for ( init; condition; increment) {  
 statement(s);  
}

while(condition) {  
 statement(s);  
}

Printf: **%[flags][width][.precision][length]specifier**

| **Specifier & Output** |
| --- |
| **C-**Character |
| **d or I -**Signed decimal integer |
| **E** Scientific notation (mantissa/exponent) using e character |
| **E** Scientific notation (mantissa/exponent) using E character |
| **F** Decimal floating point |
| **G** Uses the shorter of %e or %f |
| **G** Uses the shorter of %E or %f |
| **O** Signed octal |
| **S** String of characters |
| **U** Unsigned decimal integer |
| **X** Unsigned hexadecimal integer |
| **X** Unsigned hexadecimal integer (capital letters) |
| **P** Pointer address |
| **N** Nothing printed |
| **%** Character |

#include<stdio.h>

int fun()

{

static int count = 0;

count++;

return count;

}

int main()

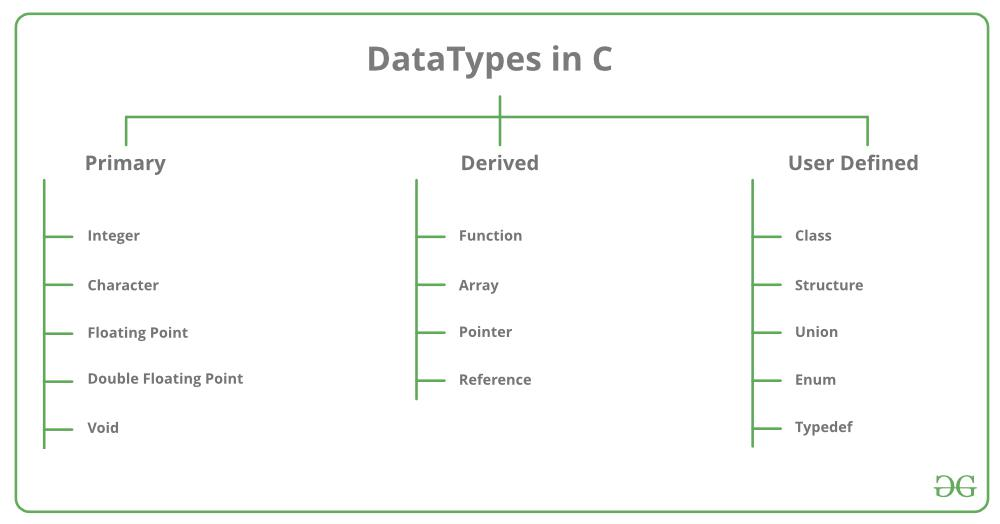
{

printf("%d ", fun());

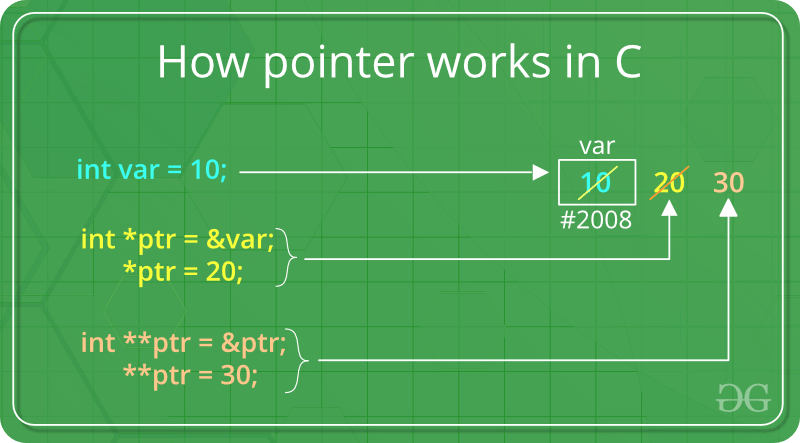
printf("%d ", fun());

return 0;

}



| Data Type | Memory (bytes) | Range | Format Specifier |
| --- | --- | --- | --- |
| short int | 2 | -32,768 to 32,767 | %hd |
| unsigned short int | 2 | 0 to 65,535 | %hu |
| unsigned int | 4 | 0 to 4,294,967,295 | %u |
| int | 4 | -2,147,483,648 to 2,147,483,647 | %d |
| long int | 4 | -2,147,483,648 to 2,147,483,647 | %ld |
| unsigned long int | 4 | 0 to 4,294,967,295 | %lu |
| long long int | 8 | -(2^63) to (2^63)-1 | %lld |
| unsigned long long int | 8 | 0 to 18,446,744,073,709,551,615 | %llu |
| signed char | 1 | -128 to 127 | %c |
| unsigned char | 1 | 0 to 255 | %c |
| float | 4 | 1.2E-38 to 3.4E+38 | %f |
| double | 8 | 1.7E-308 to 1.7E+308 | %lf |
| long double | 16 | 3.4E-4932 to 1.1E+4932 | %Lf |



| // C program to illustrate Pointers  #include <stdio.h>    **void** geeks()  {  **int** var = 20;    // declare pointer variable  **int**\* ptr;    // note that data type of ptr and var must be same  ptr = &var;    // assign the address of a variable to a pointer  **printf**("Value at ptr = %p \n", ptr);  **printf**("Value at var = %d \n", var);  **printf**("Value at \*ptr = %d \n", \*ptr);  }    // Driver program  **int** main()  {  geeks();  **return** 0;  } |
| --- |

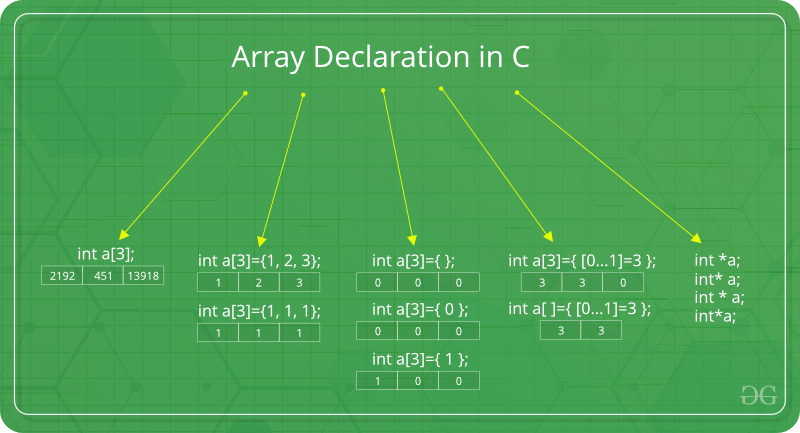
**Output**Value at ptr = 0x7ffd15b5deec   
Value at var = 20   
Value at \*ptr = 20

#include <stdio.h>  
#include <stdbool.h>  
  
#define MAX 7  
  
int intArray[MAX] = {4,6,3,2,1,9,7};  
  
void printline(int count) {  
 int i;  
   
 for(i = 0;i < count-1;i++) {  
 printf("=");  
 }  
   
 printf("=\n");  
}  
  
void display() {  
 int i;  
 printf("[");  
   
 // navigate through all items   
 for(i = 0;i < MAX;i++) {  
 printf("%d ",intArray[i]);  
 }  
   
 printf("]\n");  
}  
  
void swap(int num1, int num2) {  
 int temp = intArray[num1];  
 intArray[num1] = intArray[num2];  
 intArray[num2] = temp;  
}  
  
int partition(int left, int right, int pivot) {  
 int leftPointer = left -1;  
 int rightPointer = right;  
  
 while(true) {  
 while(intArray[++leftPointer] < pivot) {  
 //do nothing  
 }  
   
 while(rightPointer > 0 && intArray[--rightPointer] > pivot) {  
 //do nothing  
 }  
  
 if(leftPointer >= rightPointer) {  
 break;  
 } else {  
 printf(" item swapped :%d,%d\n", intArray[leftPointer],intArray[rightPointer]);  
 swap(leftPointer,rightPointer);  
 }  
 }  
   
 printf(" pivot swapped :%d,%d\n", intArray[leftPointer],intArray[right]);  
 swap(leftPointer,right);  
 printf("Updated Array: ");   
 display();  
 return leftPointer;  
}  
  
void quickSort(int left, int right) {  
 if(right-left <= 0) {  
 return;   
 } else {  
 int pivot = intArray[right];  
 int partitionPoint = partition(left, right, pivot);  
 quickSort(left,partitionPoint-1);  
 quickSort(partitionPoint+1,right);  
 }   
}  
  
int main() {  
 printf("Input Array: ");  
 display();  
 printline(50);  
 quickSort(0,MAX-1);  
 printf("Output Array: ");  
 display();  
 printline(50);  
}

| #include <stdio.h>  #include <stdlib.h>    // Creating the structure  **struct** student {  **char** name[80];  **int** age;  **float** percentage;  };    // Creating the structure object  **struct** student\* emp = NULL;    // Driver code  **int** main()  {  // Assigning memory to struct variable emp  emp = (**struct** student\*)  **malloc**(**sizeof**(**struct** student));    // Assigning value to age variable  // of emp using arrow operator  emp->age = 18;    // Printing the assigned value to the variable  **printf**("%d", emp->age);    **return** 0;  } |
| --- |

**Output:**

18



binarySearch(arr, x, low, high)  
 repeat till low = high  
 mid = (low + high)/2  
 if (x == arr[mid])  
 return mid  
   
 else if (x > arr[mid]) // x is on the right side  
 low = mid + 1  
   
 else // x is on the left side  
 high = mid - 1

binarySearch(arr, x, low, high)  
 if low > high  
 return False   
   
 else  
 mid = (low + high) / 2   
 if x == arr[mid]  
 return mid  
   
 else if x > arr[mid] // x is on the right side  
 return binarySearch(arr, x, mid + 1, high)  
   
 else // x is on the left side  
 return binarySearch(arr, x, low, mid - 1)

| // C program for implementation of Bubble sort  #include <stdio.h>    **void** swap(**int**\* xp, **int**\* yp)  {  **int** temp = \*xp;  \*xp = \*yp;  \*yp = temp;  }    // A function to implement bubble sort  **void** bubbleSort(**int** arr[], **int** n)  {  **int** i, j;  **for** (i = 0; i < n - 1; i++)    // Last i elements are already in place  **for** (j = 0; j < n - i - 1; j++)  **if** (arr[j] > arr[j + 1])  swap(&arr[j], &arr[j + 1]);  }    /\* Function to print an array \*/  **void** printArray(**int** arr[], **int** size)  {  **int** i;  **for** (i = 0; i < size; i++)  **printf**("%d ", arr[i]);  **printf**("\n");  }    // Driver program to test above functions  **int** main()  {  **int** arr[] = { 5, 1, 4, 2, 8 };  **int** n = **sizeof**(arr) / **sizeof**(arr[0]);  bubbleSort(arr, n);  **printf**("Sorted array: \n");  printArray(arr, n);  **return** 0;  } Dedicated functions for string comparison and assignment: strcmp(string1, string2)// returns negative, 0 or positivestrcpy(string1, string2) // string2 is copied in string1 |
| --- |