POINTERS IN C++

pointers are variables that stores the address of another variable

```
int a;
int *p;
p = &a;
a = 4;
print p  it gives address
print *p it gives the value
```

deferencing

pointers are statically typed because we can dereference it

void pointers

void* pointer;

we cannot dereference void pointer

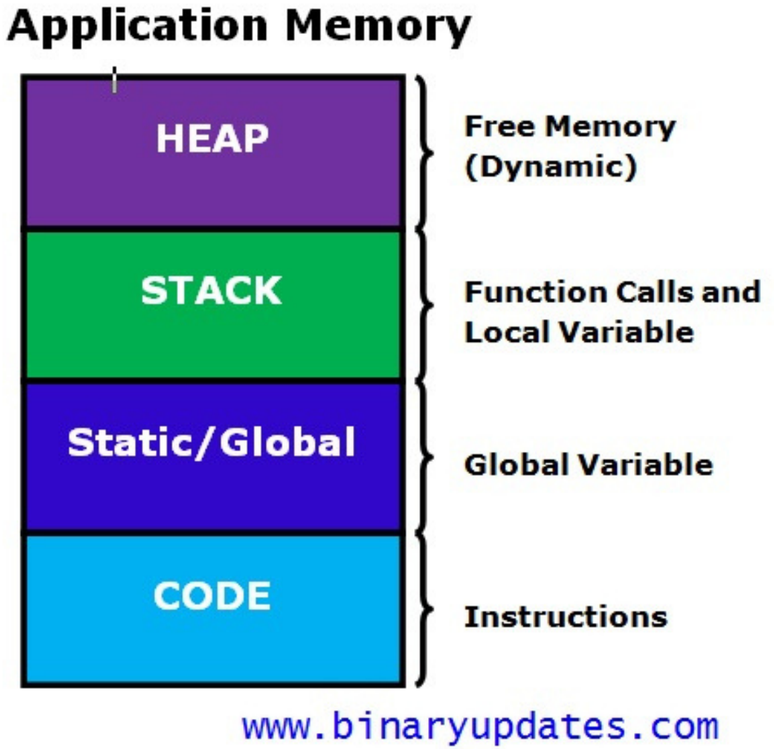
pointer to pointer

```
int x = 5;
int* p = &x;
*p = 4;
int **q = &p;
int ***r = &q;
cout<<x;
```

pointers as function arguments call by reference

call by reference saves lot of memory when compared to call by value

```
void add(int *p){
    *p +=1;
}
```



pointers and arrays

```
int a[] ={1,2,3};
   cout<<a<<endl;
   cout<<a[0]<<endl;
   cout<<*a<<endl;
```

```
char c[4] = "car";
   char* p =c;
   cout<<*(p+1);
```

arrays are always passed to a function by reference

*c or c[0] or c[0][0] are gives the first value

Dynamic Memory

stack memory

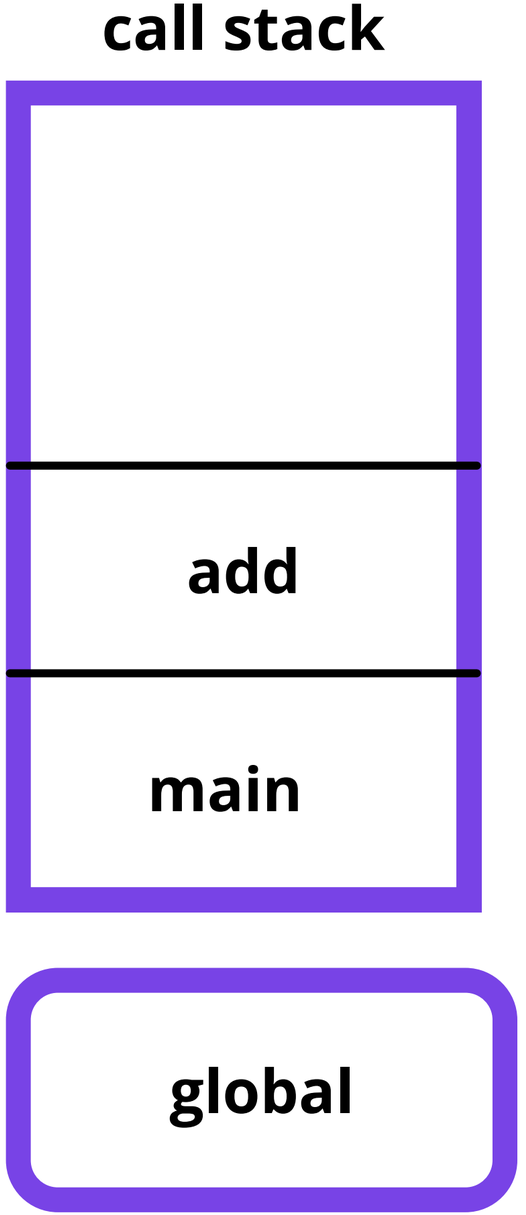
```
int add(int* a){
    int n = sizeof(a)/sizeof(a[0]);
    int ans=0;
    for(int i=0;i<n;i++){
        ans += a[i];
    }
    return ans;
}

int main(){
   int a[] = {1,2,3,4,5,6};
   cout<<add(a);
}
```

stack memory does not grow

after execution of the program the stack memory is cleared

during recursion program it leads to stack overflow



Heap

heap is free pool of memory,it grows dynamically

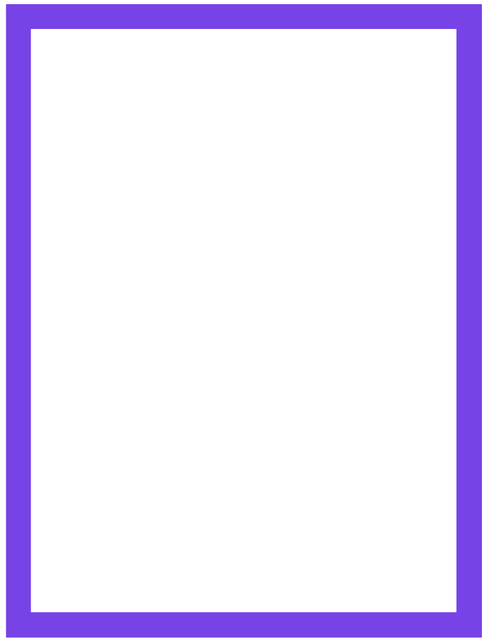
after allocation of memory in heap we need to clear it explicitly

new operator allocates memory in heap

delete deallocates the memory in heap

```
int main(){
   int a;
   int *p;
   p = new int;
   *p = 10;
   delete p;
   p = new int[20];
   delete[] p;
}
```

heap



Memory allocation in heap

malloc

int *p = (int*)malloc(3*sizeof(int));

malloc initializes the array with garbage value

calloc

calloc initializes elements with zero

int *p = (int*)calloc(3,sizeof(int));

realloc

int *r = (int*)realloc(p,2*3*sizeof(int));

call by value

```
int add(int a){ //called function
   return a+1;
}

int main(){ // calling function
   int a=10;
   int b=add(a); //call by value
   cout<<b<<endl;
}
```

call by reference

```
int add(int* a){ //called function
   return a+1;
}

int main(){ // calling function
   int a=10;
   int b=add(&a); //call by reference
   cout<<b<<endl;
}
```

Function Pointers

it stores the address of function

```
int add(int a){ //called function
   return a+1;
}

int main(){ // calling function
   int a;
   int (*p)(int); //fucntion pointer
   p = &add;
   a = (*p)(2);
   cout<<(a)<<endl;
}
```

Function pointers and callbacks

```
void a(){
    cout<<"hello";
}

void b(void(*ptr)()){   //it takes function pointer as parameter
    ptr();
}

int main(){
    b(a);
}
```

Memory Leak

it happens unproper use of memory in heap

Memory leakage occurs in C++ when programmers allocates memory by using new keyword and forgets to deallocate the memory by using delete() function or delete[] operator. One of the most memory leakage occurs in C++ by using wrong delete operator.
The delete operator should be used to free a single allocated memory space, whereas the delete [] operator should be used to free an array of data values. If a program has memory leaks, then its memory usage is satirically increasing since all systems have limited amount of memory and memory is costly. Hence it will create problems.