Templates

- When same logic/algorithm is applicable for different data types, templates are used.
- There are two types of templates:
  - Template function
  - Template class
- Template function is written using template keyword and generic argument is given by class or typename keyword.
- Based on args passed, compiler generates code for the function by replacing generic type with intended type in template function.

class

generic arg

```
template < typename T >
void swap( T *a, T *b)
{
    T t = *a;
      *a = *b;
      *b = t;
}
int main() {
    int x=10, y=20;
  ✓ swap(&x, &y);
    double p=1.1, q=2.2;
  ✓ swap(&p, &q);
    student s1, s2;
  ✓ swap(&s1, &s2);
}
```

swap<int>

swap<double>

swap<student>

```
void swap(int *a, int *b)
{
    int t = *a;
    *a = *b;
    *b = t;
}   ✓
```

```
void swap(double* a, double* b)
{
double t = *a;
    *a = *b;
    *b = t;
}   ✓
```

```
void swap(Student *a, Student *b)
{
Student t = *a;
    *a = *b;
    *b = t;
}   ✓
```

```
// specialized template for,
void Swap (char *a, char *b)
{
    char t[n];
    strcpy(t, a);
    strcpy(a, b);
}   strcpy(b, t);
```

Templates

- We can also implement specialized template function, if need a different logic for any speical data type.
- Template arguments may have constant values.
- Like template function, template class is also declared using template keyword.
- All functions of template class are by default template functions.
- Most common application of template class is to implement data structures.

array stack queue LL tree

```
const int N = 8;

template <typename T>
class array {
        T arr[N];
public:
        array() {
            memset(arr, 0, sizeof(arr));
        }
        T get(int index) {
            if(index<0 || index >= N)
                exit(1);
            return arr[index];
        }
        void set(int index, T val){
            if(index<0 || index>=N)
                return;
            arr[index] = val;
        }
};
```

```
const int N = 8;
class array {
      int arr[N];
public:
      array() {
          for(i=0; i<N; i++)
              arr[i] = 0;
      }
      int get(int index) {
          if(index<0 || index >= N)
              return -1;
          return arr[index];
      }
      void set(int index, int val){
          if(index <0 || index>=N)
              return;
          arr[index] = val;
      }
};
```

```
const int N = 8;
class array {
      double arr[N];
public:
      array() {
          for(i=0; i<N; i++)
              arr[i] = 0.0;
      }
      double get(int index) {
          if(index<0 || index >= N)
              return -1;
          return arr[index];
      }
      void set(int index, double val){
          if(index <0 || index>=N)
              return;
          arr[index] = val;
      }
};
```

```
const int N = 8;
class array {
    Student arr[N];
public:
    array() {
        memset(arr, 0, N * sizeof(Student));
    }
    Student get(int index) {
        if(index<0 || index >= N)
            exit(1);
        return arr[index];
    }
    void set(int index, Student val){
        if(index <0 || index >=N)
            return;
        arr[index] = val;
    }
};
```

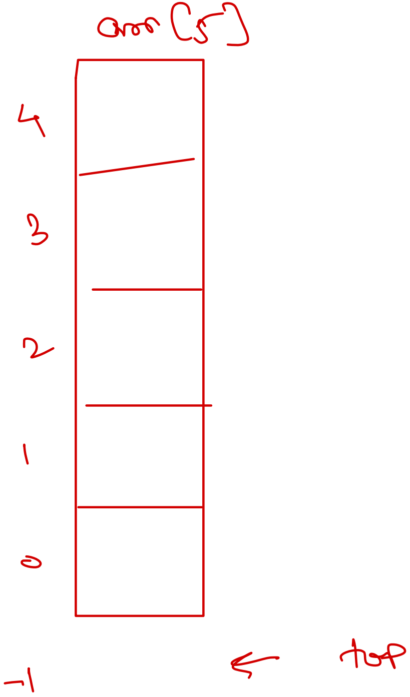
<string.h>

base addr

value

n bytes

Stack using array

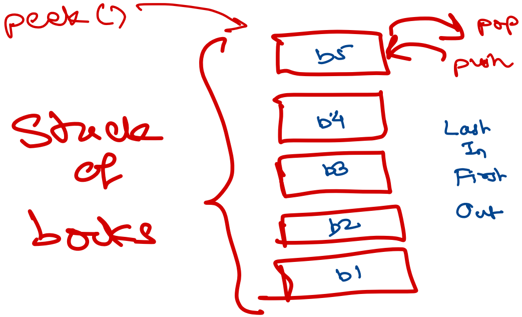


```
push()
    top++;
    arr[top] = val;

pop()
    top--;

peek()
    return arr[top];

empty()
    top == -1
```



Stack operations

1. push()
2. pop()
3. peek()
4. empty()