Exercise 5: Arrays

R Ram Kaushik

April 3, 2018

1 Polynomial evaluation

A polynomial $a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1x + a_0$ is represented by an array `a[0:n]` of its coefficients. Write a program to compute the value of a polynomial using Horner's rule. The crux of the algorithm is: (*accumulator*) **Algorithm development**

```
s = 0
for i = n-1 down to 0:
  s = s * x + a[i]
```

Compare the algorithm with the algorithm for summing the items of an array.

1.1 Specification

A function `polynomial()` which takes an array `a[]`, it's length `n` and `x` as inputs and returns the sum to the calling function. ** Prototype

```
int polynomial(int a[], int x, int n)
```

1.2 Prototype

```
int polynomial(int a[], int x, int n)
```

1.3 Program Design

The program has function `polynomial(int a[], int x, int n)` which finds the sum and returns it, and `main()`, which reads the input from `stdin` , calls the function and prints the output on `stdout`.

1.4 Algorithm

```
def polynomial(a[],x,n):
   s=0
   for i in range(n-1,0,-1):
      s=s*x+a[i]
   return s
```

1.5 Source Code

```
#include<stdio.h>
int polynomial(int a[], int x, int n){
  int s=0;
  for(int i=n-1;i>=0;i--){
    s=s*x+a[i];
  }
  return s;
}
int main(){
  int a[20],n,x;
  scanf("%d",&n);
  for(int i=0;i<n;i++){
    scanf("%d",&a[i]);
  }
  scanf("%d",&x);
  int m=polynomial(a,x,n);
  printf("%d\n",m);
}
```

1.6 Test Input

```
4
1 2 5 1
3
```

1.7 Output

```
79
```

2 Binary search

We are given a sorted array of numbers. Define a function

```
binary_search(a, n, target)
```

that searches for `target` in `a[0:n]` using binary search algorithm. Let the function return an index `i` such that (*search*)

```
a[0:i] < target <= a[i:n]
```

2.1 Specification

A function binary_search() which takes a sorted array a[], the length n and the element to be searched t as inputs and returns the index to the calling function.

2.2 Prototype

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

2.3 Program Design

The program consists of a function binary_search(int a[], int t, int n) which returns an index based on the condition, and main(), which gets the input from stdin, calls the function and prints the value on stdout.

2.4 Algorithm

```
def binary_search(a[],t,n):
   b,e=0,n
   while b!=e:
      m=(b+e)/2
      if a[m]<t:
         b=m+1
      else:
         e=m
   return b
```

2.5 Source Code

```
#include<stdio.h>
int binary_search(int a[], int t, int n){
  int b=0,e=n,m;
  while(b!=e){
    m=(b+e)/2;
    if(a[m]<t)
      b=m+1;
    else
      e=m;
  }
  return b;
}
int main(){
  int a[20],n,t;
  scanf("%d",&n);
  for(int i=0;i<n;i++){
    scanf("%d",&a[i]);
  }
  scanf("%d",&t);
  int m=binary_search(a,t,n);
  printf("%d",m);
}
```

2.6 Test Input

```
10
0 10 12 21 24 39 45 53 75 89
14
```

2.7 Output

```
3
```

3 Selection sort

Selection sort is an algorithm for sorting an array of items, say `a[0:n]`. The idea of the algorithm is expressed below:

```
swap a[0], a[minimum(a,0,n)]
swap a[1], a[minimum(a,1,n)]
swap a[2], a[minimum(a,2,n)]
...
swap a[n-2], a[minimum(a,n-2,n)]
```

which uses `minimum(a, i, n)` to find the minimum of a subarray `a[i:n]`.

```
selection_sort (a, 0, n):
   for i = 0 to n-2:
      swap a[i], a[minimum(a, i, n)]
```

Implement selection sort, using `minimum()` function. Note: remember that when a function changes the items of an array parameter, the changes are effected in the items of the actual array argument also.

Test the function from `main()` for several lists of numbers. Each test should read a list of numbers from stdin.

3.1 Specification

2 functions `min()`, which takes array `a[]`, start index `l`, and end index `h` as inputs and returns the index of smallest number, and `selection_sort()`, which takes array `a[]`, length `n` as inputs and sorts the array in ascending order.

3.2 Prototype

```
int min(int a[], int l, int h)
void selection_sort(int a[], int n)
```

3.3 Program Design

The progam consists of 2 functions `min(int a[], int l, int h)`, which finds the index of the smallest number of the array within l and h and returns it, `selection_sort(int a[], int n)`, which sorts the array in ascending order, and `main()`, which gets the input from `stdin`, calls the function and prints the output on `stdout`.

3.4 Algorithm

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

3.5 Source Code

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

```
void selection_sort(int a[], int n){
  int m,t;
  for(int i=0;i<n;i++){
    m=min(a,i,n);
    t=a[i];
    a[i]=a[m];
    a[m]=t;
  }
}
int main(){
  int a[20],n;
  scanf("%d",&n);
  for(int i=0;i<n;i++){
    scanf("%d",&a[i]);
  }
  selection_sort(a,n);
  for(int i=0;i<n;i++){
    printf("%d ",a[i]);
  }
}
```

3.6 Test Input

```
11 12 1 6 67 34 15 23 56 32
```

3.7 Output

```
1 6 11 12 15 23 32 34 56 67
```

4 Polish National Flag (PNF)

In an array of items `a[low:high]`, each item is either positive or negative. Define a function `partition(a, low, high)` that partitions the array into two subarrays `a[low:i]` and `a[i:high]` such that all the negative items of the array form `[low:i]`, and all the positive items form `[i:high]`. Test the function from `main()`. Use several lists of numbers for testing. (Note: We will use this algorithm for implementing `quicksort()`.)

4.1 Specification

A function `pnf()`, which takes array `a[l:h]` as input and returns the index of the last negative number in the new array.

4.2 Prototype

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

4.3 Program Design

The program has a function `pnf(int a[], int l, int h)` which returns the index of the last negative number in the new array, and `main()`, which gets the input from `stdin`, calls the function and prints the value on `stdout`.

4.4 Algorithm

```
def pnf(a[],low,high):
   i,p=l,l
   while i<h:
      if a[i]<0:
         a[i],a[p]=a[p],a[i]
         p+=1
      i+=1
   return p
```

4.5 Source Code

```
#include<stdio.h>
int pnf(int a[], int l, int h){
  int i=l,p=l;
  while(i<h){
    if(a[i]<0){
      int t=a[i];
      a[i]=a[p];
      a[p]=t;
      p++;
    }
    i++;
  }
  return p;
}
int main(){
  int a[20],n;
  scanf("%d",&n);
  for(int i=0;i<n;i++){
    scanf("%d",&a[i]);
  }
  printf("\n");
  int p=pnf(a,0,n);
  for(int j=0;j<p;j++){
    printf("%d ",a[j]);
  }
  printf("\n");
  for(int j=p;j<n;j++){
    printf("%d ",a[j]);
  }
}
```

4.6 Test Input

```
20 -8 56 45 -90 21 -7 1 -3 5
```

4.7 Output

```
-8  -90  -7  -3
20   21  56   1  45  5
```

5 Dutch National Flag (DNF)

Dutch National Flag (DNF) is similar to PNF, but partitions the array `a[l:h]` into three subarrays `[l:i]`, `[i:j]` and `[j:h]`. Each item of the array has one of the three properties. Items having the same property should form one subarray each.

5.1 Specification

2 functions `print(a[l:h])`, used to print the array, `dnf()` which takes array `a[l:h]` and `c` as inputs and arrange the array based on `c`.

5.2 Prototype

```
void print(char a[], int l, int h)
int dnf(char a[], int l, int h, char c)
```

5.3 Program Design

The program contains 2 functions `print(char a[], int l, int h)`, which prints the array, `dnf(char a[], int l, int h, char c)`, which returns the index upto which the array has been rearranged, and `main()` which gets input from `stdin` and calls the functions.

5.4 Algorithm

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

```
def dnf(a[],l,h,c):
    i,p=l,l
    while i<h:
        if a[i]==c:
            a[i],a[p]=a[p],a[i]
            p+=1
        i+=1
```

5.5 Source Code

```
#include<stdio.h>
#include<string.h>
void print(char a[], int l, int h){
  for(int i=l;i<h;i++){
    printf("%c ",a[i]);
  }
}
int dnf(char a[], int l, int h, char c){
  int i=l,p=l;
  while(i<h){
    if(a[i]==c){
      char t=a[i];
      a[i]=a[p];
      a[p]=t;
      p++;
    }
    i++;
  }
  return p;
}
int main(){
  char a[50],c,d;
  int n,p,q;
  scanf("%s",a);
  n=strlen(a);
  scanf("%c%c",&c,&d);
  printf("\n");
  p=dnf(a,0,n,c);
  print(a,0,p);
  printf("\n");
  q=dnf(a,p,n,d);
  print(a,p,q);
  printf("\n");
  print(a,q,n);
  printf("\n");
}
```

5.6 Test Input

```
aaaaabbbccbbccaccacbbac
b
c
```

5.7 Output

```
            b  b  b  b  b  b  b
            a  c  c  a  a  c  c  a  c  c  a  c  a  a  a  c
```