1.1

```
#include <stdio.h>

int factorial(int N)  //pass by value

{

    int result = 1;

    for(int i = 2; i < N+1; i++){

        result *= i;

    }

    return result;

}


int power(int x, int a)  //pass by value

{

    if(a == 0){

        return 1;

    }

    int result = 1;

    for(int i = 0; i < a; i++){

        result *= x;

    }

    return result;

}

//for pass by reference: give the parameters with their addresses
(&)

int main() {

    // Write C code here

    int x, a, N;

    printf("Enter x: ");

    scanf("%d", &x);

    printf("Enter a: ");

    scanf("%d", &a);
    printf("Enter N: ");
    scanf("%d", &N);
    float sum = 0;
    for(int i = 1; i < N+1; i++){
        int inner = power(x, a);
        int outer = power(inner, i);
        int fac = factorial(i);
        sum += outer/fac;
    }
    printf("Number = %f",sum);

    return 0;
}
```

2.1

Pointers are used to allocate memory dynamically, they provide faster access to the memory and increase the efficiency. Whereas & returns the address of the pointer, * returns the value pointed by that pointer.

2.2

```
#include <stdio.h>

int main()
{
    int *pa; //pointer declared
    int NumArray[11];  //declaring an array with 11 elements

    for (int n = 0; n < 11; n++){  //all elements of the array are initialized as -1
        NumArray[n] = -1;
    }
    pa = NumArray;    *pa = 3;  //element at index 0 becomes 3
    pa++;             *pa = 14;  //element at index 1 becomes 14
    pa = NumArray;    *(pa+2) = 5;  //element at index 2 becomes 5
    pa = &NumArray[3]; *pa = 16;  //element at index 3 becomes 16
    pa = NumArray + 5;  *pa = 8; //element at index 5 becomes 8
    pa = NumArray + 4;  *pa = 17;  //element at index 4 becomes 17
    pa = NumArray;      *(pa+6) = 8; //element at index 6 becomes 8

    for(int n = 0; n < 11; n++){  //printing the content of the array in one line
        printf("%d", NumArray[n]);
    }

    int *pb = &NumArray[3];  //another pointer pointing to the index 3 (16)
    printf("\n %d %d \n", *pa, *pb);  //pa was initialized as NumArray so it points to the first element

    *pb = *pa + *(pa+2);  //value of the pb becomes 3 + 5 = 8, so NumArray[3] becomes 8 as well

    *(pa-3) = *py;  //out of boundary and py is not defined, it would give an error

    printf("\n %d %d \n", *pa, *py);

    for(int n = 0; n < 11; n++)  //printing the content of the array in one line
    {
        printf("%d\n", NumArray[n]);
    }
    return 0;
}
```

2.3

```
void swapNums(int x, int y) { //version1
  int z = x;
  x = y;
  y = z;
}

void swapNums(int &x, int &y) {  //version2
  int temp = x;
  x = y;
  y = temp;
}
```

3.a

```
struct resident{
    char residentID[11];
    int age;
    int nad;
    int ndvt;
    float grade;
};
```

3.b

```
resident* allocateMemory(){
    resident* arr = (resident)malloc(sizeof(resident) * 80000000);
    return arr;
}
```

3.c

3.d

```
#include <math.h>
void calculateGrade(struct resident r, int a, int b)
{
    float grade;
    grade = 0.9 * (0.7 * pow(r->nad, a) + 0.3 * pow(r->nvdt, b)) +
0.1 * (100 - r->age);
    r->grade = grade;
}
```

3.e

```
void sort(resident* arr){  //insertion sort O(n^2)
    int i, key, j;
    for (i = 1; i < n; i++) {
        key_id = arr[i]->residentID;
        key_age = arr[i]->age;
        key_nad = arr[i]->nad;
        key_ndvt = arr[i]->ndvt;
        key_grade = arr[i]->grade;
        j = i - 1;
        while (j >= 0 && arr[j]->grade > key) {
            arr[j + 1]->residentID = arr[j]->residentID;
            arr[j + 1]->age = arr[j]->age;
            arr[j + 1]->nad = arr[j]->nad;
            arr[j + 1]->ndvt = arr[j]->ndvt;
            arr[j + 1]->grade = arr[j]->grade;
            j = j - 1;
        }
        arr[j + 1]->residentID = key_id;
        arr[j + 1]->age = key_age;
        arr[j + 1]->nad = key_nad;
        arr[j + 1]->ndvt = key_ndvt;
        arr[j + 1]->grade = key_grade;
    }
}
```