**Keywords, Identifier, Literals, Operators and Expression Assignment**

1. Choose all valid identifiers

a. int int

b. int \_numvalue

c. float price\_money

d. char name1234567890123456789012345678901234567890

e. char name value

f. char $name

🡺In C, valid identifiers must start with a letter (uppercase or lowercase) or an underscore (\_), followed by letters, digits, or underscores. They cannot be keywords.

* **a. int int:** Invalid (int is a keyword)
* **b. int \_numvalue:** Valid
* **c. float price\_money:** Valid
* **d. char name1234567890123456789012345678901234567890:** Valid (though impractically long)
* **e. char name value:** Invalid (contains a space)
* **f. char $name:** Invalid (contains a special charactername∗∗:Invalid(containsaspecialcharacter)

2. What is the meaning of the following keywords, show the usage.

a. auto

b. extern

c. volatile

d. sizeof

e. const

**a. auto:** The auto keyword is used for automatic variables. By default, all local variables are automatic.

void function() {

auto int x = 10; // same as int x = 10;

}

**b. extern:** The extern keyword is used to declare a global variable or function in another file.

extern int num; // declaration

int main() {

printf("%d", num); // usage

}

**c. volatile:** The volatile keyword tells the compiler that the value of the variable may change at any time, preventing optimization.

volatile int flag = 1;

while (flag) {

// do something

}

**d. sizeof:** The size of operator returns the size of a variable or data type.

int x;

printf("%lu", sizeof(x)); // prints the size of int

**e. const:** The const keyword is used to declare variables whose value cannot be changed.

const int y = 10;

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

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";:** ptr is a pointer to a string literal “ABC”. The contents of the string literal cannot be modified.

**b. char arr[] = "ABC";:** arr is an array initialized with the string “ABC”. The contents of the array can be modified.

**ptr:** You cannot modify the contents of ptr because it points to a string literal, which is read-only.

**arr:** You can modify the contents of arr because it is an array with its own copy of the string.

String Literal: The string literal is "ABC" in the case of ptr.

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);

}

🡺void main() {

int a = 5, b = 5;

printf("\n%d %d", a--, --b); // Output: 5 4

printf("\n%d %d", b++, --b); // Output: 4 3

}

🡺a-- prints 5 and then decrements a to 4.

🡺b decrements b to 4 and then prints 4.

🡺b++ prints 4 and then increments b to 5.

🡺--b decrements b to 4 and then prints 3.

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;

}

🡺**Fix the Code Snippet:** The error is due to the const keyword, which makes num read-only. Also, k is used uninitialized.

#include<stdio.h>

int main()

{

int i, k = 0;

const int num = 0;

for(i = 0; i < 9; i++) {

k = k + 1;

}

/\* num = num + k; Cannot modify a const variable\*/

printf("final value of k: %d\n", k);

printf("value of num: %d\n", num);

return 0;

}

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);

}

🡺 **Evaluate the value of f1, f2 and f3.**

int main()

{

int i = 10;

int j = 3;

float f1 = i / j; /\* Integer division, f1 = 3.0 \*/

float f2 = (float) i / j; /\* Floating-point division, f2 = 3.333333 \*/

float f3 = (float) (i / j); /\* Integer division first, then cast, f3 = 3.0 \*/

}

* f1: 3.0 (integer division)
* f2: 3.333333 (floating-point division)
* f3: 3.0 (integer division, then cast to float)