Exercise 4: Iterative Statement

R Ram Kaushik

April 13, 2018

1 Indent

Define a function `indent(n)` to print n times the pattern (`|--`) in a line. `n`, the number of times is given as a parameter. Construct an input file in which each line is formated as a pair of numbers referred to as `level` and `key`. For each line

```
level key
```

print `level` number of times the pattern `|--` followed by the `key`. (*count*).

```
0 5
1 10
1 15
2 20
2 25
2 30
2 35
3 40
3 45
3 50
3 55
```

When the lines in the file are read and printed, the display will be as shown below.

```
5
|--10
|--15
|--|--20
|--|--25
|--|--30
|--|--35
|--|--|--40
|--|--|--45
|--|--|--50
|--|--|--55
```

(*count*)

1.1 Specification

A function `indent()`, which takes an array `a[]`, number of lines `n` as input and prints the `level` and `key` based on the array on the `stdout`.

1.2 Prototype

```
void indent(int a[], int n)
```

1.3 Program Design

The program consists of a function `indent(int a[], int n)`, which prints the `level` and `key` on the `stdout`, and `main()`, which reads the input from `stdin` and calss the function.

1.4 Algorithm

```
def indent(a,n):
   for i in range(n):
      for j in range(a[i]):
         print("|--")
      print("%d\n",5*(i+1))
```

1.5 Source Code

```
#include<stdio.h>
void indent(int a[], int n){
  for(int i=0;i<n;i++){
    for(int j=0;j<a[i];j++){
      printf("|--");
    }
    printf("%d\n",5*(i+1));
  }
}
int main(){
  int n,a[100];
  scanf("%d",&n);
  for(int i=0;i<n;i++){
    scanf("%d",&a[i]);
  }
  indent(a,n);
}
```

1.6 Test Input

```
11
0 1 1 2 2 2 2 3 3 3 3
```

1.7 Output

5

2 Array Length

Represent a list of numbers by an array of numbers terminated by -1 as the end of list marker. Define a function `int array_len(int a[])` that takes such an array as the parameter and returns the length of the array, that is, the number of items in the array (*sentinel*)

2.1 Specification

A function `array_len()`, which takes an array `a[]` as the input, counts the number of elements in the array and returns it to the calling function.

2.2 Prototype

```
int array_len(int a[])
```

2.3 Program Design

The program consists of a function `array_len(int a[])`, which counts the number of elements in the array, and `main()`, which gets the input from `stdin`, calls the function, and prints the result on `stdout`.

2.4 Algorithm

```
def array_len(a):
   i=0
   while a[i]!=-1:
      i++
   return i
```

2.5 Source Code

```
#include<stdio.h>
int array_len(int a[]){
  int i=0;
  while(a[i]!=-1){
    i++;
  }
  return i;
}
int main(){
  int a[100],i=0,m;
  while(1){
    scanf("%d",&a[i]);
    if(a[i]==-1){
      break;
    }
    i++;
  }
  m=array_len(a);
  printf("%d\n",m);
}
```

2.6 Test Input

```
2 4 6 13 12 11 17 19 21 -1
```

2.7 Output

3 Reading input data

1. Read a list of numbers using `scanf()`. Format the list of numbers as a line of numbers in the input file, as illustrated below. Read the `n` items with a loop. In each iteration of the loop, let `scanf()` read one number. (*sentinel*)

   ```
   10 20 30 40 50 60
   ```

2. Read a list of `k`-tuples. Format a `k`-tuple of items as a line of `k` items, and the list as a sequence of lines, in the input file, as illustrated below. Read the list with a loop. In each iteration, let `scan()` read `k` items from the file.

```
10 20 30
30 40 50
60 70 90
```

3. Read a list of lists. Format a simple list as a line of items. Format list of lists as a sequence of lines, in the input file, as illustrated below.

   ```
   10 20 30 40 50
   30 40 50
   60 70 90 50 60 70
   ```

   Read each line into a variable `line` and parse the line into integers.

   ```
   for (i = 0; sscanf(line, "%d%n", &a[i], &nbytes) == 1; i++)
      line += nbytes;
   ```

3.1 Specification

4 Sub Array

Print a subarray. Write a function `print_array(a, low, high)` that prints the subarray `a[low:high]`, that is, the items of array `a` from `low` to `high`. `low` and `high` are called the *lower bound* and *upper bound* of the subarray. We follow the convention of upper bound excluded. That is, (*visitor*)

```
a[l:h] = a[l], a[l+1], ..., a[h-1]
```

Note that `a[h]` is not a part of `a[l:h]`.

4.1 Specification

A function `sub_array()`, which takes an array `a[]`, lower limit `l`, upper limit `h` as inputs and prints array `a[l:h]` on `stdout`.

4.2 Prototype

```
void sub_array(int a[], int l, int h)
```

4.3 Program Design

The program consists of a function `sub_array(int a[], int l, int h)`, which prints the sub array from `l` to `h`, and `main()`, which reads the input from `stdin`, and calls the function.

4.4 Algorithm

```
def sub_array(a,l,h):
   for i in range(l,h):
      print(a[i])
```

4.5 Source Code

```
#include<stdio.h>
void sub_array(int a[], int l, int h){
  for(int i=l;i<h;i++){
    printf("%d%s",a[i],i==h-1?"":",");
  }
}
int main(){
  int a[100],n,l,h;
  scanf("%d",&n);
  for(int i=0;i<n;i++){
    scanf("%d",&a[i]);
  }
  scanf("%d%d",&l,&h);
  sub_array(a,l,h);
}
```

4.6 Test Input

```
10
3 9 7 1 0 5 6 8 2 4
3 7
```

4.7 Output

1,0,5,6

5 Sum, mean, variance

1. Define a function `sum(array, low, high)` that computes the sum of the numbers of the subarray `array[low:high]`. Using this function, define a function `mean(array, low, high)` to compute the mean of the numbers in the subarray `array[low:high]`.

2. Write a function `variance(array, low, high)` to compute the variance of the numbers of the subarray `array[low:high]`. Let `variance()` use `mean()`. Test the functions `mean()` and `variance()` from `main()` which should read a list of numbers from a file and print the mean and variance. Test it for several lists of numbers. (*accumulator, map*)

3. Write a function to find the number of items above the mean.

5.1 Specification

4 functions `sum()`, which finds the sum of `a[l:h]`, `mean()`, which finds the mean, `variance()`, which finds the variance, and `count()` which finds number of people above the mean.

5.2 Prototype

```
int sum(int a[], int l, int h)
float mean(int a[], int l, int h)
float variance(int a[], int l, int h)
int count(int a[], int l, int h)
```

5.3 Program Design

The program consists of 4 functions `sum(int a[], int l, int h)`, which finds sum and returns it, `mean(int a[], int l, int h)`, which finds mean and returns it, `variance(int a[], int l, int h)` which finds variance and returns it, `count(int a[], int l, int h)`, which finds number of people above the mean and returns it and `main()`, which gets input from `stdin`, calls the functions and prints the result on `stdout`.

5.4 Algorithm

```
def sum(a,l,h):
   s=0
   for i in range(l,h):
      s+=a[i]
   return s
def mean(a,l,h):
   return sum(a,l,h)/(1.0*(h-l))
def variance(a,l,h):
   m=mean(a,l,h),s=0
   for i in range(l,h):
      s+=(a[i]-m)^2
   return s/(h-l)
```

```
def count(a,l,h):
   m=mean(a,l,h)
   s=0
   for i in range(l,h):
      if a[i]>m:
         s++
   return s
```

5.5 Source Code

```
#include<stdio.h>
int sum(int a[], int l, int h){
  int s = 0;
  for(int i = l; i < h; i++){
    s += a[i];
  }
  return s;
}
float mean(int a[], int l, int h){
  return sum(a, l, h)/(1.0*(h - l));
}
float variance(int a[], int l, int h){
  float m = mean(a, l, h), s = 0;
  for(int i = l; i < h; i++) {
    s += ((a[i] - m)*(a[i] - m));
  }
  return s/(1.0*(h - l));
}
int count(int a[], int l, int h){
  float m = mean(a, l, h);
  int s = 0;
  for(int i = l; i < h; i++) {
    if(a[i] > m){
      s++;
    }
  }
  return s;
}
int main(){
```

```
  int a[5], l, h;
  for(int i = 0; i < 5; i++) {
    scanf("%d", &a[i]);
  }
  scanf("%d %d", &l, &h);
  printf("%d %f %f %d", sum(a, l, h), mean(a, l, h), variance(a, l, h), count(a,
  return 0;
}
```

5.6 Test Input

```
72 144 53 69 78
0 5
```

5.7 Output

416 83.199997 992.560059 1

6 Prime number

Define a function `is_prime(n)` that tests whether a non-negative integer `n` is a prime number and returns `true` if `n` is prime and `false` if `n` is not prime. Test it for the first 100 integers. (*search*)

6.1 Specification

A function `is_prime()`, which takes the number `a` as input, checks if a number is prime or not and returns the result.

6.2 Prototype

```
int is_prime(int a)
```

6.3 Program Design

The program consists of a function `is_prime(int a)`, which checks if a number is prime or not and `main()`, which gets the input from `stdin`, calls the function and prints the result on `stdout`.

6.4 Algorithm

```
def is_prime(a):
   i=2,f=1
```

```
   while i<a/2:
      if a%i==0:
         f=0
         break
      i++
   return f
```

6.5 Source Code

```
#include<stdio.h>
int is_prime(int a){
  int i=2,f=1;
  while(i<a/2){
    if(a%i==0){
      f=0;
      break;
    }
    i++;
  }
  return f;
}
int main(){
  int n,f;
  scanf("%d",&n);
  f=is_prime(n);
  if(f==1){
    printf("Prime");
  }
  else{
    printf("Not prime");
  }
}
```

6.6 Test Input

```
11
14
```

6.7 Output

Prime
Not prime

7 Linear search (*search*)

1. Define a function `linear_search(a, n, target)`. It searches the subarray `a[0:n]` for the `target`. If the target is in the array, the function returns the index of the target. If the target is not in the array, the function should return an invalid index (an invalid index is one outside the range $0 \leq index < n$). Test the function from `main()`. Let `main()` read the input from stdin. Write two versions of `linear_search()`, one using `break` and the other without using `break`.

2. Implement a third version of `linear_search(array, n, target)` that uses the `target` as the sentinel at `a[n]`. Write the specification for the function.

7.1 Specification

3 functions `linear_search()`, `linear_search_n()`, `binary_search()` all which get an integer array, its length and target element as input and returns an index as the output.

7.2 Prototype

```
int linear_search(int a[], int n, int t)
int linear_search_n(int a[], int n, int t)
int binary_search(int a[], int n, int t)
```

7.3 Program Design

The program consists of 3 functions `linear_search(int a[], int n, int t)`,`linear_search_n(i a[], int n, int t)`,`binary_search(int a[], int n, int t)` which returns an index of whether an element exists in array to the caller, and `main()`, which gets the input from `stdin`, calls the function and prints the outputon `stdout`.

7.4 Algorithm

```
def linear_search(a,n,t):
   for i in range(n):
      if a[i]==t:
         break
   return i
def linear_search_n(a,n,t):
   i=0
```

```
   while i<n and a[i]!=t:
      i=i+1
   return i
def binary_search(a,n,t):
   l=0,u=n-1,f=0,m
   while l<=u and f=0:
      m=(l+u)/2
      if t==a[m]:
         f=m
      elif a[m]>t:
         u=m-1
      else:
         l=m+1
   if f==0:
      return -1
   return f
```

7.5 Source Code

```
#include<stdio.h>
int linear_search(int a[], int n, int t){
  int i = 0;
  for(i = 0; i < n; i++) {
    if(a[i] == t){
      break;
    }
  }
  return i;
}
int linear_search_n(int a[], int n, int t){
  int i = 0;
  while(i < n && a[i] != t){
    i++;
  }
  return i;
}
int binary_search(int a[], int n, int t){
  int l = 0, u = n - 1, flag = 0, mid;
  while(l <= u && flag == 0) {
```

```
    mid = (l + u)/2;
    if(t == a[mid]){
      flag = mid;
    }
    else if(a[mid] > t) {
      u = mid - 1;
    }
    else{
      l = mid + 1;
    }
  }
  if(flag == 0){
    return -1;
  }
  return flag;
}
int main(){
  int a[100], n, t;
  scanf("%d", &n);
  for(int i = 0; i < n; i++) {
    scanf("%d", &a[i]);
  }
  scanf("%d", &t);
  printf("%d %d %d", linear_search(a,n,t),linear_search_n(a,n,t),binary_search(a
  return 0;
}
```

7.6 Test Input

```
10
10 12 15 25 29 37 69 78 87 100
37
```

7.7 Output

5 5 5

8 Minimum

We are given an array `a[0:n]` of `n` comparable items. Define a function `minimum(a, low, high)` that returns the index of the smallest item in the subarray `a[low:high]`. Test the function from `main()` for several lists of numbers. Each test should read a list of numbers from stdin. (*accumulator*)

8.1 Specification

A function `min()`, which takes the array `a[]`, lower bound `l` and upper bound `h` as inputs and returns the index of the smallest element.

8.2 Prototype

```
int min(int a[], int l, int h)
```

8.3 Program Design

The program consists of a function `min(int a[], int l, int h)`, which returns the index of the smallest element, and `main()`, which gets the input from `stdin`, calls the function and prints the output on `stdout`.

8.4 Algorithm

```
def min(a,l,h):
   m=l
   for i in range(l+1,h):
      if a[i]<a[m]:
         m=i
   return m
```

8.5 Source Code

```
#include<stdio.h>
int min(int a[], int l, int h){
  int m=l;
  for(int i=l+1;i<h;i++){
    if(a[i]<a[m]){
      m=i;
    }
  }
  return m;
}
```

```
int main(){
  int n,a[30],m,l,h;
  scanf("%d",&n);
  for(int i=0;i<n;i++){
    scanf("%d",&a[i]);
  }
  scanf("%d%d",&l,&h);
  m=min(a,l,h);
  printf("%d",m);
}
```

8.6 Test Input

```
10
0 9 1 8 2 7 3 6 4 5
1 6
```

8.7 Output

2

9 Armstrong number

1. Define a function `int to_digits(n, s)` to convert an integer to a string of single digit numbers. For example, it converts 371 to [3,7,1]. The function has two outputs:
   (a) `s`, an array of single digit numbers, which is passed as a parameter, and
   (b) the number of single digits, which is returned as a value.

   Test the function from `main()`.
2. Define a function `cube(x)` that returns x^3.
3. Write a function `is_armstrong(n)` that tests whether the integer `n` is an Armstrong number. An Armstrong number is equal to the sum of cubes of its digits. Test the function to find out all the Armstrong numbers from 0 to 500.

9.1 Specification

3 functions `to_digits()`, which gets the number `n` and array `a[]` as input, stores each digit in the array and returns number of digits, `cube()`, which finds the cube of a number, and `is_armstrong()`, which gets the number, each individual digit and its length as input and checks if a number is armstrong or not.

9.2 Prototype

```
int to_digits(int n, int s[])
int cube(int n)
int is_armstrong(int n, int s[], int b)
```

9.3 Program Design

The program consists of 3 functions `to_digits(int n, int s[])` which finds number of digits and stores them in an array, `cube(int n)` which finds cube of a number, `is_armstrong(int n, int s[], int b)` which checks if a number is armstrong or not, and `main()`, which gets the input from `stdin`, calls the functions and prints the result on `stdout`.

9.4 Algorithm

```
def to_digits(n,s):
   i=0
   while n!=0:
      s[i]=n%10
      n/=10
      i+=1
   return i
def cube(n):
   return n*n*n
def is_armstrong(n,s,b):
   a=0
   for i in range(b):
      a+=cube(s[i])
   if n==a:
      return 1
   return 0
```

9.5 Source Code

```
#include<stdio.h>
int to_digits(int n, int s[]){
  int i=0;
  while(n!=0){
    s[i]=n%10;
    n/=10;
```

```
    i++;
  }
  return i;
}
int cube(int n){
  return n*n*n;
}
int is_armstrong(int n, int s[], int b){
  int a=0;
  for(int i=0;i<b;i++){
    a+=cube(s[i]);
  }
  if(n==a){
    return 1;
  }
  return 0;
}
int main(){
  int n,s[30],f,a;
  scanf("%d",&n);
  a=to_digits(n,s);
  f=is_armstrong(n,s,a);
  if(f==1){
    printf("Armstrong");
  }
  else{
    printf("Not Armstrong");
  }
}
```

9.6 Test Input

```
153
372
```

9.7 Output

Armstrong
Not Armstrong