

PROJECT REPORT ON  
**“Generic Collection Library In C”**  
**(STL in C)**



Submitted By:  
**Mr. Chavan Digvijay Changdev**  
&  
**Mr. Aurangabadkar Yuvraj Prabhuling**

Under the Guidance of:  
**Prof. Chandole N.S.**  
&  
**Prof. Patil S.V.**

Submitted to  
**Punyashlok Ahilyadevi Holkar University, Solapur**

In Practical Fulfilment of the  
**Bachelor Degree of Computer Science**  
**(Entire Computer Science)-III (2023-2024)**



S. T. U. S. Mandal's  
Sangola Mahavidyalaya, Sangola

## **CERTIFICATE**

Seat No.:

Date:    /    /2024

Seat No.:

This is to certify that,

**Mr.Chavan Digvijay Changdev** and  
**Mr.Aurangabadkar Yuvraj Prabhuling** has successfully  
completed the project titled “**Generic Collection Library In C  
(C-STL)**” at Sangola Mahavidyalaya, Sangola under guidance  
and supervision of Prof. **Chandole N.S. & Patil S.V.** in  
fulfilment of requirements of final year, Bachelor of Computer  
Science (Entire Computer Science) of working year 2023-2024.

Place: Sangola

Date:    /    /2024

Seen By  
Prof. Chandole N.S.  
(Project Guide)

External Examiner

H. O. D.  
Prof. Tate R. R.  
(Dept. of Computer Science)

# **DECLARATION**

To,  
The principal,  
Sangola Mahavidyalaya, Sangola.

Respected sir,

We, hereby declare that the project titled “**Generic Collection Library In C (C-STL)**” is the original and developed under guidance of **Prof. Chandole N.S.** and **Patil S.V.** at Sangola Mahavidyalaya, Sangola. All sources of information used in this report have been duly acknowledged.

We further declare that we have not copied this project from other which submitted to Punyashlok Ahilyadevi Holkar Solapur University, Solapur earlier.

Place: Sangola

Date:    /    /2024

**Mr. Chavan D.C.**

**Mr. Aurangabadkar Y.P.**

# **ACKNOWLEDGEMENT**

I would like to express my sincere gratitude to Department of Computer Science and other faculty members of Sangola Mahavidyalaya, Sangola for giving us to give never-ending support and for successful completion of the project.

I am also thankful to **Prof. Chandole N.S. & Prof. Patil S.V.** their invaluable guidance, support, and motivation throughout the development of the project titled- **“Generic Collection Library In C (STL in C)”**. Their expertise and insights have been crucial in shaping the direction of this work.

**Mr. Chavan Digvijay Changdev**

**Mr. Aurangabadkar Yuvraj Prabhuling**

(Dept. of Computer Science,  
Sangola Mahavidyalaya, Sangola)

# INDEX

Sr. No.	Name	Page No.
1.	Introduction	7
2.	Requirements	8
3.	Feasibility Study	9
4.	Silent Features	11
5.	<i>vector</i>	13
6.	<i>stack</i>	21
7.	<i>queue</i>	29
8.	<i>list</i>	38
9.	Bibliography	50

# **LICENSE**

MIT License

Copyright (c) 2023 Digvijay Changdev Chavan

Permission is hereby granted, free of charge, to any person obtaining a  
copy

of this software and associated documentation files (the "Software"), to  
deal

in the Software without restriction, including without limitation the rights  
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell  
copies of the Software, and to permit persons to whom the Software is  
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in  
all

copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,  
EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF  
MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO  
EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES  
OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,  
ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER  
DEALINGS IN THE  
SOFTWARE.

# INTRODUCTION

The C-STL (C Standard Template Library) is a generic collection library for the C programming language. It provides a set of template-based containers and algorithms, inspired by the C++ Standard Template Library (STL), to facilitate data management and manipulation in C programs.

## ■ Objective-

The primary objective of C-STL is to offer reusable, efficient, and generic data structures and algorithms to C programmers. By providing a comprehensive collection of containers and utilities, C-STL aims to enhance code quality, maintainability, and performance in C projects.

## ■ Features-

- 1. Generic containers:** Vector (Dynamic Array), List, Stack, Queue etc.
- 2. Dynamic.**
- 3. Memory management utilities.**
- 4. Iterator support for traversing containers.**
- 5. Customizable and extensible design.**

# **REQUIREMENT**

Requirements for use of generic collection library in C typically include:

## **Minimum Hardware Requirements :**

- 1. 500 MB Ram**
- 2. I-3 Processor**
- 3. Hard Disk**

## **Minimum Software Requirements :**

- 1. Windows 10, Linux (Any Distribution).**
- 2. Visual Studio Code, Notepad**



# **FEASIBILITY STUDY**

Feasibility Study: CSTL (C Standard Template Library)

## **1. Technical Feasibility:**

**Assessment:** The project is technically feasible as it involves implementing data structures and algorithms in C, a language well-suited for low-level systems programming.

**Skills and Expertise:** The team possesses strong expertise in C programming, data structures, and algorithms, ensuring the ability to develop and maintain the library effectively.

**Compatibility:** The library will be designed to be compatible with commonly used C compilers, operating systems, and architectures.

## **2. Economic Feasibility:**

**Cost Estimation:** The primary costs involve personnel for development and maintenance, with minimal infrastructure costs. No significant licensing or third-party component costs are anticipated.

**Market Analysis:** There's a demand for generic collection libraries in C, especially in embedded systems, system programming, and performance-critical applications.

**Revenue Potential:** While the library may be offered as open-source, potential revenue streams could include consulting services, customization, or premium support.

### **3. Operational Feasibility:**

**Resource Availability:** Required resources, including skilled developers, development tools, and hardware, are readily available or can be acquired within the project's timeframe.

**Timeline:** A realistic timeline has been developed, considering development phases, testing, documentation, and potential contingencies.

**Risk Assessment:** Identified risks, such as technical challenges or changes in requirements, have mitigation strategies in place to minimize their impact.

### **4. Legal and Regulatory Feasibility:**

**Intellectual Property:** Efforts will be made to ensure the library is developed using original code or properly attributed open-source components to avoid intellectual property issues.

**Compliance:** The library will comply with relevant open-source licenses, export regulations, and data privacy laws.

### **5. Social and Environmental Feasibility:**

**Impact Assessment:** The project's impact on society and the environment is minimal, focusing primarily on software development with no direct adverse effects.

**Stakeholder Analysis:** Key stakeholders, including developers and potential users, are supportive of the project, ensuring a positive reception upon release.

### **Conclusion:**

The feasibility study indicates that the CSTL project is viable and promising. With technical, economic, operational, legal, and social considerations addressed, the project can proceed with confidence, offering potential benefits to users and stakeholders in the C programming community.

# SILENT FEATURES

---

Silent features in a generic collection library in C refer to its key characteristics and functionalities that contribute to its usability, versatility, and performance. Here are some silent features that such a library might offer:

**Genericity:** The library allows users to work with a wide range of data types by providing generic implementations of common data structures and algorithms. This enables developers to write reusable code that is independent of specific data types.

**Abstraction:** The library abstracts implementation details, providing a uniform interface for interacting with different data structures. This simplifies the development process and promotes code clarity and maintainability.

**Flexibility:** Users can easily customize and extend the library to suit their specific requirements. They can add new data structures, algorithms, or functionality without modifying the core library code.

**Type Safety:** Despite its generic nature, the library ensures type safety by employing techniques such as void pointers or macros for data manipulation. This helps prevent type mismatches and runtime errors.

**Efficiency:** The library is designed for efficiency, with optimized implementations of data structures and algorithms. It minimizes memory usage and maximizes performance, making it suitable for use in resource-constrained environments.

**Comprehensive Documentation:**

The library is accompanied by comprehensive documentation that includes usage examples, function descriptions, and guidelines for developers. This facilitates easy integration and usage of the library in projects.

**Error Handling:**

Robust error handling mechanisms are implemented to handle exceptional scenarios such as memory allocation failures or invalid operations gracefully. This enhances the reliability and stability of applications using the library.

**Portability:** The library is designed to be portable across different platforms and C compilers. It adheres to standard C conventions and avoids platform-specific dependencies, ensuring broad compatibility.

**Test Coverage:** The library is thoroughly tested to validate the correctness and efficiency of its implementations. Comprehensive test suites cover various scenarios and edge cases to ensure reliability and robustness.

**Community Support:** The library has an active community of users and contributors who provide support, share best practices, and contribute to its ongoing development and improvement.

# 1. Vector ->

## *vector.h*

```
#ifndef vector_h
```

```
#define vector_h
```

```
typedef struct{
```

```
    void *array;
```

```
    size_t size,data_size;
```

```
    char *type;
```

```
}vector;
```

```
vector *vector_create(size_t);
```

```
void vector_push_back(vector*, void*);
```

```
void *vector_pop_back(vector*);
```

```
void vector_insert(vector*,void*,size_t);
```

```
void vector_remove(vector*,size_t);
```

```
size_t vector_size(vector*);
```

```
void vector_swap(vector*,vector*);
```

```
short vector_ispresent(vector*,void*);
```

```
short vector_isempty(vector*);
```

```
void vector_clean(vector*);
```

```
void vector_reverse(vector*);
```

```
#endif
```

## *vector.c*

```
##include <stdlib.h>
```

```
#include <string.h>
```

```
#include <ctype.h>
```

```
#include "vector.h"
```

```
vector* vector_create(size_t size){  
    vector *vec=(vector*)malloc(sizeof(vector));  
    vec->array=NULL;  
    vec->size=0;  
    vec->data_size=size;  
    return vec;  
}
```

```
void vector_push_back(vector *vec, void *val){  
    vec->array=realloc(vec->array,(vec->size+1)*vec->data_size);  
    memcpy((char*)vec->array+vec->data_size*vec->size,val,vec->data_size);  
    vec->size++;  
}
```

```
void* vector_pop_back(vector *vec){  
    if (vec == NULL || vec->array == NULL || vec->size == 0)return NULL;  
    void *val = malloc(vec->data_size);  
    memcpy(val,(char *)vec->array + (vec->size - 1) * vec->data_size,vec->data_size);  
    vec->array = realloc(vec->array, (vec->size - 1) * vec->data_size);  
    vec->size--;  
    return val;  
}
```

```

void vector_insert(vector *vec, void *val, size_t position){
    if(position > vec->size+1)return;
    vec->array=realloc(vec->array,(vec->size+1)*vec->data_size);
    for (size_t index = vec->size; index > position; index--){
        void *dest = (char *)vec->array + index * vec->data_size;
        void *src = (char *)vec->array + (index - 1) * vec->data_size;
        memcpy(dest, src, vec->data_size);
    }
    void *dest=(char *)vec->array + position * vec->data_size;
    memcpy(dest, val, vec->data_size);
    vec->size++;
}

void vector_remove(vector *vec, size_t position){
    if(position>=vec->size)return;
    for(size_t index = position; index < vec->size - 1; index++){
        void *dest = (char *)vec->array + index * vec->data_size;
        void *src = (char *)vec->array + (index + 1) * vec->data_size;
        memcpy(dest, src, vec->data_size);
    }
    vec->array = realloc(vec->array, (vec->size - 1) * vec->data_size);
    vec->size--;
}

size_t vector_size(vector *vec){
    if(vec == NULL)return 0;
    return vec->size;
}

void vector_swap(vector *vec1, vector *vec2){
    if(vec1->data_size == vec2->data_size){

```

```

        vector *temp=malloc(sizeof(vector));
        *temp=*vec1;
        *vec1=*vec2;
        *vec2=*temp;
        free(temp);
    }
}

```

```

short vector_ispresent(vector *vec, void *val){
    if(vec==NULL)return 0;
    for(size_t index = 0; index < vec->size; index++){
        void *currentElement = (char *)vec->array + index * vec->data_size;
        if(memcmp(currentElement, val, vec->data_size) == 0)return 1;
    }
    return 0;
}

```

```

short vector_isempty(vector *vec){
    if(vec==NULL || vec->array==NULL || vec->size==0)return 1;
    return 0;
}

```

```

void vector_clean(vector *vec){
    free(vec->array);
    free(vec);
}

```

```

void vector_reverse(vector *vec){
    if(vec==NULL)return;
    void *temp=malloc(vec->data_size);
    for(size_t index = 0 ,back=vec->size-1; index < vec->size/2; index++,back--){

```



```

    void *dex = (char*)vec->array + index * vec->data_size;
    void *ack = (char*)vec->array + back * vec->data_size;
    memcpy(temp,dex,vec->data_size);
    memcpy(dex,ack,vec->data_size);
    memcpy(ack,temp,vec->data_size);
}
free(temp);
}

```

### **vector\_use.c**

```

#include<stdio.h>
#include<stdlib.h>
#include "vector.h"

int main(){
    vector *vec=vector_create(sizeof(float));
    vector *swp=vector_create(sizeof(float));
    float val;
    size_t position;
    short choice;
    while(1){
        printf("\n1.Push\n2.Pop\n3.Insert\n4.Remove\n5.Size\n6.Swap\n7.Is_present\n8.Reverse\n9.See\n10.Exit");
        printf("\nEnter your choice:");
        scanf("%d",&choice);
        switch (choice){
            case 1:
                printf("\nEnter value to push:");
                scanf("%f",&val);
                vector_push_back(vec,&val);

```

```
vector_push_back(swp,&val);
```

```
break;
```

```
case 2:
```

```
if(!vector_isempty(vec)){
```

```
    void *temp=vector_pop_back(vec);
```

```
    printf("\nPopped value is %f",*(float*)temp);
```

```
    free(temp);
```

```
}
```

```
else{
```

```
    printf("\nVector is empty...");
```

```
}
```

```
break;
```

```
case 3:
```

```
printf("\nEnter value to insert:");
```

```
scanf("%f",&val);
```

```
printf("\nEnter position for insert:");
```

```
scanf("%d",&position);
```

```
vector_insert(vec,&val,position);
```

```
break;
```

```
case 4:
```

```
printf("\nEnter position for remove:");
```

```
scanf("%d",&position);
```

```
vector_remove(vec,position);
```

```
break;
```

```
case 5:
```

```
printf("\nSize of vector is %d",vector_size(vec));
```

```
break;
```

```
case 6:
```

```
vector_swap(vec,swp);
```

```
break;
```

```
case 7:
```

```

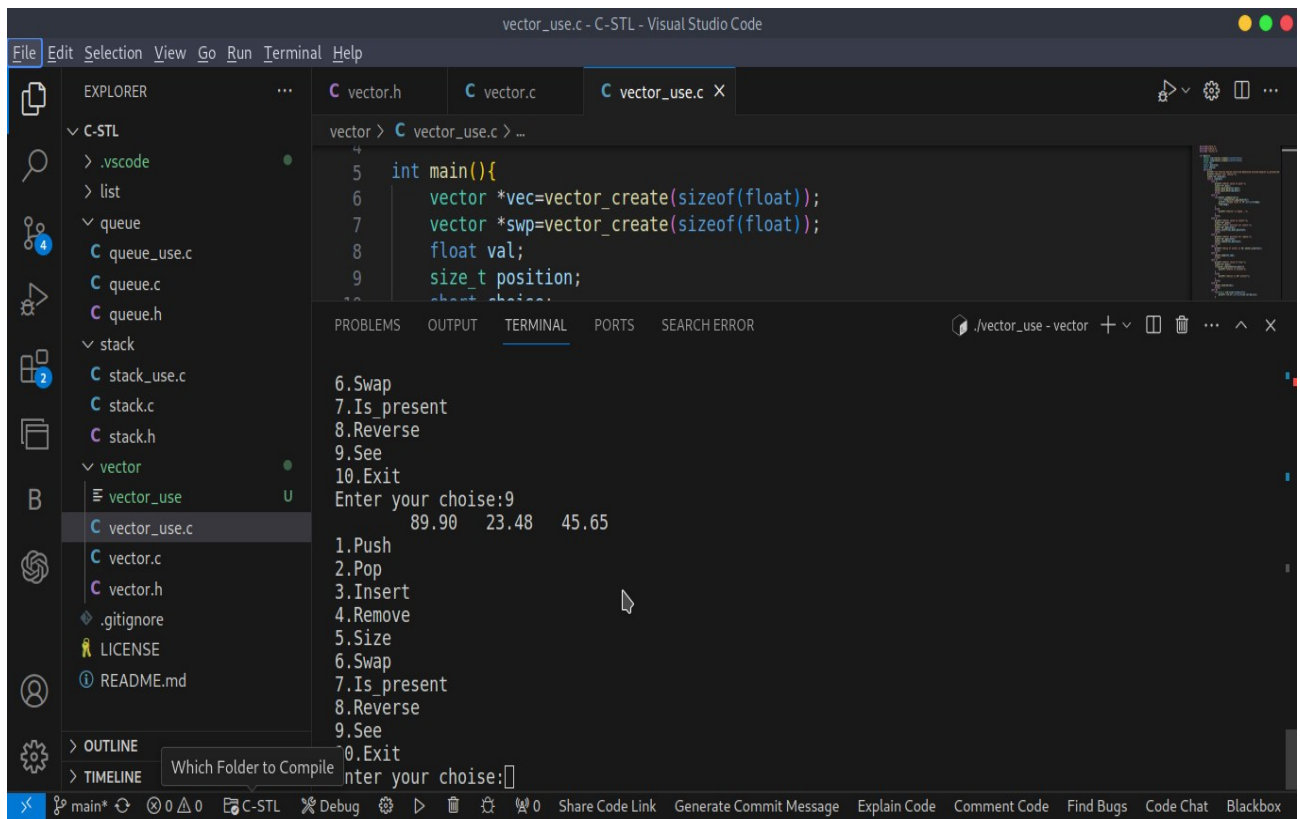
        printf("\nEnter value to find:");
        scanf("%f",&val);
        if(vector_ispresent(vec,&val)){
            printf("\nValue is present");
        }
        else{
            printf("\nValue is NOT present");
        }
        break;
case 8:
    vector_reverse(vec);
    break;
case 9:
    for(size_t i=0;i<vec->size;i++){
        printf("\t%.2f",((float*)vec->array)[i]);
    }
    break;
case 10:
    vector_clean(vec);
    vector_clean(swp);
    break;
default :
    printf("\nInvalid input");
}
if(choise==10)break;
}
return 0;
}

```

**Command to compile :** \$ gcc vector\_use.c vector.c -o vector\_use

**Command to run :** \$ ./vector\_use

## Output :



```
vector > C vector_use.c > ...
4
5 int main(){
6     vector *vec=vector_create(sizeof(float));
7     vector *swp=vector_create(sizeof(float));
8     float val;
9     size_t position;
10    char *choice;

PROBLEMS OUTPUT TERMINAL PORTS SEARCH ERROR
./vector_use - vector + - - - - -

6.Swap
7.Is present
8.Reverse
9.See
10.Exit
Enter your choice:9
89.90 23.48 45.65

1.Push
2.Pop
3.Insert
4.Remove
5.Size
6.Swap
7.Is present
8.Reverse
9.See
10.Exit
Enter your choice:
```

## 2. stack ->

### stack.h

```
#ifndef stack_h
#define stack_h

struct stack_node{
    void* data;
    struct stack_node *prev;
};

typedef struct{
    struct stack_node *top;
    size_t size;
    char *type;
}stack;

stack* stack_create(char*);

void stack_push(stack*,void*);
void* stack_pop(stack*);
void* stack_peek(stack*);
size_t stack_size(stack*);
void stack_swap(stack*,stack*);
short stack_ispresent(stack*,void*);
short stack_isempty(stack*);
void stack_clean(stack*);

#endif
```

## *stack.c*

```
#include<stdlib.h>
```

```
#include<string.h>
```

```
#include<ctype.h>
```

```
#include "stack.h"
```

```
stack* stack_create(char *data_type){  
    stack *st=(stack*)malloc(sizeof(stack));  
    st->top=NULL;  
    st->size=0;  
    st->type=(char*)malloc(strlen(data_type)+1);  
    strcpy(st->type,data_type);  
    for(int i=0;data_type[i]!='\0';i++){  
        st->type[i]=toupper(st->type[i]);  
    }  
    return st;  
}
```

```
void stack_push(stack *st,void* val){  
    struct stack_node* node=(struct stack_node*)malloc(sizeof(struct stack_node));  
    size_t size;  
    if(0==strcmp(st->type,"CHAR"))size=sizeof(char);  
    else if(0==strcmp(st->type,"STRING"))size=strlen((char*)val)+1;  
    else if(0==strcmp(st->type,"FLOAT"))size=sizeof(float);  
    else if(0==strcmp(st->type,"DOUBLE"))size=sizeof(double);  
    else if(0==strcmp(st->type,"SHORT"))size=sizeof(short);  
    else if(0==strcmp(st->type,"LONG"))size=sizeof(long);  
    else if(0==strcmp(st->type,"BOOL"))size=sizeof(short);  
    else size=sizeof(int);  
    node->data=malloc(size);
```

```
    memcpy(node->data, val, size);
    node->prev = st->top;
    st->top = node;
    st->size++;
}
```

```
void* stack_pop(stack *st){
    if(st->top == NULL) return NULL;
    void* val = st->top->data;
    struct stack_node *t = st->top;
    st->top = st->top->prev;
    free(t);
    st->size--;
    return val;
}
```

```
void* stack_peek(stack *st){
    if(st->top == NULL) return NULL;
    void* val = st->top->data;
    return val;
}
```

```
size_t stack_size(stack* st){
    size_t size = 0;
    struct stack_node *node = st->top;
    while(node != NULL){
        size++;
        node = node->prev;
    }
    st->size = size;
    return size;
}
```

```

void stack_swap(stack* st1,stack* st2){
    if(strcmp(st1->type,st2->type)==0){
        struct stack_node *temp=st1->top;
        st1->top=st2->top;
        st2->top=temp;

        size_t size=st1->size;
        st1->size=st2->size;
        st2->size=size;
    }
}

```

```

short stack_ispresent(stack* st,void* val){
    struct stack_node *node=st->top;
    while(node!=NULL){
        if(0==strcmp(st->type,"CHAR")){
            if(*(char*)node->data == *(char*)val)return 1;
        }
        else if(0==strcmp(st->type,"STRING")){
            if(strcmp((char*)node->data ,(char*)val)==0)return 1;
        }
        else if(0==strcmp(st->type,"FLOAT")){
            if(*(float*)node->data == *(float*)val)return 1;
        }
        else if(0==strcmp(st->type,"DOUBLE")){
            if(*(double*)node->data == *(double*)val)return 1;
        }
        else if(0==strcmp(st->type,"SHORT")){
            if(*(short*)node->data == *(short*)val)return 1;
        }
    }
}

```



```

else if(0==strcmp(st->type,"LONG")){
    if(*(long*)node->data == *(long*)val)return 1;
}
else if(0==strcmp(st->type,"BOOL")){
    if(*(short*)node->data == *(short*)val)return 1;
}
else{
    if(*(int*)node->data == *(int*)val)return 1;
}
node=node->prev;
}
return 0;
}

```

```

short stack_isempty(stack* st){
    if(st->top==NULL)return 1;
    return 0;
}

```

```

void stack_clean(stack* st){
    while(st->top!=NULL){
        struct stack_node *temp=st->top;
        st->top=temp->prev;
        free(temp->data);
        free(temp);
    }
    free(st->type);
    free(st);
}

```

## *stack\_use.c*

```
#include<stdio.h>

#include<stdlib.h>

#include "stack.h"

int main(){

    stack *st=stack_create("string");

    stack *s=stack_create("string");

    char val[20];

    short choise;

    struct stack_node *temp;

    while(1){

        printf("\n1.Push\n2.Pop\n3.Peek\n4.Size\n5.Swap\n6.Is_present\n7.See\n8.Exit");

        printf("\nEnter your choise:");

        scanf("%d",&choise);

        switch (choise){

            case 1:

                printf("\nEnter value to push:");

                scanf("%s",val);

                stack_push(s,val);

                stack_push(st,val);

                break;

            case 2:

                if(!stack_isempty(st)){

                    printf("\nPoped value is %s",((char*)stack_pop(st)));

                }

                else{

                    printf("\nstack is empty...");

                }

            }
```

```
        break;
case 3:
    if(!stack_isempty(st)){
        printf("\nvalue at peek is %s",((char*)stack_peek(st)));
    }
    else{
        printf("\nstack is empty...");
    }
    break;
case 4:
    printf("\nSize of stack is %d",stack_size(st));
    break;
case 5:
    stack_swap(st,s);
    break;
case 6:
    printf("\nEnter value to find:");
    scanf("%s",val);
    if(stack_ispresent(st,val)){
        printf("\nValue is present");
    }
    else{
        printf("\nValue is NOT present");
    }
    break;
case 7:
    temp=st->top;
    while(temp!=NULL){
        printf("\n%s",((char*)temp->data));
        temp=temp->prev;
    }
```

```

        break;
    case 8:
        stack_clean(st);
        stack_clean(s);
        break;
    default :
        printf("\nInvalid input");
    }
    if(choise==8)break;
}
return 0;
}

```

## **Output :**

The screenshot shows a Visual Studio Code window with the following components:

- EXPLORER:** A file tree on the left showing a project structure with folders like `C-STL`, `queue`, `stack`, and `vector`. The `stack` folder is expanded, showing files `stack_use.c`, `stack.c`, and `stack.h`.
- EDITOR:** The `stack_use.c` file is open, showing the `main()` function. Lines 5 through 10 are visible, containing the `int main()` function signature and the start of the program logic, including `stack_create` and `char val[20]`.
- TERMINAL:** The terminal at the bottom shows the output of the program. It lists menu options: `4.Size`, `5.Swap`, `6.Is_present`, `7.See`, `8.Exit`. It then prompts `Enter your choise:7` and shows the user input `7`. Below that, it shows the user input `4` and the prompt `Enter your choise:4`.

### 3. queue ->

#### queue.h

```
#ifndef queue_h
#define queue_h

struct queue_node{
    void *data;
    struct queue_node *next;
};

typedef struct{
    struct queue_node *front,*rear;
    size_t size;
    char *type;
}queue;

queue* queue_create(char* data_type);

void queue_push(queue*, void*);
void* queue_pop(queue*);

size_t queue_size(queue*);
void queue_swap(queue*,queue*);
short queue_ispresent(queue*,void*);
short queue_isempty(queue*);
void queue_clean(queue*);
void queue_reverse(queue*);
void* queue_peek(queue*);

#endif
```

## queue.c

```
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
#include "queue.h"
```

```
queue* queue_create(char* data_type){
    queue *q=NULL;
    while(q==NULL)q=(queue*)malloc(sizeof(queue));
    q->front=NULL;
    q->rear=NULL;
    q->size=0;
    q->type=(char*)malloc(strlen(data_type)+1);
    strcpy(q->type,data_type);
    for(int i=0;q->type[i]!='\0';i++){
        q->type[i]=toupper(q->type[i]);
    }
    return q;
}
```

```
void queue_push(queue* q, void* val){
    struct queue_node* node=NULL;
    node=(struct queue_node*)malloc(sizeof(struct queue_node));
    size_t size;
    if(0==strcmp(q->type,"CHAR"))size=sizeof(char);
    else if(0==strcmp(q->type,"STRING"))size=strlen((char*)val)+1;
    else if(0==strcmp(q->type,"FLOAT"))size=sizeof(float);
    else if(0==strcmp(q->type,"DOUBLE"))size=sizeof(double);
    else if(0==strcmp(q->type,"SHORT"))size=sizeof(short);
```

```

else if(0==strcmp(q->type,"LONG"))size=sizeof(long);
else if(0==strcmp(q->type,"BOOL"))size=sizeof(short);
else size=sizeof(int);
node->data=NULL;
node->data=malloc(size);
memcpy(node->data,val,size);
node->next=NULL;
if(q->rear==NULL){
    q->rear=node;
    q->front=node;
}
else{
    q->rear->next=node;
    q->rear=node;
}
q->size++;
}

```

```

void* queue_pop(queue* q){
    if(q->front==NULL)return NULL;
    void *val=q->front->data;
    struct queue_node *t=q->front;
    if(q->front==q->rear){
        q->front=NULL;
        q->rear=NULL;
    }
    else{
        q->front=q->front->next;
    }
    free(t);
    q->size--;
}

```

```
    return val;
}
```

```
size_t queue_size(queue* q){
    size_t size=0;
    struct queue_node *temp=q->front;
    while(temp!=NULL){
        size++;
        temp=temp->next;
    }
    q->size=size;
    return size;
}
```

```
void queue_swap(queue* q1,queue* q2){
    if (0==strcmp(q1->type,q2->type)) {
        struct queue_node *node=q1->front;
        q1->front=q2->front;
        q2->front=node;

        node=q1->rear;
        q1->rear=q2->rear;
        q2->rear=node;

        size_t size=q1->size;
        q1->size=q2->size;
        q2->size=size;

    }
}
```



```

short queue_ispresent(queue* q,void* val){
    struct queue_node *node=q->front;
    while(node!=NULL){
        if(0==strcmp(q->type,"CHAR")){
            if(*(char*)node->data == *(char*)val)return 1;
        }
        else if(0==strcmp(q->type,"STRING")){
            if(strcmp((char*)node->data ,(char*)val)==0)return 1;
        }
        else if(0==strcmp(q->type,"FLOAT")){
            if(*(float*)node->data == *(float*)val)return 1;
        }
        else if(0==strcmp(q->type,"DOUBLE")){
            if(*(double*)node->data == *(double*)val)return 1;
        }
        else if(0==strcmp(q->type,"BOOL")){
            if(*(short*)node->data == *(short*)val)return 1;
        }
        else{
            if(*(int*)node->data == *(int*)val)return 1;
        }
        node=node->next;
    }
    return 0;
}

```

```

short queue_isempty(queue* q){
    if(q->front==NULL)return 1;
    return 0;
}

```

```
void queue_clean(queue* q){
    while(q->front!=NULL){
        struct queue_node *temp=q->front;
        q->front=temp->next;
        free(temp->data);
        free(temp);
    }
    free(q->type);
    free(q);
}
```

```
void queue_reverse(queue* q) {
    struct queue_node* prev = NULL;
    struct queue_node* current = q->front;
    while (current != NULL) {
        struct queue_node* next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    struct queue_node* temp = q->front;
    q->front = q->rear;
    q->rear = temp;
}
```

```
void* queue_peek(queue* q) {
    if (q->front == NULL) {
        return NULL;
    }
    void* val = q->front->data;
    return val;
}
```

```
}
```

## **queue\_use.c**

```
#include<stdio.h>
```

```
#include<stdlib.h>
```

```
#include "queue.h"
```

```
int main(){
```

```
    queue *qu=queue_create("string");
```

```
    queue *q=queue_create("string"),*t;
```

```
    char val[20];
```

```
    short choice;
```

```
    struct queue_node *temp;
```

```
    while(1){
```

```
        printf("\n1.Push\n2.Pop\n3.Size\n4.Swap\n5.Is_present\n6.Reverse\n7.See\n8.Peek\n9.Exit");
```

```
        printf("\nEnter your choice:");
```

```
        scanf("%d",&choice);
```

```
        switch (choice){
```

```
            case 1:
```

```
                printf("\nEnter value to push:");
```

```
                scanf("%s",val);
```

```
                queue_push(q,val);
```

```
                queue_push(qu,val);
```

```
                break;
```

```
            case 2:
```

```
                if(!queue_isempty(qu)){
```

```
                    printf("\nPopped value is %s",((char*)queue_pop(qu)));
```

```
                }
```

```
            else{
```

```
                printf("\nqueue is empty...");
```

```

    }
    break;
case 3:
    printf("\nSize of queue is %d",queue_size(qu));
    break;
case 4:
    queue_swap(qu,q);
    break;
case 5:
    printf("\nEnter value to find:");
    scanf("%s",val);
    if(queue_ispresent(qu,val)){
        printf("\nValue is present");
    }
    else{
        printf("\nValue is NOT present");
    }
    break;
case 6:
    queue_reverse(qu);
    break;
case 7:
    temp=qu->front;
    while(temp!=NULL){
        printf("<- %s ",((char*)temp->data));
        temp=temp->next;
    }
    break;
case 9:
    queue_clean(qu);
    queue_clean(q);

```

```

        break;
    case 8:
        if(!queue_isempty(qu)){
            printf("\nPeeked value is %s",((char*)queue_peek(qu)));
        }
        else{
            printf("\nqueue is empty...");
        }
        break;
    default :
        printf("\nInvalid input");
    }
    if(choise==9)break;
}
return 0;
}

```

## **Output :**

queue\_use.c - C-STL - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

- C-STL
  - .vscode
  - list
  - queue
    - queue\_use
    - queue\_use.c
    - queue.c
    - queue.h
  - stack
    - stack\_use
    - stack\_use.c
    - stack.c
    - stack.h
  - vector
    - vector\_use
    - vector\_use.c
    - vector.c
    - vector.h
  - .gitignore
  - LICENSE

queue > C queue\_use.c > ...

```

1 #include<stdio.h>
2 #include<stdlib.h>
3 #include "queue.h"
4
5 int main(){
6     queue *qu=queue_create("string");

```

PROBLEMS OUTPUT TERMINAL PORTS SEARCH ERROR

./queue\_use - queue + - - - - -

```

4.Swap
5.Is_present
6.Reverse
7.See
8.Peek
9.Exit
Enter your choice:7
<- Ram <- q <- Sita <- q <- Mata <- Mother
1.Push
2.Pop
3.Size
4.Swap
5.Is_present
6.Reverse
7.See
8.Peek
9.Exit
Enter your choice:

```

main\* C-STL Debug 0 Share Code Link Generate Commit Message Explain Code Comment Code Find Bugs Code Chat Blackbox

## 4. list ->

### *list.h*

```
#ifndef list_h
#define list_h

struct list_node{
    void* data;
    struct list_node *next;
};

typedef struct{
    struct list_node *head;
    size_t size;
    char *type;
}list;

list* list_create(char*);
struct list_node* create_list_node(list*,void*);
void list_push_front(list* , void*);
void* list_pop_front(list*);
void list_push_back(list* , void*);
void* list_pop_back(list*);
void list_insert(list*,void*,size_t);
void list_remove(list*,size_t);
size_t list_size(list*);
void list_swap(list*,list*);
short list_ispresent(list*,void*);
short list_isempty(list*);
void list_clean(list*);
void list_reverse(list*);
#endif
```

## *list.c*

```
#include<stdlib.h>
```

```
#include<string.h>
```

```
#include<ctype.h>
```

```
#include "list.h"
```

```
list* list_create(char* data_type){
```

```
    list *li=(list*)malloc(sizeof(list));
```

```
    li->head=NULL;
```

```
    li->size=0;
```

```
    li->type=(char*)malloc(sizeof(strlen(data_type)+1));
```

```
    strcpy(li->type,data_type);
```

```
    for(int i=0;data_type[i]!='\0';i++){
```

```
        li->type[i]=toupper(li->type[i]);
```

```
    }
```

```
    return li;
```

```
}
```

```
struct list_node* create_list_node(list* li,void* val){
```

```
    struct list_node *node=NULL;
```

```
    while(node==NULL)node=(struct list_node*)malloc(sizeof(struct list_node));
```

```
    size_t size;
```

```
    if(0==strcmp(li->type,"CHAR"))size=sizeof(char);
```

```
    else if(0==strcmp(li->type,"STRING"))size=strlen((char*)val)+1;
```

```
    else if(0==strcmp(li->type,"FLOAT"))size=sizeof(float);
```

```
    else if(0==strcmp(li->type,"DOUBLE"))size=sizeof(double);
```

```
    else if(0==strcmp(li->type,"SHORT"))size=sizeof(short);
```

```
    else if(0==strcmp(li->type,"LONG"))size=sizeof(long);
```

```
    else if(0==strcmp(li->type,"BOOL"))size=sizeof(short);
```

```
    else size=sizeof(int);
```

```
node->data=NULL;
while(node->data==NULL)node->data=malloc(size);
memcpy(node->data,val,size);
node->next=NULL;
return node;
}
```

```
void list_push_front(list* li, void* val){
    struct list_node *node=create_list_node(li,val);
    if(li->head==NULL){
        li->head=node;
    }
    else{
        node->next=li->head;
        li->head=node;
    }
    li->size++;
}
```

```
void* list_pop_front(list* li){
    struct list_node *temp=li->head;
    if(temp==NULL)return NULL;
    void* val=temp->data;
    li->head=temp->next;
    free(temp);
    li->size--;
    return val;
}
```



```
void list_push_back(list* li, void* val){
    struct list_node *node=create_list_node(li,val);
    if(li->head==NULL){
        li->head=node;
    }
    else{
        struct list_node *temp=li->head;
        while(temp->next!=NULL){
            temp=temp->next;
        }
        temp->next=node;
    }
}
```

```
void* list_pop_back(list* li){
    struct list_node *temp=li->head;
    void* val;
    if(li->head==NULL)return NULL;
    if(temp->next==NULL){
        li->head=NULL;
        val=temp->data;
        free(temp);
    }
    else{
        while(temp->next->next!=NULL){
            temp=temp->next;
        }
        val=temp->next->data;
        free(temp->next);
        temp->next=NULL;
    }
}
```

```
li->size--;  
return val;  
}
```

```
void list_insert(list* li,void* val,size_t position){  
    struct list_node* node = create_list_node(li, val);  
    if(position == 1 || li->head==NULL){  
        node->next = li->head;  
        li->head = node;  
    }  
    else{  
        struct list_node* temp = li->head;  
        for(size_t i = 2; i < position; i++){  
            temp = temp->next;  
        }  
        node->next = temp->next;  
        temp->next = node;  
    }  
    li->size++;  
}
```

```
void list_remove(list* li,size_t position){  
    if(li->head == NULL || position > li->size)return;  
    struct list_node* temp = li->head,*t;  
    if(position == 1 ){  
        li->head=temp->next;  
        free(temp);  
    }  
    else{  
        for(size_t i=2;i<position;i++){  
            temp=temp->next;  
        }  
        t=temp->next;  
        temp->next=t->next;  
        free(t);  
    }  
    li->size--;  
}
```

```

    }
    t=temp->next;
    temp->next=t->next;
    free(t);
}
li->size--;
}

```

```

size_t list_size(list* li){
    size_t size=0;
    struct list_node *temp=li->head;
    while(temp!=NULL){
        temp=temp->next;
        size++;
    }
    li->size=size;
    return size;
}

```

```

void list_swap(list* li1, list* li2){
    if(strcmp(li1->type,li2->type)==0){
        size_t size=sizeof(list);
        list *t=(list*)malloc(size);
        memcpy(t,li1,size);
        memcpy(li1,li2,size);
        memcpy(li2,t,size);
        free(t);
    }
}

```

```

short list_ispresent(list* li,void* val){
    struct list_node *node=li->head;
    while(node!=NULL){
        if(0==strcmp(li->type,"CHAR")){
            if(*(char*)node->data == *(char*)val)return 1;
        }
        else if(0==strcmp(li->type,"STRING")){
            if(strcmp((char*)node->data ,(char*)val)==0)return 1;
        }
        else if(0==strcmp(li->type,"FLOAT")){
            if(*(float*)node->data == *(float*)val)return 1;
        }
        else if(0==strcmp(li->type,"DOUBLE")){
            if(*(double*)node->data == *(double*)val)return 1;
        }
        else if(0==strcmp(li->type,"SHORT")){
            if(*(short*)node->data == *(short*)val)return 1;
        }
        else if(0==strcmp(li->type,"LONG")){
            if(*(long*)node->data == *(long*)val)return 1;
        }
        else if(0==strcmp(li->type,"BOOL")){
            if(*(short*)node->data == *(short*)val)return 1;
        }
        else{
            if(*(int*)node->data == *(int*)val)return 1;
        }
        node=node->next;
    }
    return 0;
}

```

```
short list_isempty(list* li){  
    if(li->head==NULL)return 1;  
    return 0;  
}
```

```
void list_clean(list* li){  
    while(li->head!=NULL){  
        struct list_node *temp=li->head;  
        li->head=li->head->next;  
        free(temp);  
    }  
    free(li);  
}
```

```
void list_reverse(list* li) {  
    struct list_node* prev = NULL;  
    struct list_node* temp = li->head;  
    while (temp != NULL) {  
        struct list_node* next = temp->next;  
        temp->next = prev;  
        prev = temp;  
        temp = next;  
    }  
    li->head = prev;  
}
```

## *list\_use.c*

```
#include<stdio.h>

#include<stdlib.h>

#include "list.h"

int main(){

    list *li=list_create("int");

    list *lt=list_create("int");

    int num;

    size_t pos;

    short choice;

    struct list_node *temp;

    while(1){

        printf("\n1.Push front\n2.Pop front\n3.Push back\n4.Pop back\n5.Insert\
n6.Remove\n7.Size\n8.Swap\n9.Isresent\n10.Reverse\n11.See\n12.Exit\n");

        printf("\nEnter your choice:");

        scanf("%d",&choice);

        switch (choice){

            case 1:

                printf("\nEnter value to push front:");

                scanf("%d",&num);

                list_push_front(li,&num);

                list_push_front(lt,&num);

                break;

            case 2:

                if(!list_isempty(li)){

                    printf("\nPoped value from front:%d",*((int*)list_pop_front(li)));

                }

                else{

                    printf("\nlist is empty....");

                }

            }

        }

    }
```

```
}
```

```
break;
```

```
case 3:
```

```
printf("\nEnter value to push back:");
```

```
scanf("%d",&num);
```

```
list_push_back(li,&num);
```

```
list_push_back(lt,&num);
```

```
break;
```

```
case 4:
```

```
if(!list_isempty(li)){
```

```
    printf("\nPopped value from front:%d",*((int*)list_pop_back(li)));
```

```
}
```

```
else{
```

```
    printf("\nlist is empty....");
```

```
}
```

```
break;
```

```
case 5:
```

```
printf("\nEnter value for insert:");
```

```
scanf("%d",&num);
```

```
printf("\nEnter position for insert:");
```

```
scanf("%d",&pos);
```

```
list_insert(li,pos,&num);
```

```
break;
```

```
case 6:
```

```
printf("\nEnter position for remove:");
```

```
scanf("%d",&pos);
```

```
if(!list_isempty(li)){
```

```
    list_remove(li,pos);
```

```
    printf("\nValue is removed");
```

```
}
```

```
else{
```

```
        printf("\nlist is empty....");
    }
    break;
case 7:
    printf("\nThe size of list : %d",list_size(li));
    break;
case 8:
    list_swap(li,lt);
    break;
case 9:
    printf("\nEnter the number is to find :");
    scanf("%d",&num);
    if(list_ispresent(li,&num)){
        printf("\nElement is present");
    }
    else{
        printf("\nElement is not present");
    }
    break;
case 10:
    list_reverse(li);
    break;
case 11:
    temp=li->head;
    while(temp!=NULL){
        printf("\t%d",*((int*)temp->data));
        temp=temp->next;
    }
    break;
case 12:
    list_clean(li);
```



```

        list_clean(lt);

        break;

    default :

        printf("\nInvalid input");

    }

    if(choise==12)break;

}

return 0;

}

```

## Output :

The screenshot displays the Visual Studio Code interface. The Explorer sidebar on the left shows a project structure with folders for C-STL, queue, stack, and vector, each containing a .c, .h, and .use file. The main editor shows the source code of list\_use.c with lines 53 to 57 highlighted. The terminal window at the bottom shows the program's output, including a menu of operations and the user's choice of 11.

```

list > C list_use.c > main()
5  int main()
53
54     printf("\nEnter value for insert:");
55     scanf("%d", &num);
56     printf("\nEnter position for insert:");
57     scanf("%d", &pos);
    list insert(li, &num, pos);

```

```

12.Exit

Enter your choise:11
          44545   3443   45   569

1.Push front
2.Pop front
3.Push back
4.Pop back
5.Insert
6.Remove
7.Size
8.Swap
9.Isresent
10.Reverse
11.See
12.Exit

Enter your choise:

```

# **BIBLIOGRAPHY**

For the complication and documentation of the project titled **“Generic Collection Library in C”**, we have referred the following books and websites:

- Books:

**"The C Programming Language" by Brian W. Kernighan and Dennis M. Ritchie:** This classic book provides a comprehensive overview of the C programming language, including details on pointers, memory management, and data structures.

**"Data Structures and Algorithms in C" by Michael T. Goodrich, Roberto Tamassia, and David M. Mount:** This book covers various data structures and algorithms in C, which can be valuable for understanding the implementation of generic collections.

**"C Programming Absolute Beginner's Guide" by Greg Perry and Dean Miller:** If you're new to C programming, this beginner-friendly book can help you grasp the fundamentals and build a solid foundation.

**Project Source Code link :** <https://github.com/DigvijayChavan6/C-STL>