C/C++ Interview Questions

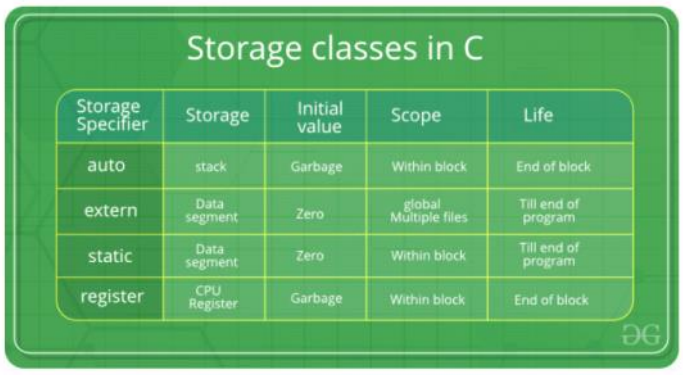
07 March 2022 14:29

© RAJAT KUMAR
https://www.linkedin.com/in/imRajat/
https://github.com/im-Rajat

Storage Classes in C :

https://www.geeksforgeeks.org/storage-classes-in-c/

C language uses 4 storage classes, namely:



Storage classes in C

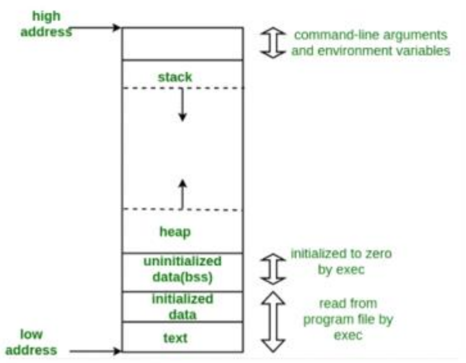
Storage Specifier	Storage	Initial value	Scope	Life
auto	stack	Garbage	Within block	End of block
extern	Data segment	Zero	global Multiple files	Till end of program
static	Data segment	Zero	Within block	Till end of program
register	CPU Register	Garbage	Within block	End of block

Memory Layout of C Programs :

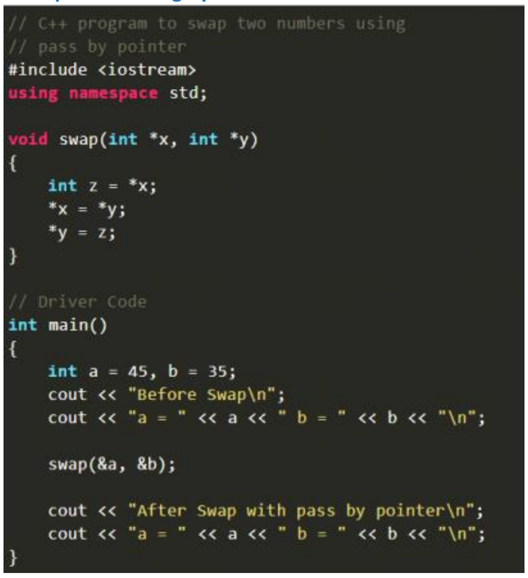
https://www.geeksforgeeks.org/memory-layout-of-c-program/

A typical memory representation of a C program consists of the following sections.

1. Text segment (i.e. instructions)
2. Initialized data segment
3. Uninitialized data segment (bss)
4. Heap
5. Stack


high address
command-line arguments and environment variables
stack
heap
uninitialized data(bss)
initialized to zero by exec
initialized data
read from program file by exec
text
low address

Example of Passing by Pointer:



```
// C++ program to swap two numbers using
// pass by pointer
#include <iostream>
using namespace std;

void swap(int *x, int *y)
{
    int z = *x;
    *x = *y;
    *y = z;
}

// Driver Code
int main()
{
    int a = 45, b = 35;
    cout << "Before Swap\n";
    cout << "a = " << a << " b = " << b << "\n";

    swap(&a, &b);

    cout << "After Swap with pass by pointer\n";
    cout << "a = " << a << " b = " << b << "\n";
}
```

Example of Passing by Reference:

```
// C++ program to swap two numbers using
```

```
#include<stdio.h>

#include <iostream>
#include <bits/stdc++.h>
using namespace std;
```

Q) Structure and Dynamic Memory

```
struct Node {
        int data;
        struct Node* next;
};

Struct Node* n1 = (Struct Node*)malloc(sizeof(struct Node));
n1->data = 1;
n1->next = NULL;

class Node      // using class instead of struct
{
   private:
       // private variables

   public:
      int data;
      Node* next;
      Node(int data, Node *next)
      {
         this->data = data;
         this->next = next;
      }
   };

Node *n1 = new Node(1, NULL);
// n1->next = NULL;
```

Q) dynamic memory allocation?
- int *p= malloc(sizeof(int)*10);

Q) malloc()
- The malloc() function is used to allocate the memory during the execution of the program.
- It does not initialize the memory but carries the garbage value.

Q) calloc()
- The calloc() is same as malloc() function, but the difference only is that it initializes the memory with zero value.

Q) What is the use of a static variable in C?
- The static variable retains its value between multiple function calls.
- Static variables are used because the scope of the static variable is available in the entire program
- The static variable is initially initialized to zero

```
void f() {
  static int i;
  ++i;
  printf("%d ",i);
}
```

Q) What is a pointer in C?
- A pointer is a variable that refers to the address of a value
  int *p; //pointer of type integer.
  int a=5;
  p=&a;

Q) NULL pointer in C?
- A pointer that doesn't refer to any address of value but NULL.
- When we assign a '0' value to a pointer of any type, then it becomes a Null pointer.

Q) far pointer in C?
- A pointer which can access all the 16 segments (whole residence memory) of RAM is known as far pointer.
- A far pointer is a 32-bit pointer that obtains information outside the memory in a given section.

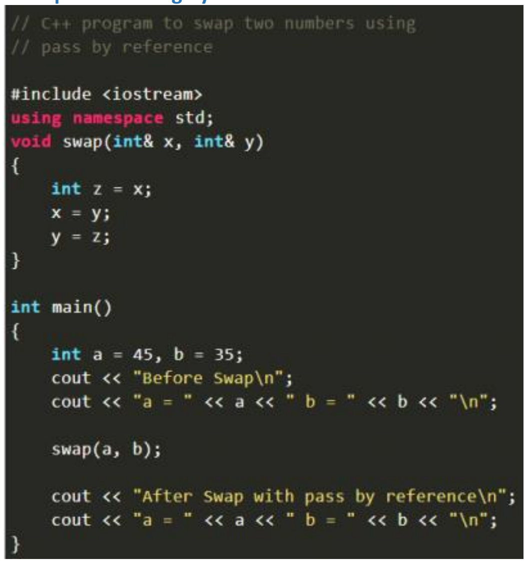
Q) dangling pointer in C?
- Dangling pointer arises when an object is deleted without modifying the value of the pointer. The pointer points to the deallocated memory.
- a pointer is pointing any memory location, but meanwhile another pointer deletes the memory occupied by the first pointer while the first pointer still points to that memory location, the first pointer will be known as a dangling pointer.

```
int *ptr = malloc(constant value); //allocating a memory space.
free(ptr); //ptr becomes a dangling pointer.
ptr=NULL; //Now, ptr is no longer a dangling pointer.
```

Q) What is polymorphism in C++?
- Polymorphism in simple means having many forms.
- Its behavior is different in different situations.
- The two types of polymorphism in c++ are:

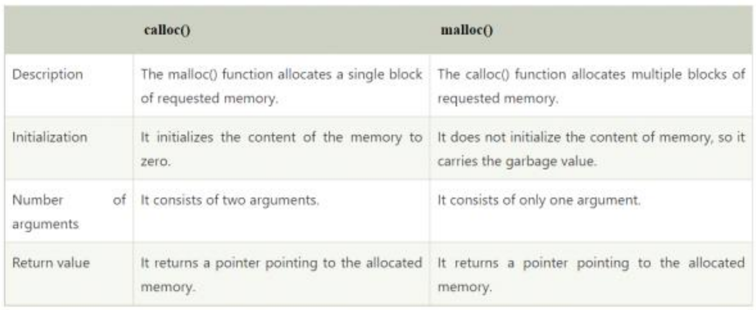
```
}
```

Example of Passing by Reference:



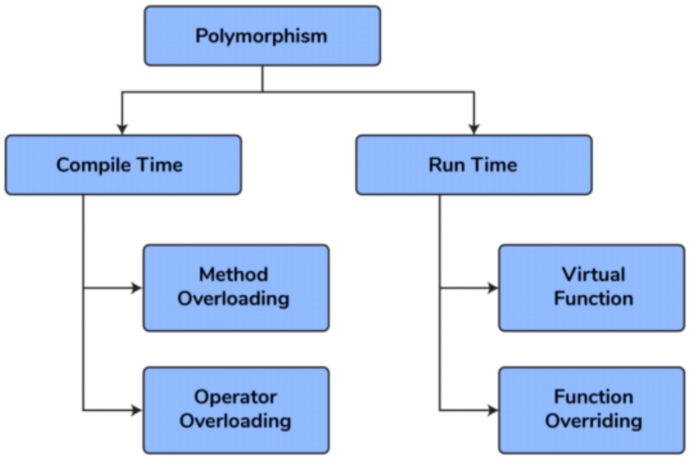
```
// C++ program to swap two numbers using
// pass by reference

#include <iostream>
using namespace std;
void swap(int& x, int& y)
{
    int z = x;
    x = y;
    y = z;
}

int main()
{
    int a = 45, b = 35;
    cout << "Before Swap\n";
    cout << "a = " << a << " b = " << b << "\n";

    swap(a, b);

    cout << "After Swap with pass by reference\n";
    cout << "a = " << a << " b = " << b << "\n";
}
```

Difference between malloc() and calloc()?

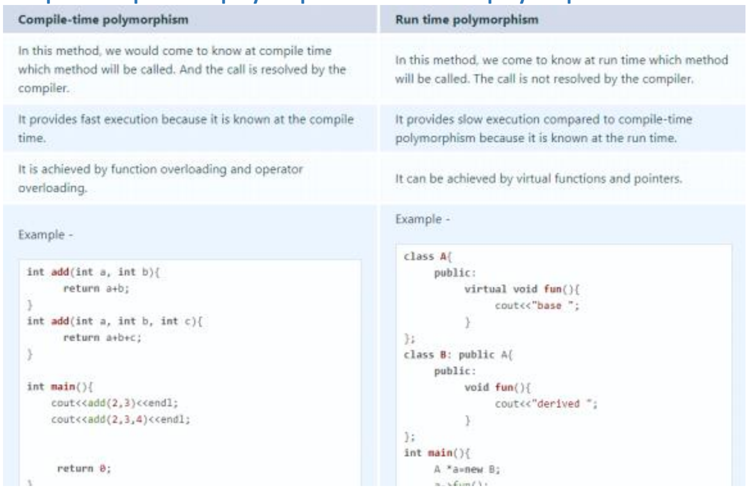


	calloc()	malloc()
Description	The malloc() function allocates a single block of requested memory.	The calloc() function allocates multiple blocks of requested memory.
Initialization	It initializes the content of the memory to zero.	It does not initialize the content of memory, so it carries the garbage value.
Number of arguments	It consists of two arguments.	It consists of only one argument.
Return value	It returns a pointer pointing to the allocated memory.	It returns a pointer pointing to the allocated memory.

Polymorphism



Compare compile time polymorphism and Runtime polymorphism



Compile-time polymorphism	Run time polymorphism
In this method, we would come to know at compile time which method will be called. And the call is resolved by the compiler.	In this method, we come to know at run time which method will be called. The call is not resolved by the compiler.
It provides fast execution because it is known at the compile time.	It provides slow execution compared to compile-time polymorphism because it is known at the run time.
It is achieved by function overloading and operator overloading.	It can be achieved by virtual functions and pointers.
Example -	Example -

```
int add(int a, int b){
      return a+b;
}
int add(int a, int b, int c){
      return a+b+c;
}

int main(){
    cout<<add(2,3)<<endl;
    cout<<add(2,3,4)<<endl;


     return 0;
```

```
class A{
     public:
          virtual void fun(){
               cout<<"base ";
          }
};
class B: public A{
     public:
          void fun(){
               cout<<"derived ";
          }
};
int main(){
     A *a=new B;
```

Q) What is polymorphism in C++?
- Polymorphism in simple means having many forms.
- Its behavior is different in different situations.
- The two types of polymorphism in c++ are:
  - Compile Time Polymorphism
  - Runtime Polymorphism

Q) Toggling k-th bit of a number
- return (n ^ (1 << (k-1)));
- Or num = (num ^ (1 << (k-1)));

Q) Constructor and Destructor Calling
- Parent Constructor -> Child Constructor
- Child Destructor -> Parent Destructor

Q) void pointers?
- a pointer which is having no datatype associated with it. It can hold addresses of any type.

  ```
  void *ptr;
  char *str;
  ptr=str;         // no error
  str=ptr;         // error because of type mismatch
  ```

- We can assign a pointer of any type to a void pointer but the reverse is not true unless you typecast it as

  ```
  str=(char*) ptr;
  ```

Q) abstract class?
- Class which has at least one pure virtual function
- In C++ class is made abstract by declaring at least one of its functions as <>strong> pure virtual function.
- A pure virtual function is specified by placing "= 0" in its declaration.
- Its implementation must be provided by derived classes.
- A class containing the pure virtual function cannot be used to declare the objects of its own, such classes are known as abstract base classes.

Q) Data Abstraction in C++
- Abstraction means displaying only essential information and hiding the details.
- Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.
- process of providing only the essential details to the outside world and hiding the internal details, i.e., representing only the essential details in the program.

Q) C++ Friend function
- If a function is defined as a friend function in C++, then the protected and private data of a class can be accessed using the function.

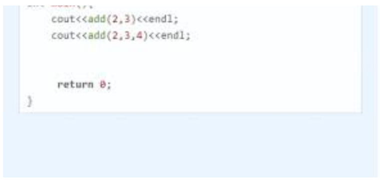
  ```
  class class_name
  {
     friend data_type function_name(argument/s);         // syntax of friend function.
  };

  class Box
  {
     private:
        int length;
     public:
        Box(): length(0) { }
        friend int printLength(Box); //friend function
  };
  int printLength(Box b)
  {
    b.length += 10;
     return b.length;
  }
  int main()
  {
     Box b;
     cout<<"Length of box: "<< printLength(b)<<endl;
     return 0;
  }
  ```

Q) Inheritance

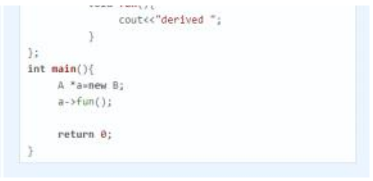
  ```
  class Programmer: public Account {
            public:
            float bonus = 5000;
     };
  ```

Q) Copy Constructor
- A Copy constructor is an overloaded constructor used to declare and initialize an object from another object.
- Class_name(const class_name &old_object);

```
#include <iostream>
using namespace std;
class A
{
   public:
    int x;
    A(int a)           // parameterized constructor.
    {
      x=a;
```

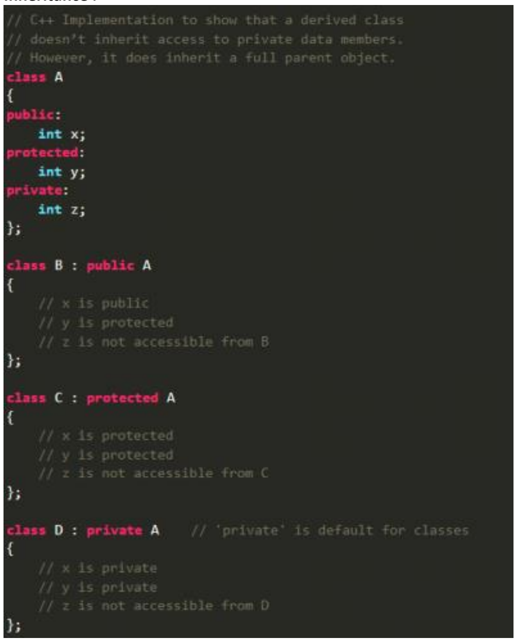


```
    cout<<add(2,3)<<endl;
    cout<<add(2,3,4)<<endl;


     return 0;
}
```



```
            cout<<"derived ";
        }
};
int main(){
    A *a=new B;
    a->fun();

    return 0;
}
```

Inheritance :



```
// C++ Implementation to show that a derived class
// doesn't inherit access to private data members.
// However, it does inherit a full parent object.
class A
{
public:
    int x;
protected:
    int y;
private:
    int z;
};

class B : public A
{
    // x is public
    // y is protected
    // z is not accessible from B
};

class C : protected A
{
    // x is protected
    // y is protected
    // z is not accessible from C
};

class D : private A    // 'private' is default for classes
{
    // x is private
    // y is private
    // z is not accessible from D
};
```

```
    int x;
    A(int a)             // parameterized constructor.
    {
      x=a;
    }
    A(A &i)             // copy constructor
    {
        x = i.x;
    }
};
int main()
{
  A a1(20);             // Calling the parameterized constructor.
  A a2(a1);             //  Calling the copy constructor.
  cout<<a2.x;
  return 0;
}
```

Q) Fix Memory Leakage

```
Person * p3 = NULL; //NullPtr;
p3 = new Person();

p3->printFunc();


if(p3 != NULL)
{
  delete p3;
  p3 = NULL;
}
```