KEY WORDS IN ***C***

1. AUTO:

* auto is the default storage class variable that is declared inside a function or a block.
* auto variables can only be accessed within the function/block they are declared.
* By default, auto variables have garbage values assigned to them.
* Automatic variables are also called local variables as they are local to a function.
* It is used to store the value for variable

1. BREAK:

* The break statement is used to terminate the innermost loop. It generally terminates a loop or a break statement.
* The continue statement skips to the next iteration of the loop.
* "break" is a fundamental keyword in C for altering the flow of control within loops and switch statements, providing a way to exit prematurely when certain conditions are met.
* Top of Form
* Bottom of Form
* a switch statement, break is used to terminate each case block.
* If break is not used, execution will "fall through" to the next case (unless it's the end of the switch or another break is encountered).

1. CASE:

* The switch statement in C is used as an alternate to the if-else ladder statement.
* For a single variable i.e, switch variable it allows us to execute multiple operations for different possible values of a single variable.
* **Notes:**
* Each case label must be a constant integer expression or character literal.
* The switch statement evaluates the expression once and then compares its value with each case label sequentially.
* Use break to exit the switch statement after executing the corresponding case block. If break is omitted, control will "fall through" to the next case label.
* The default case is optional but recommended to handle unexpected values of the expression.

1. CHAR:

* char keyword in C is used to declare a character variable in the C programming language.
* It is a data type used to store single characters (like 'a', 'b', '1', '$', etc.) in C programming.
* In memory, each char typically occupies 1 byte (8 bits).
* Character literals are enclosed in single quotes ('). For example, 'A', 'b', '1', '%', etc.

1. CONST:

* The const keyword defines a variable who’s value cannot be changed.
* Constants make code more readable by giving meaningful names to values that do not change.
* It prevents accidental modification of values that are intended to remain constant, reducing the risk of logical errors.

1. CONTINUE:

* "continue" is a keyword used within loop constructs (for, while, do-while) to skip the current iteration of the loop and proceed directly to the next iteration.
* It is used to selectively skip certain iterations based on a condition, without terminating the loop entirely.
* Use continue when you want to skip executing the remaining code inside the loop for a specific condition but continue with the next iteration of the loop.
* continue helps in avoiding deeply nested conditional statements by allowing you to skip iterations based on simple conditions.

1. DEFAULT:

* The default keyword specifies the code to execute if none of the case constants in the switch statement match the value of the expression being evaluated.
* The default case in a switch statement provides a fallback option for handling unexpected or unspecified values of the evaluated expression.
* It is not necessary to include a default case in every switch statement, but it is good practice to provide one to handle unexpected inputs or as a form of error handling.
* The default case is often used to display error messages, handle default behaviors, or manage edge cases in program logic.

1. DO:

* **Purpose**: Executes a block of code repeatedly based on a condition.
* **Usage**: Useful when you want to run a block of code at least once before checking the condition.
* **Example**: do { ... } while (condition);
* **Note**: Ensure the condition eventually becomes false to exit the loop.

1. DOUBLE:

* Purpose: Stores decimal numbers with double precision.
* Usage: Provides higher accuracy and larger range compared to float.
* Example: double num = 3.14159;
* Note: Uses more memory than float but is more precise.

1. else:

* Purpose: Provides an alternative action if an if condition is false.
* Usage: Helps in decision-making between two or more actions.

1. enum:

* **Purpose**: Defines a set of named integral constants.
* **Usage**: Improves code readability by using meaningful names instead of numbers.
* **Note**: Enums start numbering from 0 unless specified.

1. **extern**:

* **Purpose**: Declares a variable or function that is defined in another file.
* **Usage**: Allows sharing variables and functions between different source files.
* **Example**: extern int count;
* **Note**: Ensures the variable or function is defined elsewhere in the program.

1. **float**:

* **Purpose**: Stores decimal numbers with single precision.
* **Usage**: Uses less memory compared to double but has lower precision.
* **Example**: float num = 3.14;
* **Note**: Suitable for applications where precision is not critical.

1. **for**:

* **Purpose**: Creates a loop that executes a block of code a fixed number of times.
* **Usage**: Combines initialization, condition, and increment/decrement in one line.
* **Example**:

c

Copy code

for (int i = 0; i < 10; i++) {

// code to execute

}

* **Note**: for loops are useful when the number of iterations is known.

1. **goto**:

* **Purpose**: Transfers control to a labeled statement within the same function or file.
* **Usage**: Provides a way to jump to a specific part of the code.
* **Example**:

c

Copy code

goto label;

// ...

label:

// code to execute

* **Note**: Often discouraged due to making code harder to understand and maintain.

1. **if**:

* **Purpose**: Executes a block of code based on a specified condition.
* **Usage**: Essential for decision-making in programming.
* **Example**:

c

Copy code

if (condition) {

// code to execute if condition is true

}

* **Note**: Can be followed by else for alternative execution.

1. **int**:

* **Purpose**: Stores integer numbers (whole numbers).
* **Usage**: Basic data type for counting and indexing.
* **Example**: int num = 42;
* **Note**: Occupies a fixed amount of memory, typically 4 bytes.

1. **long**:

* **Purpose**: Stores larger integer numbers than int.
* **Usage**: Useful when int is not large enough to hold the required range of values.
* **Example**: long num = 1234567890;
* **Note**: Occupies more memory than int, usually 4 or 8 bytes.

1. **register**:

* **Purpose**: Suggests to the compiler to store a variable in a CPU register for faster access.
* **Usage**: Optimizes performance of critical variables.
* **Example**: register int counter;
* **Note**: Modern compilers may ignore this suggestion as they optimize automatically.

1. **return**:

* **Purpose**: Exits from a function and optionally returns a value to the caller.
* **Usage**: Essential for functions that produce results or perform actions.
* **Example**: return 0;
* **Note**: Functions declared as void do not return a value (return;).

1. **short**:

* **Purpose**: Stores smaller integer numbers than int.
* **Usage**: Saves memory when space is critical.
* **Example**: short num = 100;
* **Note**: Typically occupies 2 bytes of memory.

1. **signed**:

* **Purpose**: Specifies that a data type can represent both positive and negative values.
* **Usage**: Default for most integer types (int, short, long).
* **Example**: signed int temperature = -10;
* **Note**: Can explicitly use signed to emphasize signedness, though it's usually implicit.

1. **sizeof**:

* **Purpose**: Returns the size of a variable or data type in bytes.
* **Usage**: Essential for memory allocation and operations requiring knowledge of data size.
* **Example**: int size = sizeof(int);
* **Note**: Size may vary depending on the system and compiler.

1. **static**:

* **Purpose**: Specifies that a variable or function has internal linkage.
* **Usage**: Retains the variable's value between function calls.
* **Example**: static int count = 0;
* **Note**: Not visible or accessible outside the file where it's declared.

1. **struct**:

* **Purpose**: Defines a composite data type that can store multiple variables of different types.
* **Usage**: Organizes related data into a single unit for easier management.
* **Example**:

c

Copy code

struct Point {

int x;

int y;

};

* **Note**: Accessed using dot (.) notation (structName.memberName).

1. **switch**:N

* **Purpose**: Provides multi-way branching based on the value of an expression.
* **Usage**: Alternative to multiple if-else statements for clarity and efficiency.
* **Example**:

c

Copy code

switch (grade) {

case 'A':

// code for 'A' grade

break;

// other cases

}

* **Note**: Each case must end with break to prevent fall-through.

1. **typedef**:

* **Purpose**: Creates an alias or new name for an existing data type.
* **Usage**: Simplifies complex type declarations and improves code readability.
* **Example**:

c

Copy code

typedef unsigned long long int ULL;

ULL num = 1234567890ULL;

* **Note**: Can be used with struct, enum, or any data type.

1. **union**:

* **Purpose**: Allows storing different data types in the same memory location.
* **Usage**: Saves memory when only one member of the union is needed at a time.
* **Example**:

c

Copy code

union Data {

int i;

float f;

char str[20];

};

* **Note**: Accesses the active member using the same memory space.

1. **unsigned**:

* **Purpose**: Specifies that a data type can represent only non-negative values (zero or positive).
* **Usage**: Expands the range of positive values compared to signed counterparts.
* **Example**: unsigned int count = 10;
* **Note**: unsigned is useful when dealing with quantities or flags.

1. **void**:

* **Purpose**: Indicates that a function does not return a value or that a pointer has no specific type.
* **Usage**: Essential for defining functions that perform actions without returning results (void functions) or for generic pointers (void\*).
* **Example**:

c

Copy code

void printMessage() {

printf("Hello, World!\n");

}

* **Note**: void\* can hold the address of any data type.

1. **volatile**:

* **Purpose**: Indicates that a variable's value can be changed unexpectedly.
* **Usage**: Ensures that reads and writes to the variable are not optimized out by the compiler.
* **Example**: volatile int flag = 0;
* **Note**: Useful for variables modified by hardware or different threads.

1. **while**:

* **Purpose**: Creates a loop that executes a block of code as long as a specified condition is true.
* **Usage**: Suitable when the number of iterations is not known beforehand.