Argument passing in function

Exchange the value of two variables using a function

```
#include <stdio.h>
void swap(int, int);
int main()
{
   //int num1 = 5, num2 = 10;

   swap( num1, num2);

   printf("num1 = %d\n", num1);
   printf("num2 = %d", num2);
   return 0;
}
```

```
void swap(int n1, int n2)
{
    int temp;
    temp = n1;
    n1 = n2;
    n2 = temp;
}
```

Some operations are done internally –

Parameters are assigned with the arguments –

n1 = num1
n2 = num2

But parameters are different variable

If the function could have implemented the following operations automatically -

(Parameters are stored back in the arguments) -

num1 = n1

num2 = n2

The problem could have been solved

Unfortunately this is not done for various reasons !!

The expected outcome

Output –

num1 = 5

num2 = 10

Why ?

Arguments and Parameters are different variables

Values are copied from arguments to parameters when called

So, exchange happens in parameters which does not have any effect on the arguments

The way of argument passing we saw is called call-by-value

We need Call-by-address

```
#include <stdio.h>
void swap(int *n1, int *n2);

int main()
{
   int num1 = 5, num2 = 10;

   // address of num1 and num2 is passed
   swap( &num1, &num2);

   printf("num1 = %d\n", num1);
   printf("num2 = %d", num2);
   return 0;
}
```

- n1= & num1
- n2 = & num2

```
void swap(int* p1, int* p2)
{
    int temp;
    temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
```

num1 = 10

num2 = 5

Another way of doing the same swap – pass the pointers not address -

```
#include <stdio.h>
void swap(int *n1, int *n2);

int main()
{
   int num1 = 5, num2 = 10;
   int *ptr1 = &num1, *ptr2 = &num2;
   // address of num1 and num2 is passed

   swap( ptr1, ptr2);
   printf("num1 = %d\n", num1);
   printf("num2 = %d", num2);
   return 0;
}
```

Some operations are done internally –

Parameters are assigned with the arguments –

p1 = &num1
p2 = &num2

Now, after argument passing -
***p1 is same as num1 and *p2 is same as num2**

The address of num1 and num2 are passed to the swap() function using swap(&num1, &num2);.

Pointers n1 and n2 accept these arguments in the function definition.

```
void swap(int* n1, int* n2) {
    … ..
}
```

When *n1 and *n2 are changed inside the swap() function, num1 and num2 inside the main() function are also changed.

Inside the swap() function, *n1 and *n2 swapped. Hence, num1 and num2 are also swapped.

Notice that, swap() is not returning anything; its return type is void.

Call by address

```
#include <stdio.h>

void addOne(int* ptr) {
  (*ptr)++; // adding 1 to *ptr
}
```

```
int main()
{
  int *p, i = 10;
  p = &i;
  addOne(p);

  printf("%d", *p); // 11
  return 0;
}
```

Passing an array to a function

In C programming, you can pass an entire array to functions

```
#include <stdio.h>
void display(int age1, int age2)
{
    printf("%d\n", age1);
    printf("%d\n", age2);
}
```

```
int main()
{
   int ageArray[] = {2, 8, 4, 12};

   // Passing second and third elements to display()

   display(ageArray[1], ageArray[2]);
   return 0;
}
```

Pointer basics

1. About storing the address of the bytes inside a computer memory, The variable should have enough size – so that it can accommodate the maximum address possible in the system.

2. The pointer arithmetic should be supported …

3. * operation also has to be implemented …..

Conclusion: Pointers are fundamentally integers – with these facilities…..!!

Pointer arithmetic

```
int *p;
float *q;
p is 123478996 -- 123479000
p = p + 1;      // p should point to the next integer – if
                // in an array. 1 is equivalent to
                sizeof(int)
                        // p = p + sizeof(int)

q = q + 1;      // q should point to the next float – if
                // in an array., q = q + sizeof(float)
```

Pointer arithmetic

```
int *p;
float *q;
int x= …;
p = p+x;        // p should point to the next integer – if
                // in an array. 1 is equivalent to
                sizeof(int)
                // p = p + x*sizeof(int)

q = q + x;      // q should point to the next float – if
                // in an array., q = q + x*sizeof(float)
```

A Program to calculate the sum of array elements by passing to a function

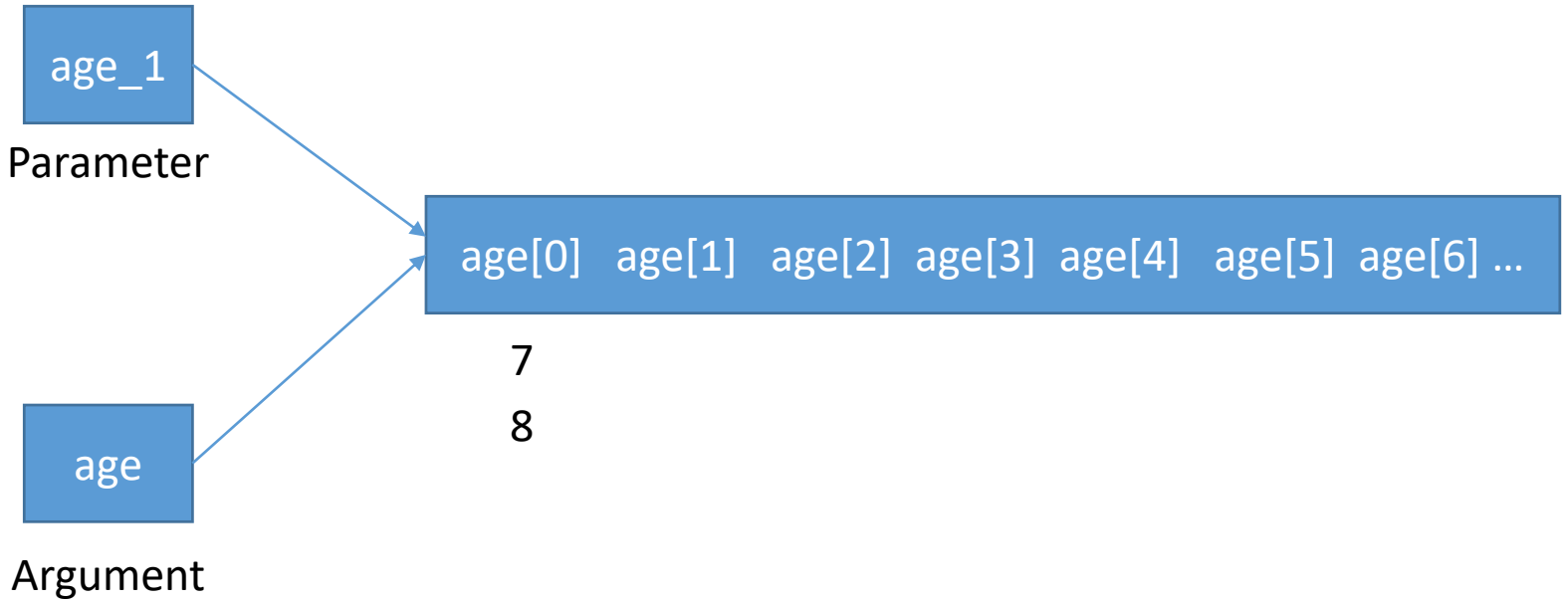
```
float calculateSum(float a[]) {
  float sum = 0.0;

  for (int i = 0; i < 6; ++i) {
              sum += a[i]; //
                    sum+=*(a+i)
      }
  return sum;
}
```

age[i]

in the function calculateSum---
is equivalent to

*(age+i)



Call by address is used, So, argument and parameters are pointing to the same array – or better to say the first element of the array.

age_1[0] = 7; age[0] = 8;

```
int main() {
    float result, age[] =
            {23.4, 55, 22.6, 3, 40.5, 18};

       result = calculateSum(age);
    printf("Result = %.2f", result);
    return 0;
}
```

Result = 162.50

Notes

To pass an entire array to a function, only the name of the array is passed as an argument.

result = calculateSum(age, 10);

Check the [] in the function definition

```
float calculateSum(float age[], int size) {
... ..
}
```

```
float calculateSum(float *a, int size) {
  float sum = 0.0;

  for (int i = 0; i < size; ++i) {
            sum += a[i];
      }
  return sum;
}
```

```
#include <stdio.h>

// Note that arr[] for fun is just a pointer even
if square brackets are used
void fun(int arr[])
// SAME AS void fun(int *arr)
{
  unsigned int n =
sizeof(arr)/sizeof(arr[0]);
  printf("\nArray size inside fun() is %d", n);
}
```

```
// Driver program
int main()
{
   int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};
   unsigned int n = sizeof(arr)/sizeof(arr[0]);
   printf("Array size inside main() is %d", n);
   fun(arr);
   return 0;
}
```

Output:

Array size inside main() is 8
Array size inside fun() is 1

Therefore in C, we must pass size of array as a parameter.

Size may not be needed only in case of ‘\0’ terminated character arrays, size can be determined by checking end of string character.

Passing array to function in C

Pointer to arr

Length of arr

```
void func( int a[] , int size )
{

}

int main( )
{
    int n=5;
    int arr[5] = { 1, 2, 3, 4, 5 };
    func( arr , n);
    return 0;
}
```

Pointer a takes the base address of array arr

The length of the array is passed

The array is passed as a pointer

Example

```
#include <stdio.h>

void fun(int *arr, unsigned int n)
{
   int i;
   for (i=0; i<n; i++)
     printf("%d  ", arr[i]);
}

// Driver program
int main()
{
   int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};
   unsigned int n = sizeof(arr)/sizeof(arr[0]);
   fun(arr, n);
   return 0;
}
```

Output:

1 2 3 4 5 6 7 8

Examples

```
#include <stdio.h>
void fun(int arr[], unsigned int n)
{
int i;
for (i=0; i<n; i++)
    printf("%d ", arr[i]);
}

// Driver program
int main()
{
int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};
unsigned int n = sizeof(arr)/sizeof(arr[0]);
fun(arr, n);
return 0;
}
```

```
#include <stdio.h>
void fun(int *arr)
{
   int i;
   unsigned int n = sizeof(arr)/sizeof(arr[0]);
   for (i=0; i<n; i++)
     printf("%d  ", arr[i]);
}

// Driver program
int main()
{
   int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};
   fun(arr);
   return 0;
}
```

```
#include <stdio.h>
#include <string.h>

void fun(char *arr)
{
   int i;
   unsigned int n = strlen(arr);
   printf("n = %d\n", n);
   for (i=0; i<n; i++)
     printf("%c  ", arr[i]);
}
```

```
// Driver program
int main()
{
   char arr[] = "geeksquiz";
   fun(arr);
   return 0;
}
```

```
#include <stdio.h>
#include <string.h>

void fun(char *arr)
{
   int i;
   unsigned int n = strlen(arr);
   printf("n = %d\n", n);
   for (i=0; i<n; i++)
     printf("%c  ", arr[i]);
}
```

```
// Driver program
int main()
{
   char arr[] = {'g', 'e', 'e', 'k', 's',
'q', 'u', 'i', 'z'};
   fun(arr);
   return 0;
}
```

Passing multi-dimensional arrays

Passing 2D array

Passing 3D array

….

Examples

Pass a 2D array to a function that can display all the numbers in the array.

displayNumber

int num[5][10][100]; 5* 10 * 100 = 5000;

5000 X sizeof(int)

int *p = num; p can be interpreted as a 1D array having 5000 elements.

Print p[0], …., p[4999], num[0][0][0], …, num[4][9][99]

P[101] → num[0][1][1] → 0 to 99 is the first 1D array. 100,101…

```
Int main(){

        int num[5][10][100];
        function(num);
}
Int function(int *p){

        How I can access the I, j , k th element using p;
        From I, j, k – apply a formula to get a value l which
        we can use as
        p[l] -> What is the formula ??
}
```

```
void displayNumbers(int num[2][2]);
```

```
void displayNumbers(int num[2][2])
{
    printf("Displaying:\n");
    for (int i = 0; i < 2; ++i) {
        for (int j = 0; j < 2; ++j) {
            printf("%d\n", num[i][j]);
        }
    }
}
```

```
int main()
{
    int num[2][2];
    printf("Enter 4 numbers:\n");
    for (int i = 0; i < 2; ++i)
        for (int j = 0; j < 2; ++j)
            scanf("%d", &num[i][j]);

    // passing multi-dimensional array to a function
    displayNumbers(num);
    return 0;
}
```

```
#include <stdio.h>
void displayNumbers(int num[2][2]);
int main()
{
   int num[2][2];
   printf("Enter 4 numbers:\n");
   for (int i = 0; i < 2; ++i)
      for (int j = 0; j < 2; ++j)
         scanf("%d", &num[i][j]);

   // passing multi-dimensional array
to a function
   displayNumbers(num);
   return 0;
}
```

```
void displayNumbers(int
num[2][2])
{
   printf("Displaying:\n");
   for (int i = 0; i < 2; ++i) {
      for (int j = 0; j < 2; ++j) {
         printf("%d\n", num[i][j]);
      }
   }
}
```

Output

```
Enter 4 numbers:
2
3
4
5
Displaying:
2
3
4
5
```

```
#include <stdio.h>
const int M = 3;
const int N = 3;

void print(int arr[M][N])
{
    int i, j;
    for (i = 0; i < M; i++)
      for (j = 0; j < N; j++)
        printf("%d ", arr[i][j]);
}
```

```
int main()
{
    int arr[][3] = {{1, 2, 3},
{4, 5, 6}, {7, 8, 9}};
    print(arr);
    return 0;
}
```

1 2 3 4 5 6 7 8 9

void print(int *arr)

void print(int arr[])

```
#include <stdio.h>
const int N = 3;

void print(int arr[][N], int m)
void print(int (*arr)[N], int m)
{
    int i, j;
    for (i = 0; i < m; i++)
      for (j = 0; j < N; j++)
        printf("%d ", arr[i][j]);
}
```

```
int main()
{
    int arr[][3] = {{1, 2, 3},
{4, 5, 6}, {7, 8, 9}};
    print(arr, 3);
    return 0;
}
```

Array Pointer / Pointer to an array

- int a[10]; a is an integer array
- What is a ?
  - a is a pointer to the first element of the array –
  - First element of the array is an integer
  - So, a is a pointer to an integer
  - But a is constant

- What is diff between int *p and a ??
- p = a; a = p;

Array Pointer / Pointer to an array

- int b[10][10]; a is a 2D integer array
- What is b ?
  - b is a pointer to the first element of the array –
  - First element of the array is an array of 10 integers
  - So, b is a pointer to an array of 10 integers
  - But b is constant

What is the difference – that an element x is pointing to a single integer or a chunk of N number of integers ??

Array Pointer / Pointer to an array

- Imagine X is a integer pointer
- Imagine Y is a pointer to a set of integers – say 10 integers
- Fundamentally – both allow storing of address of bytes.
- X++; X = X + sizeof(int)
- Y++; Y = Y + 10*sizeof(int)
- Y = Y + 2; Y = Y + 2*10*sizeof(int)

```
#include <stdio.h>
void print(int *arr, int m, int n)
{
    int i, j;
    for (i = 0; i < m; i++)
      for (j = 0; j < n; j++)
        printf("%d ", *((arr+i*n) + j)); //printf("%d ",*(arr+i*n +
j));
}

int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    int m = 3, n = 3;

    // We can also use "print(&arr[0][0], m, n);"
    print((int *)arr, m, n);
    return 0;
}
```

Pointer to an array is also known as array pointer

```
int a[3] = {3, 4, 5 };
int *ptr = a;
```

We have a pointer ptr that focuses to the 0th component of the array. We can likewise declare a pointer that can point to whole array rather than just a single component of the array.

Syntax:

data type (*var name)[size of array];

data type (*var name)[Chunk size];

```
int x = 10;

int *ptr1;
ptr1= &x;

int y[10];
int (*ptr2)[10];
ptr1 = y;          // No warnings will be there
ptr2 = y;          // Warning will be there
```

ptr1++

```
int x = 10;

int *ptr1;
ptr1= &x;

int y[100][10];
int (*ptr2)[10];
ptr1 = y;          // Here there warnings will be there
ptr2 = y;          // There will be no warning
```

ptr2++

```
int x = 10;

int *ptr1;
ptr1= &x;
int y[100][10];
int (*ptr2)[10];
ptr1 = y;          // Here there warnings will be there
ptr2 = y;          // There will be no warning

Y[0][4];
ptr2[0][4];        == *(ptr2+4)
```

```
int (* ptr)[5] = NULL;
```

Pointer to an array of five integers.

Parenthesis to pronounce pointer to an array.

Subscript has higher priority than indirection, it is crucial to encase the indirection operator and pointer name inside brackets.

```
// C program to demonstrate
// pointer to an array.

#include <stdio.h>

int main()
{
   // Pointer to an array of five numbers
   int *a;
   int b[5] = { 1, 2, 3, 4, 5 };

   int i = 0;
   // Points to the whole array b

   a = &b[0];
   a = b;

   for (i = 0; i < 5; i++)

      printf("%d\n", *(a));

   return 0;
}
```

```
1
2
3
4
5
```

“Array of pointers” is an array of the pointer variables.

It is also known as pointer arrays.

```
int *var_name[array_size];
```

```
int *ptr[3];
```

```
// C program to demonstrate
// example of array of pointers.

#include <stdio.h>

const int SIZE = 3;

void main()
{

    // creating an array
    int arr[] = { 1, 2, 3 };

    // we can make an integer pointer array to
    // storing the address of array elements
    int i, *ptr[SIZE];

    for (i = 0; i < SIZE; i++) {

        // assigning the address of integer.
        ptr[i] = &arr[i];
    }

    // printing values using pointer
    for (i = 0; i < SIZE; i++) {

        printf("Value of arr[%d] = %d\n", i,
*ptr[i]);
    }
}
```

Output:

```
Value of arr[0] = 1
Value of arr[1] = 2
Value of arr[2] = 3
```

Swap two pointers in the main functions using a functions

```
main(){

        int a = 10, b = 20;
        int *x=&a, *y=&b;

        swap(&a,&b);  // No use        - will not work
        swap(x,y);       // No use        - will not work

        print *x→ should be pointing to b;
        and print *y → y should be pointing to a;
}
```

Swap two pointers in the main functions using a functions

```
swap(int **ax, int **ay){
        int *temp;

        temp = *ax;
        *ax=*ay;
        *ay = temp;
}
```

```
swap(&x, &y);
```

X→ b;

Y→ a;

We want to exchange the pointer values.

How to return more than one parameters

Write a function that will return five parameters to the main function ….

```
main(){

        int x,y,z,a,b;

        x = function(&y, &z, &a, &b);

        x, y, z, a, b – all will get values from the function

}
```

How to return more than one parameters

Write a function that will return five parameters to the main function ….

```
int function(int *y, int *z, int *a, int *b){

        y = 10;
        a = 20;
        b = 30;
        Z = 40;
        return 50;
}
```

How to return more than one parameters

Write a function that will return five parameters to the main function ….

```
void *function(int *a){

        //int *a;   //use malloc to declare an array ;
        //a = (int *)malloc(sizeof(int)*10);

        //store the values in the array;
        a[0] = 1; a[1] = 2 …….

        //return the address of the array
        //return a;
}
```