Xi'an Jiaotong-Liverpool University
西交利物浦大学

# CPT104 Operating System Concepts

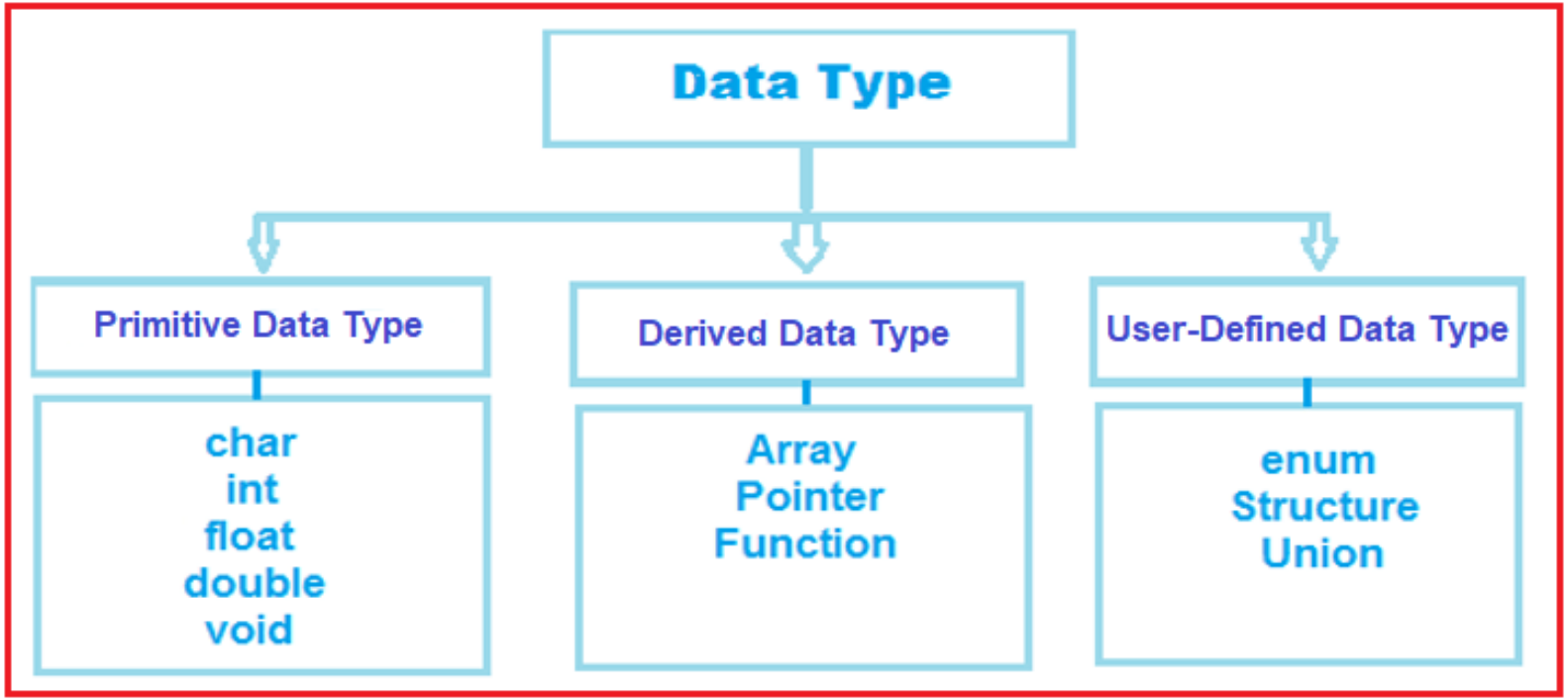
## Lab 6

# Review of C language

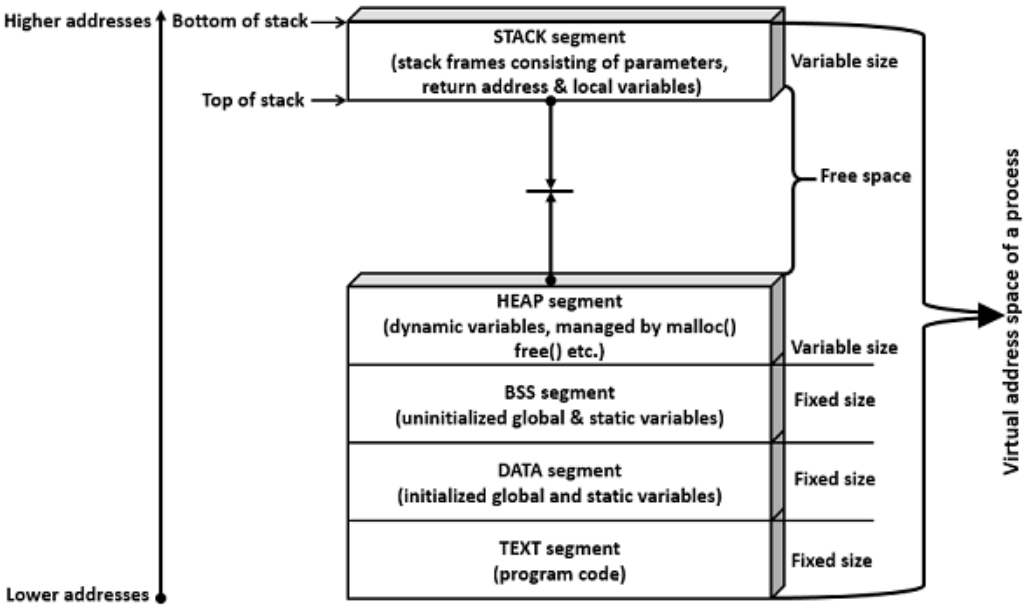
# Contents

- C language review
  - Data types and variables
  - Control flow
  - Pointer
  - Functions and program structure
- Exercises
- Recommended coding style

# Data types and variables



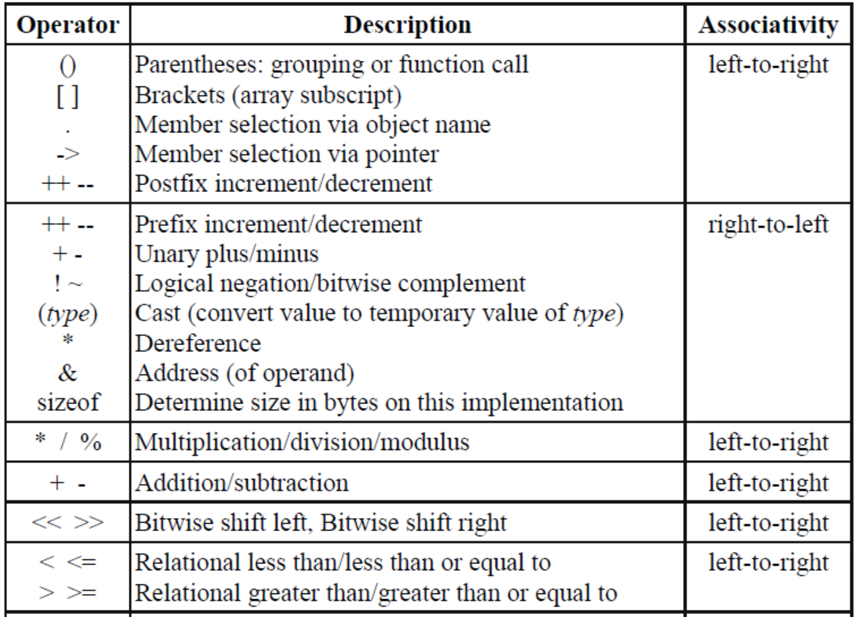
# Data types and variables




## Memory layout of local, static, and global variables

# Data types and variables

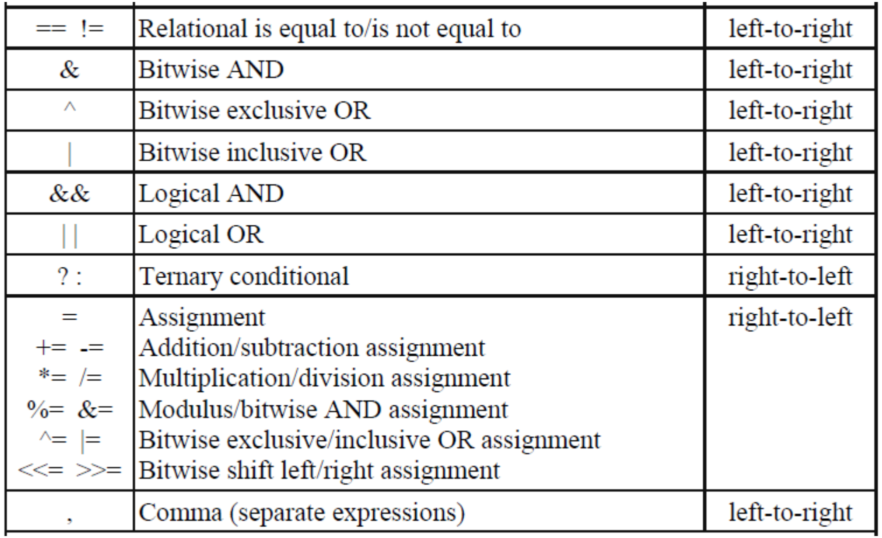
1



| Operator | Description | Associativity |
|---|---|---|
| () | Parentheses: grouping or function call | left-to-right |
| [ ] | Brackets (array subscript) | |
| . | Member selection via object name | |
| -> | Member selection via pointer | |
| ++ -- | Postfix increment/decrement | |
| ++ -- | Prefix increment/decrement | right-to-left |
| + - | Unary plus/minus | |
| ! ~ | Logical negation/bitwise complement | |
| (*type*) | Cast (convert value to temporary value of *type*) | |
| * | Dereference | |
| & | Address (of operand) | |
| sizeof | Determine size in bytes on this implementation | |
| * / % | Multiplication/division/modulus | left-to-right |
| + - | Addition/subtraction | left-to-right |
| << >> | Bitwise shift left, Bitwise shift right | left-to-right |
| < <= | Relational less than/less than or equal to | left-to-right |
| > >= | Relational greater than/greater than or equal to | |

6

7



| | | |
|---|---|---|
| == != | Relational is equal to/is not equal to | left-to-right |
| & | Bitwise AND | left-to-right |
| ^ | Bitwise exclusive OR | left-to-right |
| \| | Bitwise inclusive OR | left-to-right |
| && | Logical AND | left-to-right |
| \|\| | Logical OR | left-to-right |
| ? : | Ternary conditional | right-to-left |
| = | Assignment | right-to-left |
| += -= | Addition/subtraction assignment | |
| *= /= | Multiplication/division assignment | |
| %= &= | Modulus/bitwise AND assignment | |
| ^= \|= | Bitwise exclusive/inclusive OR assignment | |
| <<= >>= | Bitwise shift left/right assignment | |
| , | Comma (separate expressions) | left-to-right |

15

## Operator priority (Precedence)

# Data types and variables

**Constant :**

- using const keyword to define a variable const
  double PI = 3.14;
  PI = 2.9; //Error
- using #define preprocessor
- #define PI 3.14

**Undefined array size** to provide flexible initialization.

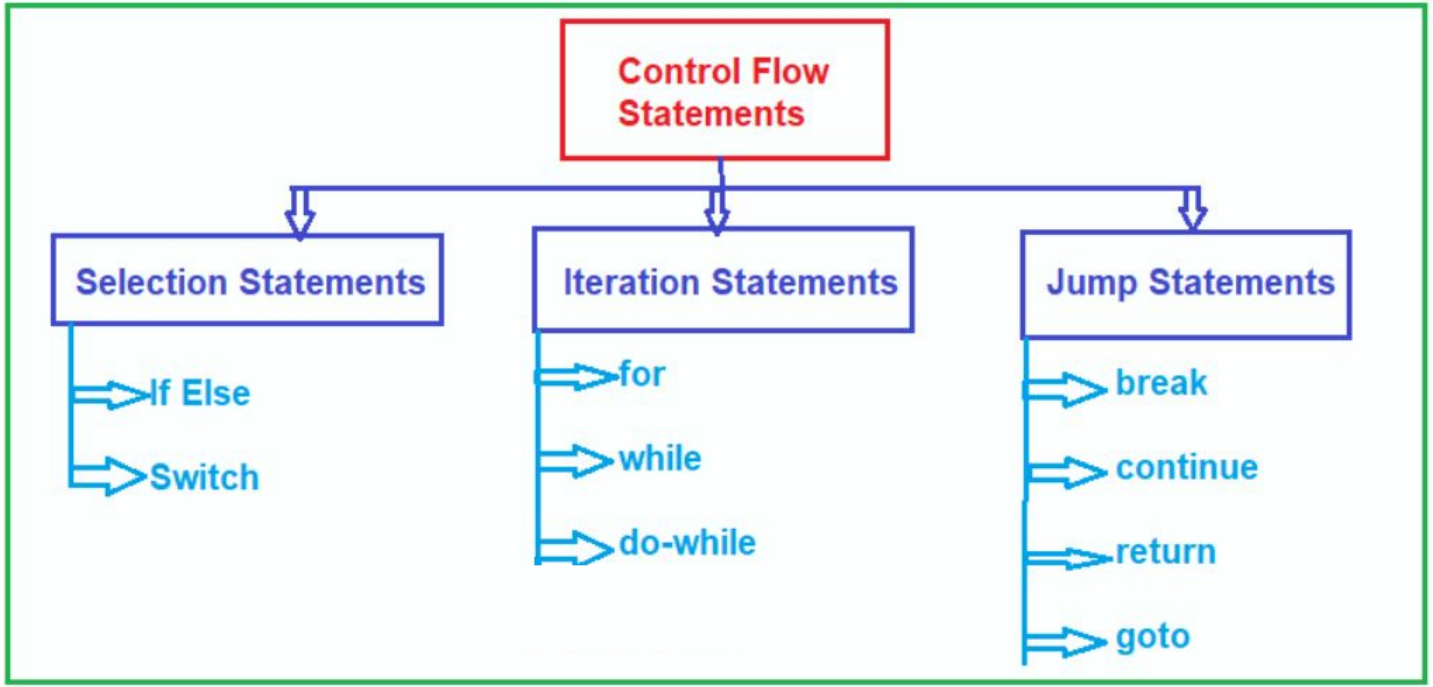
- char mark[] = "Hello, world!";

**typedef :**

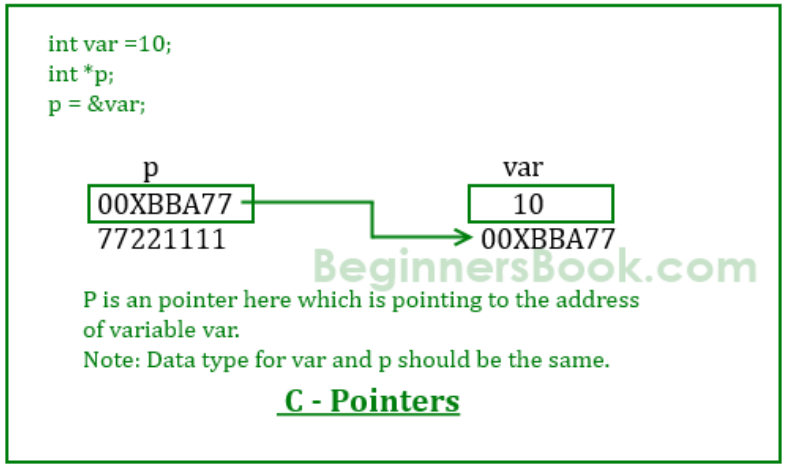
```
typedef struct person {
    char name[50];
    int age;
} Person;
Person p1;
```

```
Person create_person(char name[], int age) {
    Person p;
    strcpy(p.name, name);
    p.age = age;
    return p;
}
```

# Control flow



# Pointer




Pointer concept

## Pointer initialization

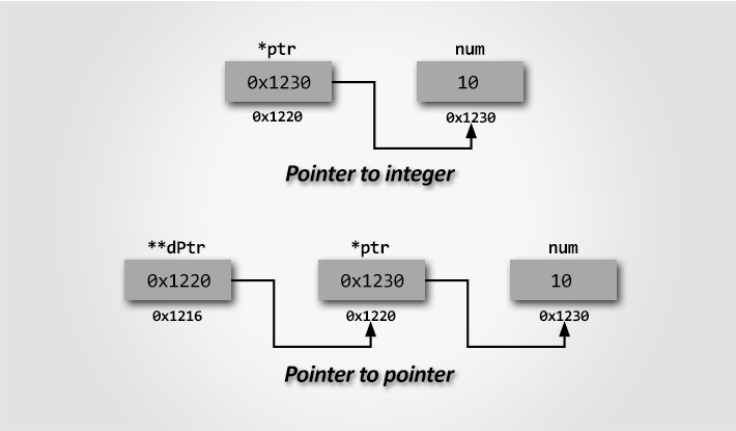
- Initializing a pointer to NULL:
  int *ptr = NULL;
- Initializing a pointer to a variable:
  int num = 10;
  int *ptr = #
- Initializing a pointer to a string:
  char *str = "Hello, world!";
- Initializing a pointer to an array:
  int arr[] = {1, 2, 3, 4, 5};
  int *ptr = arr;
- Initializing a pointer to a structure:
  struct struct_name *pointer_name;

# Pointer

## Pointer and array

```
int arr[5] = {1, 2, 3, 4, 5};
int *ptr = arr;//or int *ptr = &arr[0];
printf("%d\n", *(ptr+2));
// or can be presented like:
printf("%d\n", ptr[2]);
```

## Pointer to pointer




Complete the function **void printStrLen(char ** input, int num)** that given **an array of strings** (which are pointers to chars) and an integers num which indicates how many strings in the array, print the length of each string in a separate line.

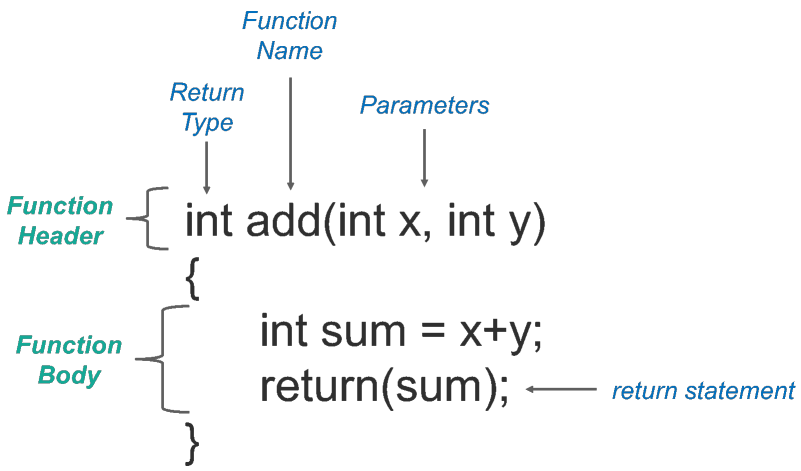
**For example:**

| Test | Result |
|---|---|
| `char * input[3] = {"x", "55", "ab34"};`<br>`printStrLen(input, 3);` | 1<br>2<br>4 |

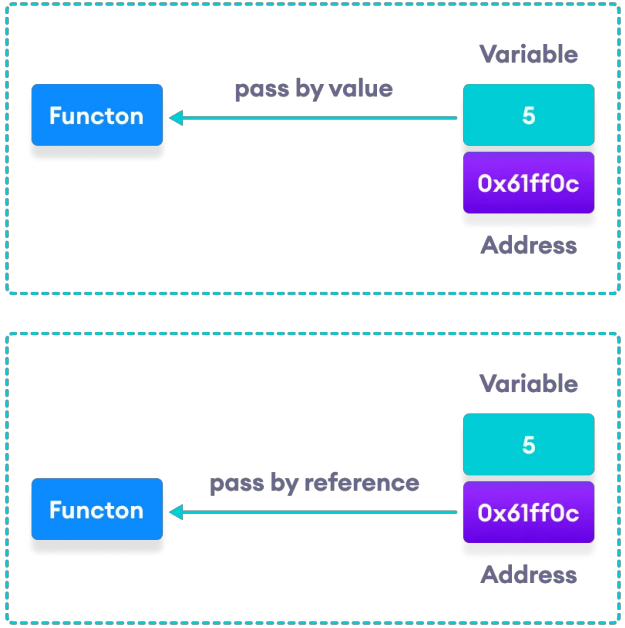
```
void printStrLen(char** input, int num) {

        for(int i=0;i<num;i++){
         int str_length=0;
         while(*(*(input+i)+str_length)!='\0')
             str_length++;
         printf("%d\n",str_length);
     }

}
```

# Functions

## Function definition




## Pass by value & by reference




## Function declaration

```
int function_name(int var1, int var2);
```

# Functions

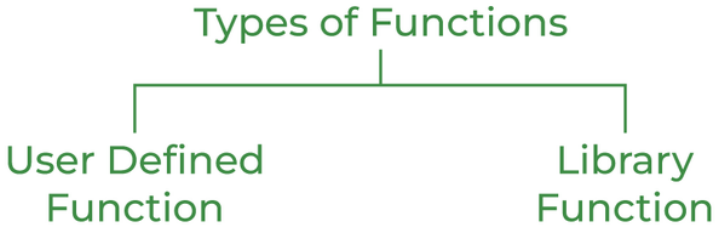
## Recursive function

```
void recurse() {
    ... .. ...
    recurse();
    ... .. ...
}

int main() {
    ... .. ...
    recurse();
    ... .. ...
}
```

recursive call

function call

- Base Case
- Recursive step




- The header <stdlib.h> declares functions for number conversion, storage allocation, and similar tasks.
- The header <math.h> declares mathematical functions and macros.
- The input and output functions, types, and macros defined in <stdio.h>

Refer to reference book: **Appendix B - Standard Library**

# General structure of a c programming

For a complex program, user's defined header file could be declared here

1. #include <>
2. #include ""
3. #defines
4. Data Types (e.g., structures)
5. Globals
6. Prototypes
7. Code

```
// name of program ---->document section
#include<stdio.h>
#include<conio.h>      --->link sectioon
#define MIN 99     ---->define section
void fun();      ----->function declaration section
int a=100;     ------->global variable section
void main()   ---->main section
{
int a=200;      ----->local variable
printf(" hello world");
getch();                --->body of main function
}
void fun()
{                       ->function defination
printf(" hello fun");
}
```

# Exercise

**What will be the output of following program?**

```c
#include<stdio.h>
void f(int *p, int *q)
{
  p = q;
  *p = 2;
}
int i = 0, j = 1;
int main()
{
  f(&i, &j);
  printf("%d %d", i, j);
  getchar();
  return 0;
}
```

a) 2 2

b) 2 1

c) 0 1

d) 0 2

# Exercise

**What will be the output of following program?**

```
#include<stdio.h>
void mystery(int *ptra, int *ptrb)
{
   int *temp;
   temp = ptrb;
   ptrb = ptra;
   ptra = temp;
}
int main()
{
    int a=2016, b=0, c=4, d=42;
    mystery(&a, &b);
    if (a < c)
       mystery(&c, &a);
    mystery(&a, &d);
    printf("%d", a);
}
```

a) 2016

b) 0

c) 4

d) 42

# Exercise

**What will be the output of following program?**

```
#include <iostream>
#include <string.h>
using namespace std;
int main()
{
    int count;
    for (count = 0; count < 10; ++count) {
        printf("#");
        if (count > 6)
            continue;
        printf("%d", count);
    }
    return 0;
}
```

a) #0#1#2#3#4#5#6###

b) #0#1#2#3#4#5#6#7#8#9#10

c) #0#1#2#3#4#5#

d) #0#1#2#3#4#5##7#8#9#10

# Exercise

**What will be the output of following program?**

```
int f(int x)
{
    if(x <= 4)
        return x;
    return f(--x);
}
void main()
{
    printf("%d ", f(7));
}
```

a) 4 5 6 7

b) 1 2 3 4

c) 4

d) syntax error

# Exercise

**What will be the output of following program?**

```
int main()
{
    struct forest
    {
        int trees;
        int animals;
    }F1,*F2;
    F1.trees=1000;
    F1.animals=20;
    F2=&F1;
    printf("%d ",F2.animals);
    return 0;
}
```

a) 0

b) 20

c) Compiler error

d) None of the above

# Exercise

**When an array is passed as parameter to a function, which of the following statements is correct?**

a) The function can change values in the original array.

b) In C, parameters are passed by value, the function cannot change the original value in the array.

c) It results in compilation error when the function tries to access the elements in the array.

d) Results in a run time error when the function tries to access the elements in the array.

# Recommended coding style of C language

## Layout

- Use 4 spaces per indent level
- Use 1 space between keyword and opening bracket
- Do not use space between function name and opening bracket
- Opening curly bracket is always at the same line as keyword (for, while, do, switch, if, ...)
- Use single space after every comma, before and after comparison and assignment operators

```c
struct struct_name {
    char* a;
    char b;
};
```

```c
/* OK */
if (condition)
while (condition)
for (init; condition; step)
do {} while (condition)
```

```c
int32_t a = sum(4, 3);          /* OK */
int32_t a = sum (4, 3);         /* Wrong */
```

```c
for (i = 0; i < 5; ++i) {
}
```

```c
/* OK */
if (c) {
    do_a();
} else {
    do_b();
}
```

```c
a = 3 + 4;              /* OK */
for (a = 0; a < 5; ++a) /* OK */
```

```c
func_name(5, 4);        /* OK */
func_name(4,3);         /* Wrong */
```

# Recommended coding style of C language

## Naming conventions:

- Use descriptive names for variables, functions, and other identifiers.
- Use underscores for multi-word variables and functions (do not use camelcase in some recommendations), and use all caps with underscores for constants.
- Declare pointer variables with asterisk aligned to type; When declaring multiple pointer variables, you may declare them with asterisk aligned to variable name

```
double  hot_water_temperature;
```

```
int count_some_item(...) {
    ...
}
```

```
/* OK */
char* a;

/* Wrong */
char *a;
char * a;
```

```
/* OK */
char *p, *n;
```

# Recommended coding style of C language

## Simple Statements

- make your program more **concise and readable**.

refer to reference book for more examples

```
while ((c = getchar()) != EOF) {
        process the character
}
```

```
int c, i, j;

for (i = 0, j = strlen(s)-1; i < j; i++, j--) {
    c = s[i];
    s[i] = s[j];
    s[j] = c;
}
```

```
while (--lim > 0 && (c=getchar()) != EOF && c != '\n')
    s[i++] = c;
```

```
// Macro to check and print even odd number
#define EVEN_ODD(num)                \
    if (num & 1)                     \
        printf("%d is odd\n", num); \
    else                             \
        printf("%d is even\n", num);
```

```
/* strcpy:  copy t to s; pointer version 3 */
void strcpy(char *s, char *t)
{
    while (*s++ = *t++)
        ;
}
```

# Recommended coding style of C language

## Comments

```
/*
 * Here is a block comment.
 * The comment text should be tabbed or spaced over uniformly.
 * The opening slash-star and closing star-slash are alone on a line.
 */

 if (argc > 1) {
        /* Get input file from command line. */
        if (freopen(argv[1], "r", stdin) == NULL) {
                perror(argv[1]);
        }
 }

 if (a == EXCEPTION) {
         b = TRUE;       /* special case */
 } else {
         b = isprime(a); /* works only for odd a */
 }
```

- It should be pointed out that there is no one "**official**" coding style for C language, but some general recommended guidelines could be followed.

Reference:

https://github.com/MaJerle/c-code-style

https://www2.cs.arizona.edu/~mccann/cstyle.html