1. How to prevent the variable a be changed?

```c
void func(int *p)
{
    *p += 1;
    return;
}

int main(void)
{
    int a = 2;
    func(&a);
    printf("%d\n", a);    // 3

    return 0;
}
```

Method	Use Case
const int *p (function argument)	When passing large data with read only access
int p (pass by value)	When passing small data type int, char, float etc.

2. Supplement the corresponding formal parameters and add the corresponding actual parameters. List all the forms that you know. Refer to the one-dimensional array a

```c
void funca1(int a[]){}
void funca2(int a[ARR_NUM]){}
void funca3(int *a){}

void funcb(){}
void funcc(){}
void funcd(){}
void funce(){}

int main(int argc, char *argv[], char *envp[])
{
    int a[ARR_NUM] = {1, 2, 3};
    int *b[5];
    int (*c)[5];
    int d[4][5];
    int **e;

    funca1(a);
    funca2(a);
    funca3(a);

    funcb(b);
    funcc(c);
    funcd(d);
    funce(e);

    return 0;
}
```

```c
#include <stdio.h>

#define ARR_NUM 3

// Typedefs for different pointer types
typedef int* IntPtr;
typedef int** IntPtrPtr;
typedef int (*ArrayPtr)[5];
typedef int* PtrArray[5];

// For array of pointers (a)
void funca1(int a[]) {}          // Equivalent to int *a
void funca2(int a[ARR_NUM]) {}   // Also equivalent to int *a
void funca3(int *a) {}           // Pointer notation (same as int a[])
void funca4(int a[3]) {}         // Fixed-size (still treated as int *)
void funca4(void *a) {}         // Fixed-size (still treated as int *)
void funca5(int (*a)) {}         // Explicit pointer notation

// For array of pointers (b)
void funcb1(int *b[5]);          // Array of pointers with size
void funcb2(int *b[]);           // Array of pointers
void funcb3(int **b);            // Pointer to pointer
void funcb4(int *(*b));          // Explicit pointer notation
void funcb5(void *b);            // Void pointer
void funcb6(PtrArray b);         // Typedef array of pointers
void funcb7(IntPtrPtr b);        // Typedef pointer to pointer

// For pointer to array (c)
void funcc1(int (*c)[5]);        // Pointer to array
void funcc2(int c[][5]);         // Array form
void funcc3(int c[3][5]);        // With first dimension
void funcc4(void *c);            // Void pointer
void funcc5(ArrayPtr c);         // Typedef pointer to array

```

```c

// For 2D array (d)
void funcd1(int d[4][5]);        // Full dimensions
void funcd2(int d[][5]);         // Partial dimensions
void funcd3(int (*d)[5]);        // Pointer to array
void funcd4(void *d);            // Void pointer
void funcd5(ArrayPtr d);         // Typedef pointer to array

// For pointer to pointer (e)
void funce1(int **e);            // Standard form
void funce2(int *(*e));          // Explicit pointer form
void funce3(void *e);            // Void pointer
void funce4(IntPtrPtr e);        // Typedef pointer to pointer

int main(int argc, char *argv[], char *envp[])
{
    int a[ARR_NUM] = {1, 2, 3};
    int *b[5];
    int (*c)[5];
    int d[4][5];
    int **e;

    // Passing a one-dimensional array `a` to different functions
    funca1(a);
    funca2(a);
    funca3(a);
    funca4(a);
    funca4(a);
    funca6(a);

    // Array of pointers (b) calls
    funcb1(b);
    funcb2(b);
    funcb3(b);
```

```
63
64      // Array of pointers (b) calls
65      funcb1(b);
66      funcb2(b);
67      funcb3(b);
68      funcb4(b);
69      funcb5(b);
70      funcb6(b);
71      funcb7(b);
72
73      // Pointer to array (c) calls
74      funcc1(c);
75      funcc2(c);
76      funcc3(c);
77      funcc4(c);
78      funcc5(c);
79
80      // 2D array (d) calls
81      funcd1(d);
82      funcd2(d);
83      funcd3(d);
84      funcd4(d);
85      funcd5(d);
86
87      // Pointer to pointer (e) calls
88      funce1(e);
89      funce2(e);
90      funce3(e);
91      funce4(e);
92
93      return 0;
94  }
95
```

3. Write a program lists out all the types of pointer step size that you know

```c
#include <stdio.h>

int main() {
    // Step 1: Declare arrays for demonstration
    char char_arr[]     = {'A', 'B'};
    int int_arr[]       = {1, 2};
    long long_arr[]     = {1L, 2L};
    float float_arr[]   = {1.0f, 2.0f};
    long long ll_arr[]  = {1LL, 2LL};
    double double_arr[] = {1.0, 2.0};

    // Step 2: Initialize pointers to arrays
    char *char_ptr      = char_arr;
    int *int_ptr        = int_arr;
    float *float_ptr    = float_arr;
    double *double_ptr  = double_arr;
    long *long_ptr      = long_arr;
    long long *ll_ptr   = ll_arr;


    // Step 3: Display step sizes using pointer arithmetic
    printf("Type        Step Size \n\n");

    // Calculate actual byte differences using pointer addresses
    printf("char      |    %u\n", (char*)(char_ptr + 1) - (char*)char_ptr);
    printf("int       |    %u\n", (char*)(int_ptr + 1) - (char*)int_ptr);
    printf("float     |    %u\n", (char*)(float_ptr + 1) - (char*)float_ptr);
    printf("double    |    %u\n", (char*)(double_ptr + 1) - (char*)double_ptr);
    printf("long      |    %u\n", (char*)(long_ptr + 1) - (char*)long_ptr);
    printf("long long |    %u\n", (char*)(ll_ptr + 1) - (char*)ll_ptr);

    return 0;
}
```

4. Complete the definition according to the description. using the variable ptr

description	type definition
A pointer to a char constant	const char *ptr;
A constant pointer to a char volatile	
A constant pointer to char	
An array of 10 pointers that point to an integer	
A pointer to an array of 10 integers	
A pointer to a function that takes an integer parameterand returns an integer	
An array of 10 pointers that point to a function thattakes an integer parameter and returns an integer	

Description	**Type Definitation**
A pointer to a char constant	const char *ptr;
A constant pointer to a char volatile	char volatile *const ptr;
A constant pointer to char	char *const ptr;
An array of 10 pointers that point to an integer	int *ptr[10];
A pointer to an array of 10 integers	int (*ptr)[10];
A pointer to a function that takes an integer parameter and returns an integer	int (*ptr)(int);
An array of 10 pointers that point to a function that takes an integer parameter and returns an integer	int (*ptr[10])(int);

5. Using typedef to simplify complicated declarations
    a. int (*(*AA)(void *))[10];
    b. void (*BB(int, void (*)(int)))(int);
    c. char (*(*CC[3])())[5]

```c
#include<stdio.h>

// A.
typedef int Array10[10];                        // Type for an array of 10 integers
typedef Array10 *PtrArray10;                    // Pointer to an array of 10 integers
typedef PtrArray10 (*FuncAA)(void *);           // Function pointer returning PtrArray10

FuncAA AA;                                      // Equivalent to: int (*(*AA)(void *))[10];


// B.
typedef void (*FuncIntVoid)(int);               // Function pointer for void func(int)
typedef FuncIntVoid (*FuncBB)(int, FuncIntVoid);// Function returning FuncIntVoid

FuncBB BB;                                      // Equivalent to: void (*BB(int, void (*)(int)))(int);


// C.
typedef char CharArray5[5];                     // Define an array of 5 chars
typedef CharArray5 *FuncCC(void);               // Function returning a pointer to array
typedef FuncCC *PtrFuncCC[3];                   // Array of 3 function pointers

PtrFuncCC CC;                                   // Equivalent to: char (*(*CC[3])())[5];

```