**KEYWORDS IN C PROGRAMMING LANGUAGE**

**1. auto:** Declares a variable with automatic storage duration, meaning it is created when entering a block and destroyed when exiting.

auto int x = 10;

**2. break:** Terminates the nearest enclosing loop or switch statement and transfers control to the statement immediately following it.

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

if(i == 5) {

break;

}

printf("%d ", i);

}

**3. case:** A label within a switch statement that identifies a specific value to match against the switch expression.

int x = 2;

switch(x) {

case 1:

printf("One");

break;

case 2:

printf("Two");

break;

default:

printf("Default");

}

**4. char:** Defines a data type capable of holding a single character or small integer value.

char ch = 'A';

printf("%c", ch);

**5. const:** Declares a variable as unmodifiable, ensuring its value cannot be changed after initialization.

const int x = 10;

x = 20; // Error: assignment of read-only variable

**6. continue:** Ends the current iteration of a loop prematurely and starts the next iteration immediately.

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

if(i % 2 == 0) {

continue;

}

printf("%d ", i);

}

**7. default:** Specifies the default case in a switch statement, executed when none of the other cases match the switch expression.

int x = 3;

switch(x) {

case 1:

printf("One");

break;

case 2:

printf("Two");

break;

default:

printf("Default");

}

**8. do:** Initiates a loop that repeatedly executes a block of statements until a specified condition becomes false.

int i = 0;

do {

printf("%d ", i);

i++;

} while(i < 5);

**9. double:** Defines a data type capable of holding double-precision floating-point numbers, typically with higher precision than float.

double pi = 3.14159;

printf("%lf", pi);

**10. else:** Executes a block of code if the preceding if statement's condition evaluates to false.

int x = 10;

if(x < 5) {

printf("x is less than 5");

} else {

printf("x is 5 or more");

}

**11. enum:** Introduces a user-defined data type called an enumeration, consisting of a set of named integer constants.

enum week {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};

enum week today = Wednesday;

**12. extern:** Declares a variable or function that is defined elsewhere in the program or externally, allowing its use across multiple files.

extern int x; // Declaration

int x = 10; // Definition

**13. float:** Declares a data type capable of holding single-precision floating-point numbers.

float f = 3.14f;

printf("%f", f);

**14. for:** Initiates a loop that executes a block of statements repeatedly based on specified initialization, condition, and iteration expression.

float f = 3.14f;

printf("%f", f);

**15. goto:** Transfers control to a labeled statement elsewhere in the program, allowing unconditional jumps.

int i = 0;

while (i < 10) {

if (i == 5) {

goto label;

}

printf("%d ", i);

i++;

}

label:

printf("Jumped to label");

**16. if:** Executes a block of statements if a specified condition evaluates to true.

int x = 10;

if(x > 5) {

printf("x is greater than 5");

}

**17. int:** Defines a data type capable of holding whole numbers, both positive and negative.

int x = 10;

printf("%d", x);

**18. long:** Declares a data type capable of holding larger integers than int, typically with a wider range.

long int x = 100000L;

printf("%ld", x);

**19. register:** Suggests to the compiler that a variable should be stored in a CPU register for faster access.

register int x = 10;

**20. return:** Exits from a function and optionally returns a value to the calling function.

int sum(int a, int b) {

return a + b;

}

**21. short:** Defines a data type capable of holding smaller integers than int, typically with a narrower range.

short int x = 5;

printf("%d", x);

**22. signed:** Specifies that a data type can hold both positive and negative numbers.

signed int x = -10;

printf("%d", x);

**23. sizeof:** Returns the size of a variable or data type in bytes.

int x = 10;

printf("%zu", sizeof(x));

**24. static:** Specifies that a variable retains its value between function calls, allowing it to preserve its value across multiple invocations.

static int x = 10;

x++;

printf("%d", x);

**25. struct:** Defines a user-defined data type that can hold multiple variables of different data types under a single name.

struct Point {

int x;

int y;

};

struct Point p = {10, 20};

**26. switch:** Starts a control structure that evaluates an expression against multiple case labels and executes code blocks based on the matching case.

int x = 2;

switch(x) {

case 1:

printf("One");

break;

case 2:

printf("Two");

break;

default:

printf("Default");

}

**27. typedef:** Creates an alias or synonym for an existing data type, allowing for easier and more intuitive use of complex data structures.

typedef unsigned long ul;

ul x = 1000;

**28. union:** Defines a user-defined data type that allows storing different data types in the same memory location, enabling memory-efficient usage.

union Data {

int i;

float f;

char str[20];

};

union Data data;

data.i = 10;

**29. unsigned:** Specifies that a data type can hold only non-negative numbers, extending its range for positive values but sacrificing representation of negative values.

unsigned int x = 10;

printf("%u", x);

**30. void:** Specifies that a function does not return a value, or indicates a pointer that has no specific data type.

void sayHello() {

printf("Hello");

}

**31. volatile:** Informs the compiler that the value of a variable may change at any time, even if it appears not to be modified, ensuring the compiler does not optimize away potential changes.

volatile int x = 10;

**32. While:** Initiates a loop that repeatedly executes a block of statements while a specified condition remains true.

int i = 0;

while(i < 5) {

printf("%d ", i);

i++;

}