C Programming Important Topics — Full Notes

---

1. Identifier and Its Rules

Short Answer:
Identifiers are names given to variables, functions, arrays, etc. They help identify programming elements.

Detailed Explanation:
Identifiers are user-defined names used for various programming elements like variables and functions.

Rules:

1. Must begin with a letter (A–Z or a–z) or underscore (_).
2. Can contain letters, digits (0–9), and underscores.
3. No spaces or special symbols allowed.
4. Cannot be a keyword.
5. Case-sensitive (`Total` and `total` are different).

Example:

```
int age;
float total_marks;
char studentName[20];
```

Valid: `sum`, `_data1`, `totalMarks`
Invalid: `1value`, `total-marks`, `float` (keyword)

---

2. Keywords / Variable / Character Set

Short Answer:
Keywords are reserved words.
Variables store data.
Character set defines valid characters.

Detailed Explanation:

➤ **Keywords:**

Predefined words with fixed meanings.
Example:
`int`, `float`, `if`, `else`, `return`, `for`, `while`, `break`, `continue`, etc.

➤ **Variables:**

Used to store data of specific type.

Syntax:

```
data_type variable_name = value;
```

Example:

```
int age = 20;
float price = 99.50;
char grade = 'A';
```

➤ **Character Set:**

Characters allowed in C program:

- Letters (A–Z, a–z)
- Digits (0–9)
- Special symbols (+, -, *, /, %, etc.)
- White spaces
- Escape sequences (`\n`, `\t`)

---

3. Actual and Formal Parameters

Short Answer:
Actual parameters → values passed in function call.
Formal parameters → variables declared in function definition.

Detailed Explanation:

```
#include <stdio.h>
void add(int a, int b) // formal parameters
{
    printf("Sum = %d", a + b);
}

int main()
{
```

```
    add(5, 10); // actual parameters
    return 0;
}
```

- **Actual Parameters:** `5, 10`
- **Formal Parameters:** `int a, int b`

---

4. Loops / Infinite Loops (for, while, do-while)

Short Answer:
Loops repeat a block of code multiple times until a condition becomes false.

Detailed Explanation:

➤ For Loop

```
for(initialization; condition; update)
{
    // code
}
```

Example:

```
for(int i=1; i<=5; i++)
    printf("%d ", i);
```

➤ While Loop

```
int i=1;
while(i<=5)
{
    printf("%d ", i);
    i++;
}
```

➤ Do-While Loop

Executes once before checking condition.

```
int i=1;
do {
    printf("%d ", i);
    i++;
} while(i<=5);
```

➤ **Infinite Loop**

If condition never becomes false.

```
while(1) {  // or for(;;)
    printf("Running...\n");
}
```

---

5. Operators / Precedence / Associativity

Short Answer:
Operators perform operations. Precedence decides which executes first. Associativity decides direction.

Detailed Explanation:

➤ **Types of Operators:**

- Arithmetic: `+ - * / %`
- Relational: `== != > < >= <=`
- Logical: `&& || !`
- Assignment: `= += -=`
- Increment/Decrement: `++ --`
- Conditional: `?:`
- Bitwise: `& | ^ << >> ~`

➤ **Example:**

```
int a=10, b=5, c=2;
int result = a - b * c; // b*c first due to precedence
printf("%d", result); // Output: 0
```

➤ **Precedence Example:**

Operator	Description	Associativity
`()`	Brackets	Left to Right
`* / %`	Multiplication/Division	Left to Right
`+ -`	Addition/Subtraction	Left to Right

=	Assignment	Right to Left

---

6. Bitwise / Logical / sizeof() Operators

Short Answer:
Used for bit-level, logic, and memory size operations.

Detailed Explanation:

➤ **Bitwise Operators:**

Operate on bits.

Operator	Meaning
&	AND
`	`
^	XOR
~	NOT
<<	Left Shift
>>	Right Shift

Example:

```
int a = 5, b = 3;
printf("%d", a & b); // 1 (0101 & 0011)
```

➤ **Logical Operators:**

Operator	Meaning
&&	AND
`	
!	NOT

Example:

```
if(a>0 && b>0) printf("Both positive");
```

➤ **sizeof() Operator:**

Gives size of data type or variable.

```
printf("%lu", sizeof(int)); // typically 4
```

---

7. Break / Continue / goto Statements

Short Answer:
Used for controlling loop flow.

Detailed Explanation:

➤ **break:**

Stops loop immediately.

```
for(int i=1;i<=5;i++){
    if(i==3) break;
    printf("%d ",i);
}
// Output: 1 2
```

➤ **continue:**

Skips current iteration.

```
for(int i=1;i<=5;i++){
    if(i==3) continue;
    printf("%d ",i);
}
// Output: 1 2 4 5
```

➤ **goto:**

Jumps to labeled statement.

```
int i=1;
start:
printf("%d ", i);
i++;
if(i<=5) goto start;
```

---

8. Switch Statement / Programs

Short Answer:
Used to execute one case from multiple options.

Detailed Explanation:

```
int choice = 2;
switch(choice)
{
    case 1: printf("Hello"); break;
    case 2: printf("Hi"); break;
    default: printf("Invalid");
}
```

Rules:

- Each case must end with `break;`.
- `default` is optional.
- Only integers/characters allowed in `case`.

---

9. Call by Value / Call by Reference

Short Answer:

- **Call by Value:** Copies the value → changes not reflected.
- **Call by Reference:** Passes address → changes reflected.

Detailed Explanation:

➤ Call by Value

```
void change(int x){
    x = 10;
}
int main(){
    int a = 5;
    change(a);
    printf("%d", a); // Output: 5
}
```

➤ Call by Reference

```
void change(int *x){
    *x = 10;
}
int main(){
    int a = 5;
    change(&a);
    printf("%d", a); // Output: 10
}
```

---

10. Functions

Short Answer:
Functions are blocks of code that perform specific tasks and can be reused.

Detailed Explanation:

➤ **Syntax:**

```
return_type function_name(parameters)
{
    // body
}
```

➤ **Example:**

```
#include <stdio.h>

int add(int a, int b) // function definition
{
    return a + b;
}

int main()
{
    int sum = add(5, 10); // function call
    printf("Sum = %d", sum);
    return 0;
}
```

➤ **Types:**

1. Library Functions (e.g., `printf()`, `scanf()`)
2. User-defined Functions

➤ **Advantages:**

- Code reusability
- Readability
- Easy debugging

🪶 Summary Table

No	Topic	Key Concept	Example Keyword
1	Identifier	Naming rules	`totalMarks`
2	Keywords/Variable	Data storage	`int age=20;`
3	Parameters	Function inputs	`add(5,10)`
4	Loops	Repetition	`for, while`
5	Operators	Calculations	`+ - * / %`
6	Bitwise/Logical	Bit & logic ops	`&, `
7	Break/Continue	Flow control	`break;`
8	Switch	Multi-choice	`switch(x)`
9	Call by Value/Reference	Function passing	`change(&a);`
10	Function	Code block	`int sum(int a,int b)`