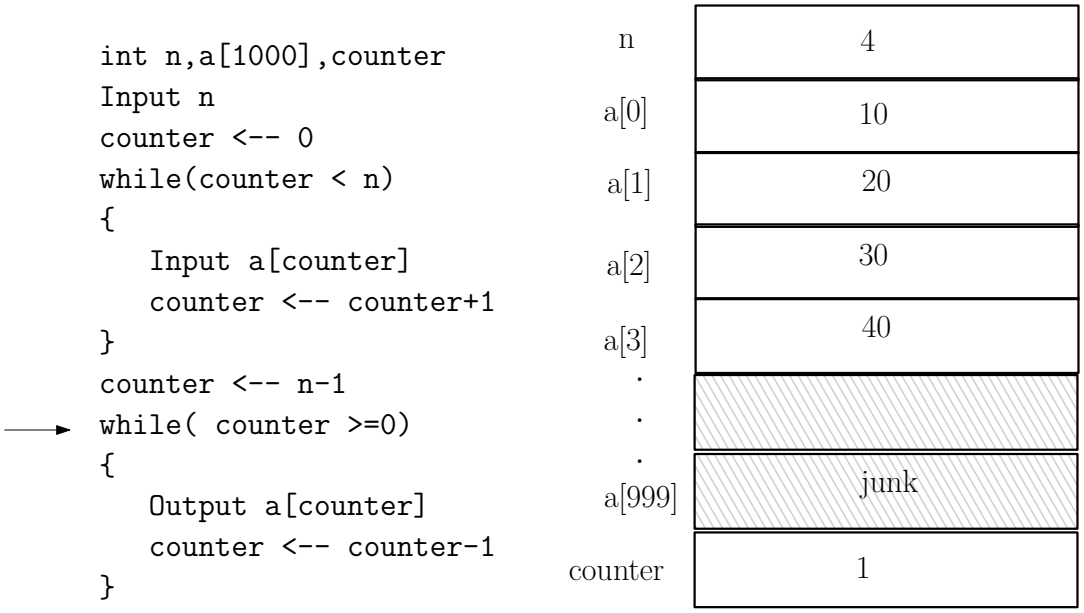


Figure 23: xxiii

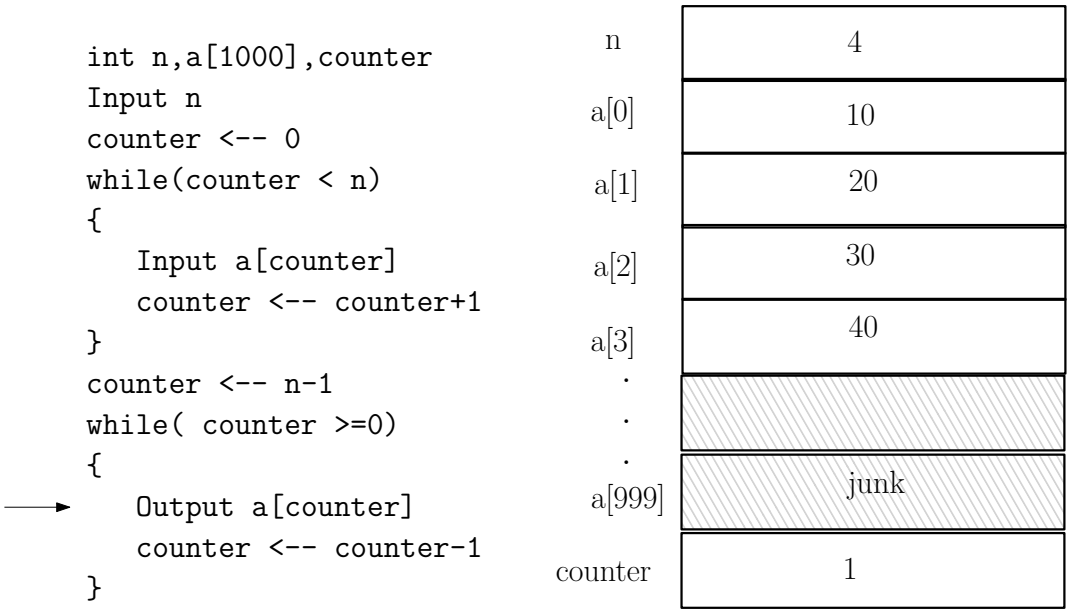


Figure 24: xxiv

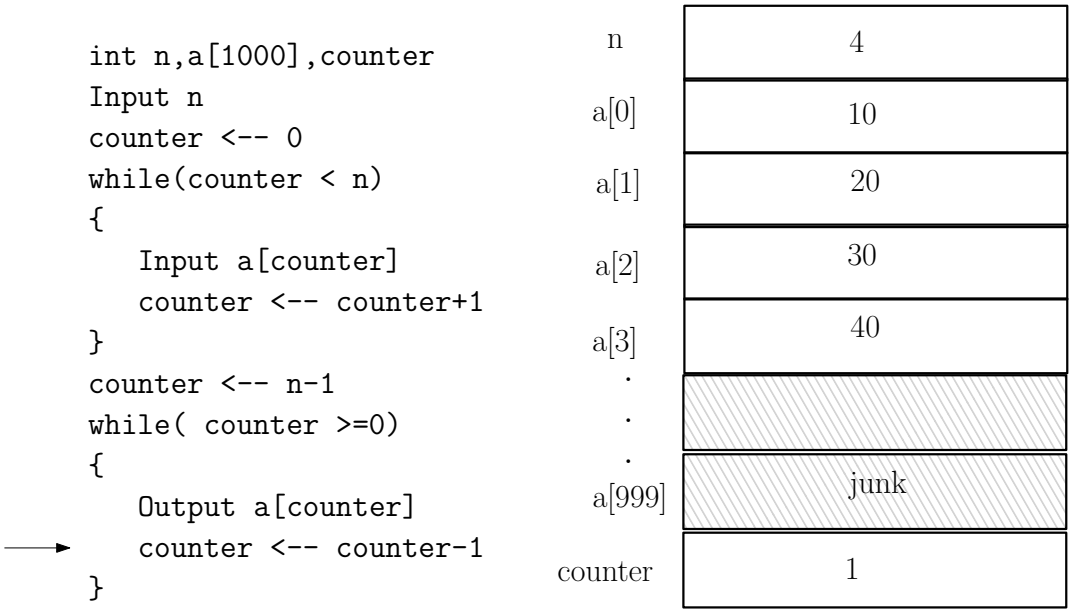


Figure 25: xxv

At this point `40 30 20` is printed on screen.

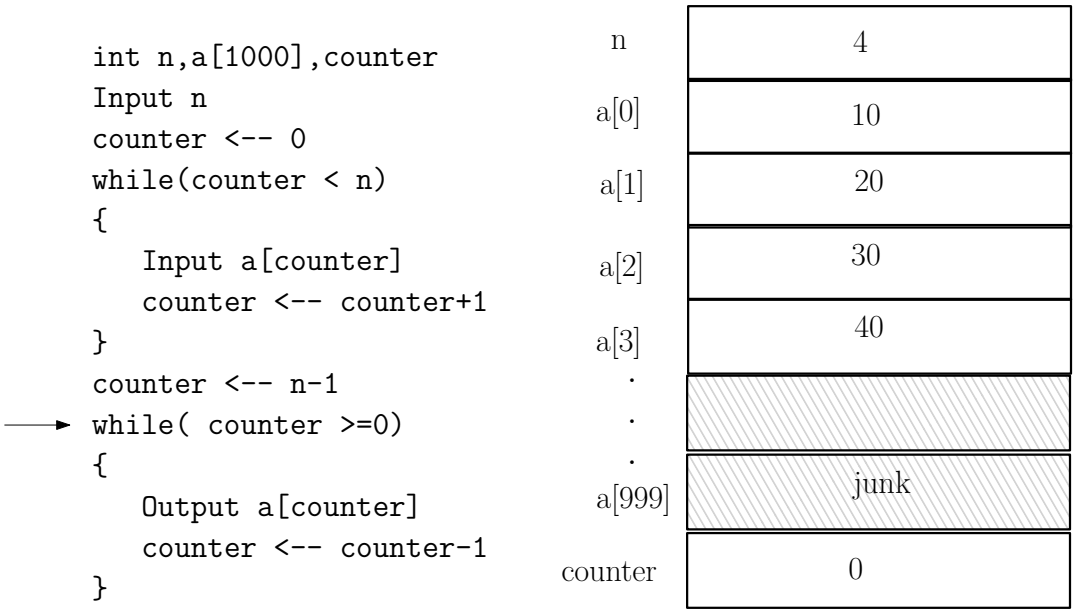


Figure 26: xxvi

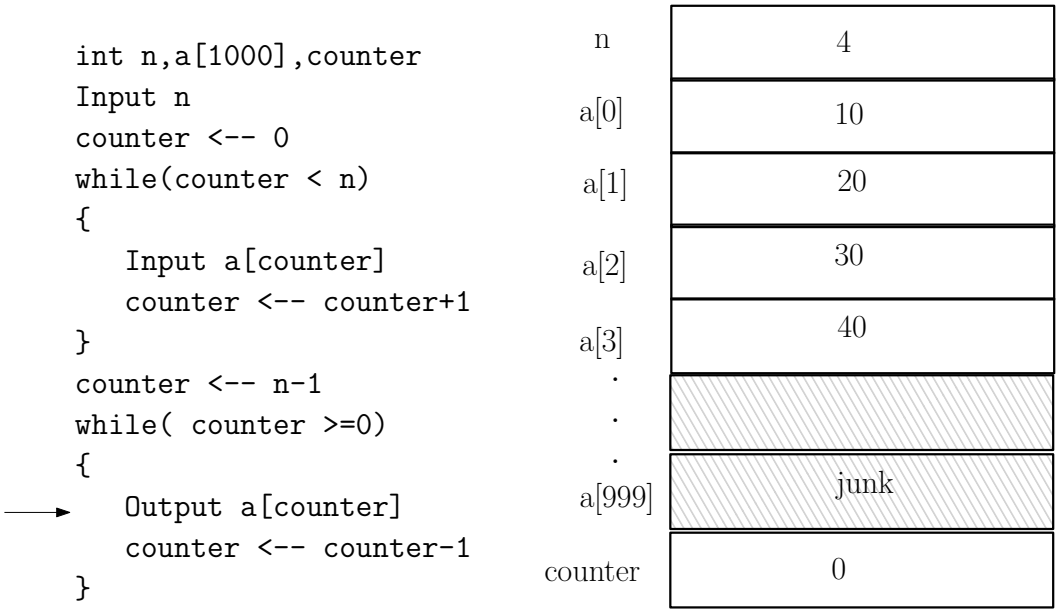


Figure 27: xxvii

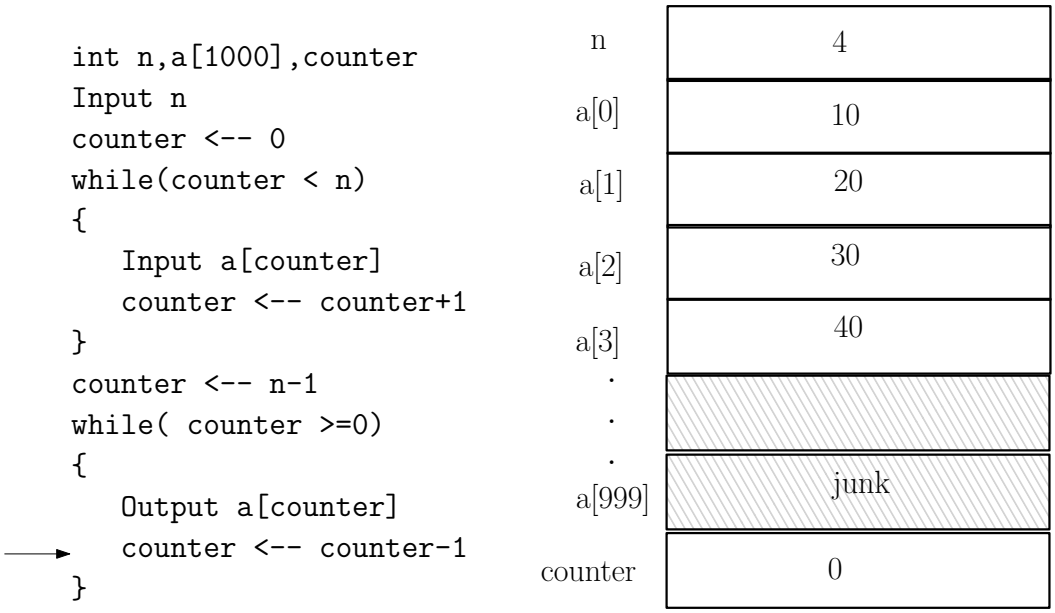


Figure 28: xxviii

At this point `40 30 20 10` **is printed on screen.**

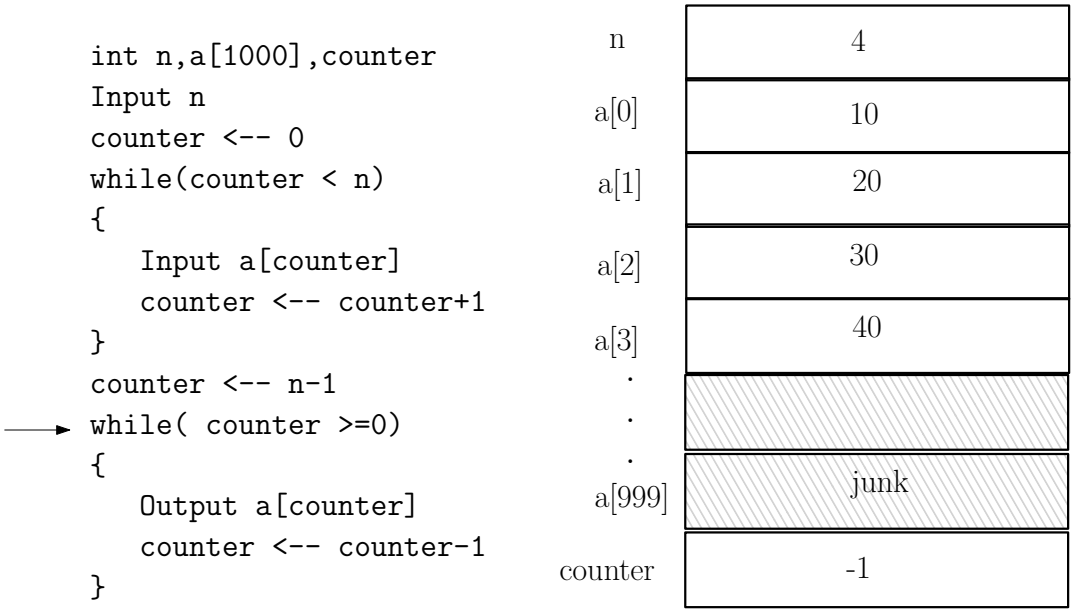


Figure 29: xxix

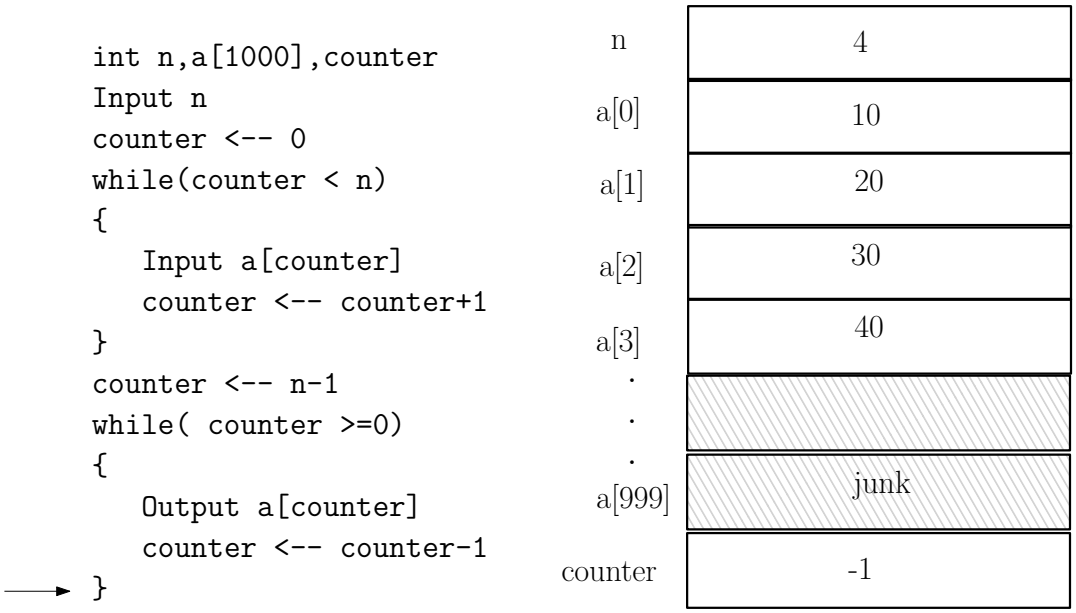


Figure 30: xxx