Exercise 6: Arrays and 2-D Arrays

Nanda H Krishna

15 March 2018

1 Boolean Functions

Problem description: Define Boolean functions `is_prime(n)` that tests whether a non-negative integer `n` is prime or not, `is_cube(n)` that tests whether number `n` is a perfect cube & `is_divisible_by (n, d)` that tests whether an integer `n` is divisible by integer `d`.

Specification: The functions `is_prime()` and `is_cube()` which take the number `n` as the input, the function `is_divisible()` takes 2 numbers `n`, `d` as the inputs. All return a boolean value to the calling function.

Prototype:

```
bool is_prime(int n)
bool is_cube(int n)
bool is_divisible(int n,int d)
```

Program Design: The program consists of 3 functions `is_prime(int n)`,`is_cube(int n)` and `is_divisible(int n,int d)` which check the condition and return a value, while `main()` reads numbers from `stdin` and calls the functions to test them.

Algorithm:

```
is_prime(n):
   flag = true
   for i in range(2, n):
      if n % i == 0:
         flag = false
         break
   return flag
is_cube(n):
   flag = false
   i = 1
   while i^3 <= n:
```

```
      if i^3 == n:
         flag = true
         break
   return flag
is_divisible(n, d):
   flag = false
   if n%d == 0:
      flag = true
   return flag
```

Program:

```
#include<stdio.h>
#include<stdbool.h>
bool is_prime(int n)
{
    int i;
    bool flag = true;
    for(i = 2; i < n; i++) {
      if(n % i == 0) { flag = false; break; }
    }
    return flag;
}
bool is_cube(int n)
{
    int i = 1;
    bool flag = false;
    while(i*i*i <= n) {
      if(i*i*i == n) { flag = true; break; }
      i++;
    }
    return flag;
}
bool is_divisible(int n, int d)
{
    bool flag = false;
    if (n % d == 0) {
      flag = true;
    }
    return flag;
```

```
}
int main()
{
   int a, b, c, d;
   scanf("%d%d%d%d", &a, &b, &c, &d);

   if(is_prime(a)) printf("Prime\n");
   else printf("Not prime\n");

   if(is_cube(b)) printf("Cube");
   else printf("Not a cube");

   if(is_divisible(c, d)) printf("Divisible");
   else printf("Not divisible");

   return 0;
}
```

Test Input:

```
13
210
49 7
```

Output:

```
Prime
Not a cube
Divisible
```

2 Sorting

Problem description: Sort the list of numbers based on their weights, where the weight of a number is defined as:

$$\text{weight}(n) = \begin{cases} 3 & n \text{ is prime.} \\ 4 & n \text{ is a multiple of 4 and divisible by 6.} \\ 5 & n \text{ is a perfect cube.} \end{cases}$$

Specification: The functions `is_prime()`, `is_cube()` take the number `n` as the input and the function `is_divisible()` takes 2 numbers `n`, `d` as the inputs, and all return a boolean value to the calling function. Function `weight_calc()` takes arrays `a`, `weight`

and length of array `n` as inputs, and assigns values to `weight` as per the conditions. Function `swap()` takes array `a` and 2 indices `m`, `n` as inputs and swaps the 2 numbers. Function `selection_sort()` takes 2 arrays `a`, `b` and length of array `n` as inputs and sorts the array in ascending order.

Prototype:

```
bool is_prime(int n)
bool is_cube(int n)
bool is_divisible(int n, int d)
void weight_calc(int a[], int weight[], int n)
void swap(int a[], int m, int n)
void selection_sort(int a[], int b[], int n)
```

Program Design: The program consists of functions `is_prime(int n)`, `is_cube(int n)` , `is_divisible(int n, int d)` which check the condition and return a value, `weight_calc(int a[], int weight[], int n)` which assigns the values to `weight` array based on the condition, `swap(int a[], int m, int n)` which swaps 2 numbers, `selection_sort(int a[],int b[], int n)` which sorts the array in ascending order and `main()` which reads the numbers from `stdin` and calls the functions, with results being printed on `stdout`.

Algorithm:

```
is_prime(n):
   flag = true
   for i in range(2, n):
      if n % i == 0:
         flag = false
         break
   return flag
is_cube(n):
   flag = false
   i = 1
   while i^3 <= n:
      if i^3 == n:
         flag = true
         break
   return flag
is_divisible(n, d):
   flag = false
   if n%d == 0:
      flag = true
```

```
   return flag
weight_calc(a[], weight[], n)
   for i in range(n):
      t = is_prime(a[i])
      u = is_cube(a[i])
      v = is_divisible(a[i], 12)
      if t == true:
         weight[i] = 3
      else if v == true:
         weight[i] = 4
      else if u == true:
         weight[i] = 5
      else:
         weight[i] = 0
swap(a[], m, n):
   t = a[m]
   a[m] = a[n]
   a[n] = t
selection_sort(a[],b[],n):
   for i in range(n-1):
      m = i
      for j in range(i+1, n):
         if a[j] < a[m]:
            m = j
      swap(a, m, i)
      swap(b, m, i)
```

Program:

```
#include<stdio.h>
#include<stdbool.h>
bool is_prime(int n)
{
    int i;
    bool flag = true;
    for(i = 2; i < n; i++) {
      if(n % i == 0) { flag = false; break; }
    }
    return flag;
}
```

```
bool is_cube(int n)
{
    int i = 1;
    bool flag = false;
    while(i*i*i <= n) {
      if(i*i*i == n) { flag = true; break; }
      i++;
    }
    return flag;
}
bool is_divisible(int n, int d)
{
    bool flag = false;
    if (n % d == 0) {
      flag = true;
    }
    return flag;
}
void weight_calc(int a[], int weight[], int n)
{
    bool t, u, v, w;
    int i;
    for(i = 0; i < n; i++) {
      t = is_prime(a[i]);
      u = is_cube(a[i]);
      v = is_divisible(a[i], 12);
      if (t == true) weight[i] = 3;
      else if(v == true) weight[i] = 4;
      else if(u == true) weight[i] = 5;
      else weight[i] = 0;
    }
}
void swap(int a[], int m, int n)
{
    int t = a[m];
    a[m] = a[n];
    a[n] = t;
}
```

```
void selection_sort(int a[],int b[], int n)
{
    int i, j, m;
    for (i = 0; i < n - 1; i++) {
      m = i;
      for (j = i + 1; j < n; j++) if(a[j] < a[m]) m = j;
      swap(a, m, i);
      swap(b, m, i);
    }
}
int main()
{
    int a[10];
    int weight[10], b[10], i;
    for(int i = 0; i < 10; i++) scanf("%d", &a[i]);
    weight_calc(a, weight, 10);
    for(i = 0; i < 10; i++) {
      printf("%d ", weight[i]);
    }
    selection_sort(weight, a, 10);
    printf("\n");
    for(i = 0; i < 10; i++) {
      printf("%d ", weight[i]);
    }
    printf("\n");
    for(i = 0; i < 10; i++) {
      printf("%d ", a[i]);
    }
    return 0;
}
```

Test Input:

```
11 12 8 10 9 6 2 24 7 216
```

Output

```
3    4    5    0    0    0    3    4    3    4
0    0    0    3    3    3    4    4    4    5
10   9    6    11   2    7    12   24   216  8
```

3 Number of above-average quantities

Problem description: Populate an array `heights[N]` with heights of persons and find how many persons are above the average height.

Specification: The inputs recieved are the array `height` with heights of people, while the output is the number of people above the average height.

Program Design: The program consists of `main()`, which reads the input from stdin, finds the average, finds number of people above average height, and prints it on `stdout`.

Algorithm:

```
s = 0
c = 0
for i in range(n):
   s += a[i]
avg = sum / n
for i in range(n):
   if a[i] > avg:
      c++
```

Source Code:

```
#include<stdio.h>
int main()
{
    int i, n, count = 0;
    float sum = 0, avg, height[100];
    scanf("%d", &n);
    for(i = 0; i < n; i++) {
       scanf("%f", &height[i]);
       sum = sum + height[i];
    }
    avg = sum/n;
    for(i = 0; i < n; i++) if(height[i] > avg) count++;
    printf("%d", count);
    return 0;
}
```

Test Input:

```
10
172 186 154 123 145 166 169 150 140 177
```

Output:

```
5
```

4 BMI calculation

Problem description: Populate a two dimensional array `a[N][N]` with heights and weights of persons and compute the Body Mass Index (BMI) of the individuals. `a[i][0]` and `a[i][1]` are the height and weight of `i` th person. BMI is defined as

$$\text{BMI} = \frac{\text{weight}}{\text{height}^2}$$

where weight is in kg and height is in m.

Specification: The inputs are the height in metres and the weight in kg, received in a 2-D array. The output is the BMI for each set of data.

Program Design: The program consists of `main()`, which gets the input from `stdin`, finds the BMI and prints the output on `stdout`.

Algorithm:

```
for i in range(n):
   bmi[i] = a[i][1]/(a[i][0] * a[i][0])
```

Source Code:

```
#include<stdio.h>
int main()
{
    int i, j, n;
    float bmi[10], a[10][2];
    scanf("%d", &n);
    for(i = 0; i < n; i++) {
       for(j = 0; j < 2; j++) scanf("%f", &a[i][j]);
    }
    for(i = 0; i < n; i++) {
       bmi[i] = (a[i][1]/(a[i][0]*a[i][0]));
       printf("%f\n", bmi[i]);
    }
    return 0;
}
```

Test Input:

```
5
1.72 65
1.77 70
1.54 60
1.86 86
1.70 75
```

Output:

```
21.971336
22.343515
25.299377
24.858366
25.951555
```