Keywords Identifier Literals Operators and Expression

1. Choose all valid identifiers
a. int int
b. int _numvalue
c. float price_money
d. char name12345678901234567890123456789012345678 90
e. char name value
f. char $name

Sol:

The valid identifiers are:

- b. int _numvalue
- c. float price_money
- d. char name12345678901234567890123456789012345678 90 (valid but long)
- f. char $name

2. What is the meaning of the following keywords, show the usage
a. auto
b. extern
c. volatile
d. sizeof
e. const

Sol:

a. auto:

-> auto is used to specify that the storage class of a variable is automatic, which is the default storage class for local variables. It means the variable is created and destroyed automatically when the function or block in which it is declared is entered or exited.

-> The auto keyword is rarely used explicitly because variables are automatically considered auto by default in local scopes.

b. extern:

-> extern is used to declare a variable or function that is defined in another translation unit (i.e., in another source file). It tells the compiler that the variable or function exists, but its memory location is in some other place (usually in another file).

c. volatile:

-> volatile tells the compiler not to optimize a variable. This is useful in situations where the value of a variable can be changed outside of the program's control (e.g., hardware registers, global variables accessed by multiple threads, or interrupt service routines).

- > The compiler will not cache or optimize access to this variable and will always read its value from memory.

d. sizeof:

- sizeof is an operator (not a function) that returns the size (in bytes) of a variable, data type, or structure.
- It is evaluated at compile time and is used to determine the amount of memory occupied by a data type or variable.

e. const:

-> const is used to declare variables whose values cannot be changed after initialization. It makes the variable constant, so any attempt to modify it will result in a compile-time error.

3. Explain the difference between the following variables.
a. char *ptr = “ABC”;
b. char arr[]=”ABC”;
Can you manipulate the contents of ptr? Why?
Can you manipulate the contents of arr? Why?
Which one of the above is a string literal?

a. char *ptr = "ABC";

- Explanation:
  - This defines a pointer ptr that points to a string literal "ABC".
  - In C, string literals are stored in **read-only** sections of memory (usually in the data segment or a read-only section of memory).
  - The pointer ptr itself is a modifiable pointer, but it points to a memory location that contains the string literal "ABC", which is typically stored in read-only memory.

b. char arr[] = "ABC";

- Explanation:
    - This defines an array of characters arr and initializes it with the string "ABC".
    - A character array is created to hold the string, and in this case, it is a **modifiable** array.
    - The string "ABC" is stored in the array arr, and the array itself holds the characters: 'A', 'B', 'C', and a null terminator '\0' at the end.

Can you manipulate the contents of ptr? Why?

You cannot manipulate the contents of the string through the pointer ptr because string literals are immutable.

Can you manipulate the contents of arr? Why?

You can manipulate the contents of the array arr because arr is a regular character array, and its elements are stored in mutable memory.

4. Predict the output of the following code .

```
void main()
{

//set a and b both equal to 5.
int a=5, b=5;
//Print them and decrementing each time.
//Use postfix mode for a and prefix mode for b.
printf("\n%d %d",a--,--b);
printf("\n%d %d",b++,--b);
}
```

Output:

5 4

4 4

5. Refer the code snippet. It fails with error. Fix it.

```
#include<stdio.h>
int main()
{
int i,k;
const int num;
```

```
/* for(i = 0;i < 9;i++)
{
k = k + 1;
} */
num = num + k; /* Compiler gives the error here */
printf("final value of k:%d\n",k);
printf("value of num:%d\n",num);
return 0;
}
```

sol:

```
#include<stdio.h>
int main() {
   int i, k = 0;
   const int num = 10;
   for(i = 0; i < 9; i++) {
      k = k + 1;
   }
   printf("final value of k: %d\n", k);
   printf("value of num: %d\n", num);
return 0;
}
```

Output:

final value of k: 9

value of num: 10

6. Consider the following code snippet. Evaluate the value of f1, f2 and f3.

```
int main()
{
int i = 10;
int j = 3;
float f1 = i / j;
float f2 = (float ) i / j;
float f3 = (float ) (i / j);
```

Output:

f1 = 3.0

f2 = 3.333333

f3 = 3.0