Exercise 4: Iterative statements

PRAVEEN KUMAR R

27/2/18

Assignment	4
Reg No	312217104114
Name	PRAVEEN KUMAR R
Grade	
Date	27-02-2018

1 Indenting using iteration

Problem description:

Define a function `indent()` to print n times the pattern `|--` in a line where n is given as a parameter to the function

Specifications:

function called `indent()` to print the pattern `|--` n times followed by the key n `input:` key ,value from the standard input `output:` `|--|--...` for n times `key`

Program code:

```
#include<stdio.h>
void indent(int level,int key)
{
  for(int i=1;i<=level;i++)
    printf("|--");
  printf("%d\n",key);
}
int main()
{
  int a,a1;
  while( scanf("%d%d",&a,&a1)!=EOF)
    {
      indent(a,a1);
    }
}
```

Test

Input

```
2 10
3 45
6 60
7 7
```

Output

```
|--|--10
|--|--|--45
|--|--|--|--|--|--60
|--|--|--|--|--|--|--7
```

2 Length of the array

Problem description:

define a function `array_len()` to find the length of the array terminated by -1

Specifications:

function called `array_len()` to find out the length of an array terminated by -1 `input`: an array terminated by -1 `output`: length of the array

Program code:

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

```
    }

  printf("%d\n",arraylen(a));
}
```

Test

Input

4 63 19 4 3 6 8 0 2 8 -1

Output

```
8
```

Sample input:

4 3 6 8 0 2 8 -1

Sample output:

8

3 Print a subarray

Problem description

write a function `print_array()` that prints a subarray `a[low:high]` that is items from low to high

Specifications

function called `print_array()` to print the subarray from low to high `input`: array,lower bound,upper bound `output`: subarray `a[low:high]`

Program code

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

```
    scanf("%d",&a[i]);
  scanf("%d%d",&low,&high);
  print_array(a,low,high);
}
```

Test

Input

```
10 23 45 22 11 233 45453  221 344 4565 223
2 8
```

Output

22 11 233 45453 221 344

Sample input

1 2 3 4 5 6 7 3 5

Sample output

4 5

4 Sum,mean,variance

Program description

Read an array of numbers from standard input. Define `sum()`,`variance()` and `mean()`, to calculate the sum ,variance and mean of the given set of numbers. Test these functions from the `main()`.

Specification

- `sum()` – Input: a[0:n], anarray of integers and n, number of elements in the array. – Output: sum of the set of numbers.
- `variance()` – Input: a[0:n], anarray of integers and n, number of elements in the array. – Output: variance of the set of numbers.
- `mean()` – Input: a[0:n], anarray of integers and n, number of elements in the array. – Output: mean of the set of numbers.

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
```

Program

```
#include<stdio.h>
int sum(int a[],int n);
float variance(int a[],int n);
float mean(int a[],int n);
int sum(int a[],int n)
{
  int sum=0;
  for(int i=0;i<n;i++)
    {
      sum+=a[i];
    }
  return sum;
}
float mean(int a[],int n)
{
  return sum(a,n)/((float)(n));
}
float variance(int a[],int n)
{
  float s=mean(a,n);
  float k=0;
  for(int i=0;i<n;i++)
    {
      float p=a[i]-s;
      k=k+p*p;
    }
  return k/n;
}
int main()
{
  int a[100],n;
  scanf("%d",&n);
  for(int i=0;i<n;i++)
    scanf("%d",&a[i]);
  printf("sum=%d\nmean=%f\nvariance=%f",sum(a,n),mean(a,n),variance(a,n));
}
```

Test

Input

10 46 38 283 54 23 239 46 224 24 245

Output

```
sum=1222
mean=122.199997
variance=10777.959961
```

5 Prime numbers

Problem description:

write a boolean function `is_prime()` to check whether a number is prime or not and prints true or false

Specifications:

function `is_prime()` to check whether a number is prime or not and check the first 100 integers `input`: an integer n `output`: true if the number is prime , flase if the number is not prime

Algorithm

```
def is_prime(a):
   i=2,f=1
   while i<a/2:
      if a%i==0:
         f=0
         break
      i++
   return f
```

program code:

```
#include<stdio.h>
#include<stdbool.h>
bool is_prime(int n)
{
  int flag=1;
  for(int i=2;i<n;i++)
    {
      if(n%i==0)
flag =0;
    }
  if(flag==1)
    return true;
  else
    return false;
}
int main()
{
  int a[100];
  int n;
  for(n=0;scanf("%d",&a[n])!=EOF;n++);
  for(int i=0;i<n;i++)
    {
      printf("%d :",a[i]);
      if(is_prime(a[i]))
printf("true");
      else
printf("false");
      printf("\n");
    }
}
```

```
  23   :true
  45   :false
  75   :false
  32   :false
 234   :false
  24   :false
 111   :false
3545   :false
```

Test

Input

23 45 75 32 234 24 111 3545

Output

```
  23  :true
  45  :false
  75  :false
  32  :false
 234  :false
  24  :false
 111  :false
3545  :false
```

6 Linear search

Problem description:

define a function `linear_search()` to search a target in an array and return the index if the item is present or else return an invalid index

Algorithmic process:

- compare each item in the array with the target element
- If they are equal then return the index thereby breaking the loop.
- If the target is not in the array then the loop will end when control variable reaches an invalid index

then the function will return an invalid index

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
```

Specifications:

`linear_search()`: to search for a target in an array and return the index of the array if present or invalid number if the target is not present

- `input`: a[0:n],an integer array, n ,target
- `output`: index of target or the length of the array(invalid)

Program code

```
#include<stdio.h>
int linear_search(int a[],int n,int t)
{
  int i;
  for(i=0;i<n;i++)
  {
    if(a[i]==t)
      break;
  }
  return i;
}
int main()
{
  int n,t,a[100];
  scanf("%d",&n);
  for(int i=0;i<n;i++)
    scanf("%d",&a[i]);
  scanf("%d",&t);
  printf("%d\n",linear_search(a,n,t));
}
```

Test

Input

```
7
3 42 4 42 22 112  44
112
```

Output

```
5
```

7 Minimum

Problem description:

write a function `min()` that returns the index of the smallest item in the array

Algorithmic process:

- Assume first element of the array as the minimum.
- Compare each item in the array with the minimum assumed .
- if they are smaller than the minimum then assign that item as the minimum and continue the loop.
- After each iteration the minimum variable will have the minimum value of the subarray a[0,i].
- After loop ends the variable will have the minimum number of the array

Algorithm

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

Specifications:

`min()` to return the index of the minimum item in the array `input`: a[0:n], an array of integers and n `output`: i such that a[i]<=a[0:n]

Program code:

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

Test

Input

```
10
33 5432 4254 545 211 45 125 44 533 65672
```

Output

```
33
```

8 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.

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.

Prototype

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

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`.

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
```

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");
  }
}
```

Test

Input

```
1634
345
```

Output

Armstrong
Not Armstrong