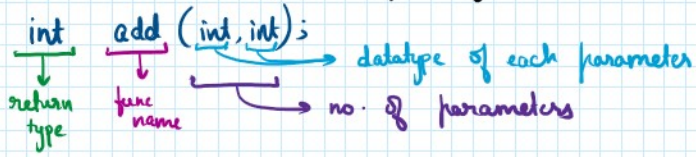
1. Functions, Arrays

28 February 2024 08:46

FUNCTIONS

Function declaration / prototype / signature



You can also name the parameters:

```
int add (int a, int b);   ✓
int add (int a, b);       X   "b" does not have a type specified
```

C does not support function overloading; you cannot have multiple functions with the same name.

Function that does not return any value, does not take any parameters:

```
void func (void);
```

Function call

arguments / actual parameters

```
int add (int a, int b);
add (a, b);   → function call
```

Function definition

parameters / formal parameters

```
int add (int a, int b)   → header
{
return a+b;
}
```

NOTE:
Functions cannot be redefined.

NOTE: Const

To avoid changes to arguments within the function, you can use "const" like so:

```
int add (const int a, const int b) {
    return a+b;
}
```

This throws an error if program tries to change a and b.

Function categories

- No return, no args taken
- No return, args taken
- Returns, no args taken
- Returns, args taken

Passing by value / parameter passing

Used to change the value of a variable outside the local scope of a function

Passing by value / parameter passing

Used to change the value of a variable outside the local scope of a function

```c
#include<stdio.h>
void increment(int *);

int main(){
    int a=2;
    printf("main: before call to fn increment a=%d\n",a);
    increment(&a);
    printf("In main: after call to fn increment a=%d\n",a);
}
 void increment(int *p){
    printf("In increment: before changing the value a=%d\n",*p);
    (*p)++;
    printf("In increment: after changing the value a=%d\n",*p);
 }
```

(*p)++; → brackets are important here

NOTE: Passing pointers

void func (int *p)

[OR]

void func (int*)

Special cases w/ pointers

int *p; → wild pointer: dereferencing gives junk value

int *p = NULL; → null pointer: dereferencing undefined

```
int f6 (){
    a=20;
    return &a;}
int *p = f6()
```

→ dangling pointer: dereferencing undefined (local variable)

ARRAYS

Collection of similar elements of same datatype; linear data structure

- Declared with square brackets

  int arr[66];

  n ∈ [0,65]

- Cannot declare an empty array; you can initialise the array without giving size though

  int arr[] ✗ int arr[] = {10,20,30} ✓

- Values in an array are stored in continuous memory locations

Array operations

```c
int a[5] = {10,20,30,40,50};
printf("%lu", sizeof(a));
int n = sizeof(a) / sizeof(a[0])
```

NOTE:
Arrays do not grow or shrink in size.

```
printf("%lu", sizeof(a));
int n = sizeof(a)/sizeof(a[0])
for(int i=0; i<n; i++) {
    printf("%d", a[i]);
}
float b[5] = {10,20,30};
int c[5];
```

`c=a;` ✗ arrays are not assignment compatible

`c[0]=a[0];` ✓ you can copy element by element;
using a loop you can copy the entire array

NOTE: Arrays do not grow or shrink in size.

Designated initialisation

```
int a[5] = {[4]=40, [2]=25};
```

0	0	25	0	0	40
0	1	2	3	4	5

```
int b[] = {[3]=16, [0]=1};
```

1	0	0	16
0	1	2	3

```
int c[] = {[3]=20, 10, [0]=1};
```

1	10	0	20
0	1	2	3

Variable length array

int variable used as length argument

```
int n=5;
int arr[n];   ]→ array of length 5
// n = 10;    ]→ does not change length of array, messes up logic
```

NOTE: Variable length arrays do not support initialisation

```
for(int i=0; i<n; i++) {
    scanf("%d", &a[i]);
}
```

internal working

$a[i] \Rightarrow *(a+i)$

Say $\&a = 4000$

$a[100] \Rightarrow *(a+i)$

$= *(4000 + (4 \times 100))$

$= *(4400)$ → address of a[100] Elements

$*(a+i) \rightleftharpoons *(i+a)$

$\Rightarrow a[i] \rightleftharpoons i[a]$

$(a+i) \rightleftharpoons (i+a)$

$\Rightarrow \&a[i] \rightleftharpoons \&i[a]$ } Addresses

Using some int *p = a,
replace a with p in the above statements to do the same thing

During runtime, arrays degenerate into constant pointers (pointers that point to one address & cannot be changed). This implies:

Assignment X Incrementation X

HOWEVER:

```
int a[5] = {1,2,3,4,5};
int *p = a;
p++;
```

✓ VALID

"p" is not a constant pointer unlike "a". After incrementation, p points to element at index location 1.