Lexicographic Order

-Under the guidance of Dr. Manoj Sahni, Associate Professor, PDPU, Gandhinagar

Made By: -

1. Parth Shah
2. Vitrag Shah
3. Nisarg Soni
4. Darshit Vachhani
5. Mitesh Vaghela

Lexicographic Order Theory

An ordering for the Cartesian product $\times$ of any two sets A and B with order relations <A and <B, respectively, such that if (a_1, b_1) and (a_2, b_2) both belong to $A \times B$, then $(a_1, b_1) < (a_2, b_2)$ iff either:

1. $a_1 <_A a_2$, or

2. $a_1 = a_2$ and $b_1 <_B b_2$.

The lexicographic order can be readily extended to cartesian products of arbitrary length by recursively applying this definition, i.e., by observing that $A \times B \times C = A \times (B \times C)$.

When applied to permutations, lexicographic order is increasing numerical order (or equivalently, alphabetic order for lists of symbols). For example, the permutations of {1,2,3} in lexicographic order are 123, 132, 213, 231, 312, and 321.

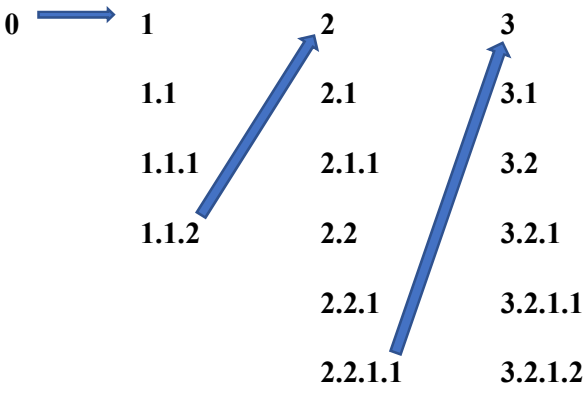
When applied to subsets, two subsets are ordered by their smallest elements. For example, the subsets of {1,2,3} in lexicographic order are {}, {1}, {1,2}, {1,2,3}, {1,3}, {2}, {2,3}, {3}.

Lexicographic order is sometimes called dictionary order.

Examples: -

1) **1, 2.2.1, 3.2, 2.2.1.1, 1.1.1, 0, 2.1, 3.2.1.1, 3, 3.1, 2.2, 2.1.1, 3.2.1, 1.1, 3.2.1.2, 2, 1.1.2**

Lexicographic Order: -



2) 1.2.2.1, 3.2.2, 3.2.1.1, 1.1, 1.2.1, 3, 0, 2, 1.2, 3.1, 2.1, 1, 3.1.1, 1.2.2, 3.2, 3.2.1

<u>Lexicographic Order</u> - 0, 1, 1.1, 1.2, 1.2.1, 1.2.2, 1.2.2.1, 2, 2.1, 3, 3.1, 3.1.1, 3.2, 3.2.1, 3.2.1.1, 3.2.2

Lexicographic Order C Program

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

struct node
{
	char arr[15];
	struct node *next;
};

void display(struct node *head)//to display the sorted list
{
	struct node *temp=head;
	printf("\n--------------------------------\n");
	if(head!=NULL)
	{
		while(temp->next!=NULL)
		{
			printf("\n->%s",temp->arr);
			temp=temp->next;
		}
		printf("\n->%s",temp->arr);
	}
	else
	{
		printf("\nList is empty\n");
	}
}

int convert(char *a)
{
	int i=0,j=0;
	while(a[i]!='.')
	{
		j=10*j+(a[i]-'0');
		i++;
	}
```

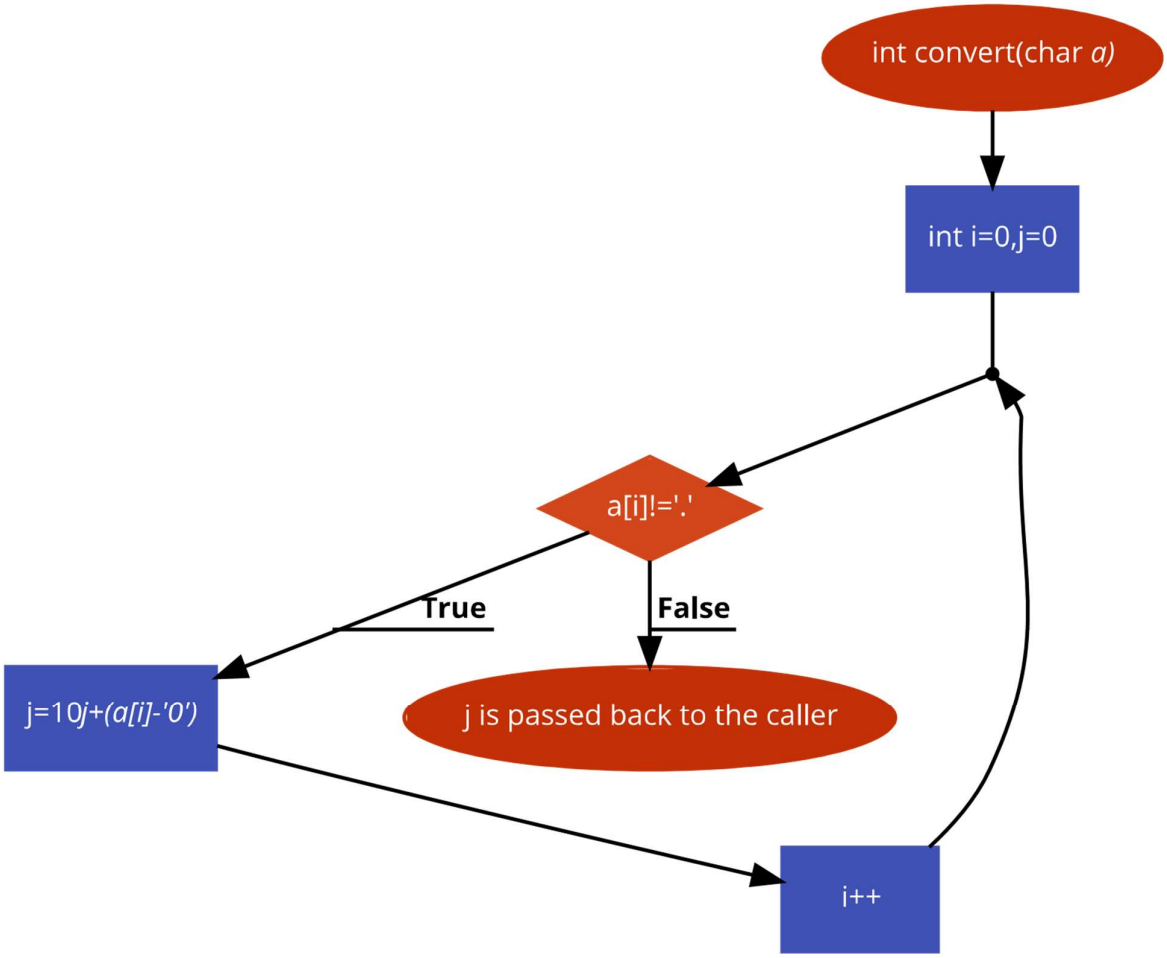
```
	return j;
}

int check(char *a,char *b)
{
	int i,j=0,temp=0;
	if(convert(a)>convert(b))
	{
		temp=1;
		j=1;
	}
	while(temp!=1)
	{
		if(a[i]==b[i]&&a[i]!='\0') //when two same things are compared
		{
			i++;			//we will update i to check next two things
			continue;		//loop will go
		}
		else if(a[i]=='\0'&&b[i]=='\0') //if two things will be same i.e 1.23 ,1.23
		{
			break;
		}
		else if(a[i]=='\0'&&b[i]!='\0')//when a is lesser than b i.e 1.11 ,1.111
		{
			break;
		}
		else if(b[i]=='\0'&&a[i]!='\0')//when b is lesser than a i.e 1.111,1.11
		{
			j=1;
			break;
		}
		else if((int)a[i]>(int)b[i])  //when we can say that we have to swap i.e 2.1 ,1.1
		{
			j=1;
			break;
		}
		else if((int)b[i]>(int)a[i])//when we do not have to swap i.e 1.1,1.2;
		{
			break;
		}
		else if((b[i]=='.')&&(a[i]!='.')&&(a[i]!='\0'))
		//when a's first digit is better than b's first digit 11.1 ,1.11
		{
			j=1;
			break;
		}
		else if((a[i]=='.')&&(b[i]!='.')&&(b[i]!='\0'))
		//when b's first digit is better than a's first digit 1.11 ,11.1
		{
			 break;
```

```
                }
                else//when we do not have to do anything except update the list
                {
                        i++;
                        continue;
                }
        }
        return j;
        //j is assigned zero that means b should come after a, if j is one that means b should come
        before a
}

struct node *insert(struct node *head,char *s)
{
        struct node *n;
        n=(struct node *)malloc(sizeof(struct node));
        strcpy(n->arr,s);
        if(head==NULL)
        {
                head=n;
                head->next=NULL;
        }
        else
        {
                if(check(head->arr,n->arr))//check function will return o or 1
                {
                        n->next=head;
                        head=n;
                }
                else
                {
                        head->next=insert(head->next,s);
                        //recursive call so next place will be compared
                }
        }
        return head;
}

void main()
{
        struct node *head;
        head=NULL;
        int i,j;
        char *a,b;
        while(243)
        {
                printf("\n--------------------------------\n1) To enter number\n2) Display list\n3)
                Exit\nEnter choice:");
                scanf("%d",&i);
                if(i==1)
```

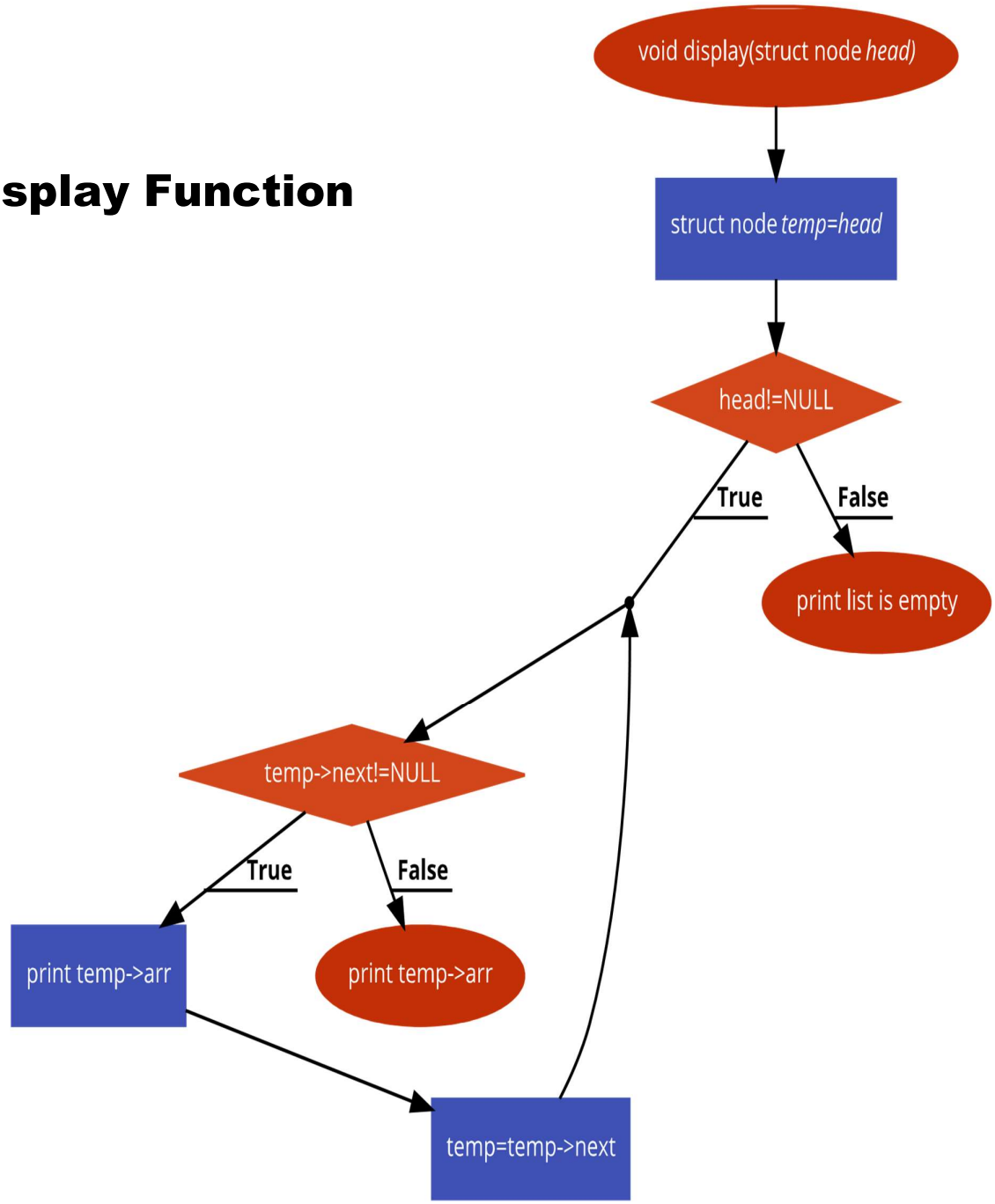
```
		{
			j=0;
			printf("\n--------------------------------\n");
			printf("\nEnter number in list:");
			a=(char *)malloc(sizeof(char)*15);
			while(1)
			{
				scanf("%c",&b);
				if(b=='\n'||b=='\0')
				{
					a[j]='\0';
					break;
				}
				else
				{
					a[j++]=b;
				}
			}
			gets(a);
			head=insert(head,a);
			free(a);
		}
		else if(i==2)
		{
			display(head);
		}
		else if(i==3)
		{
			printf("\nThank You\3");
			printf("\n--------------------------------\n");
			break;
		}
		else
		{
			printf("\nEnter valid choice\n");
		}
	}
}
```

Lexicographic Order Algorithm Flowchart

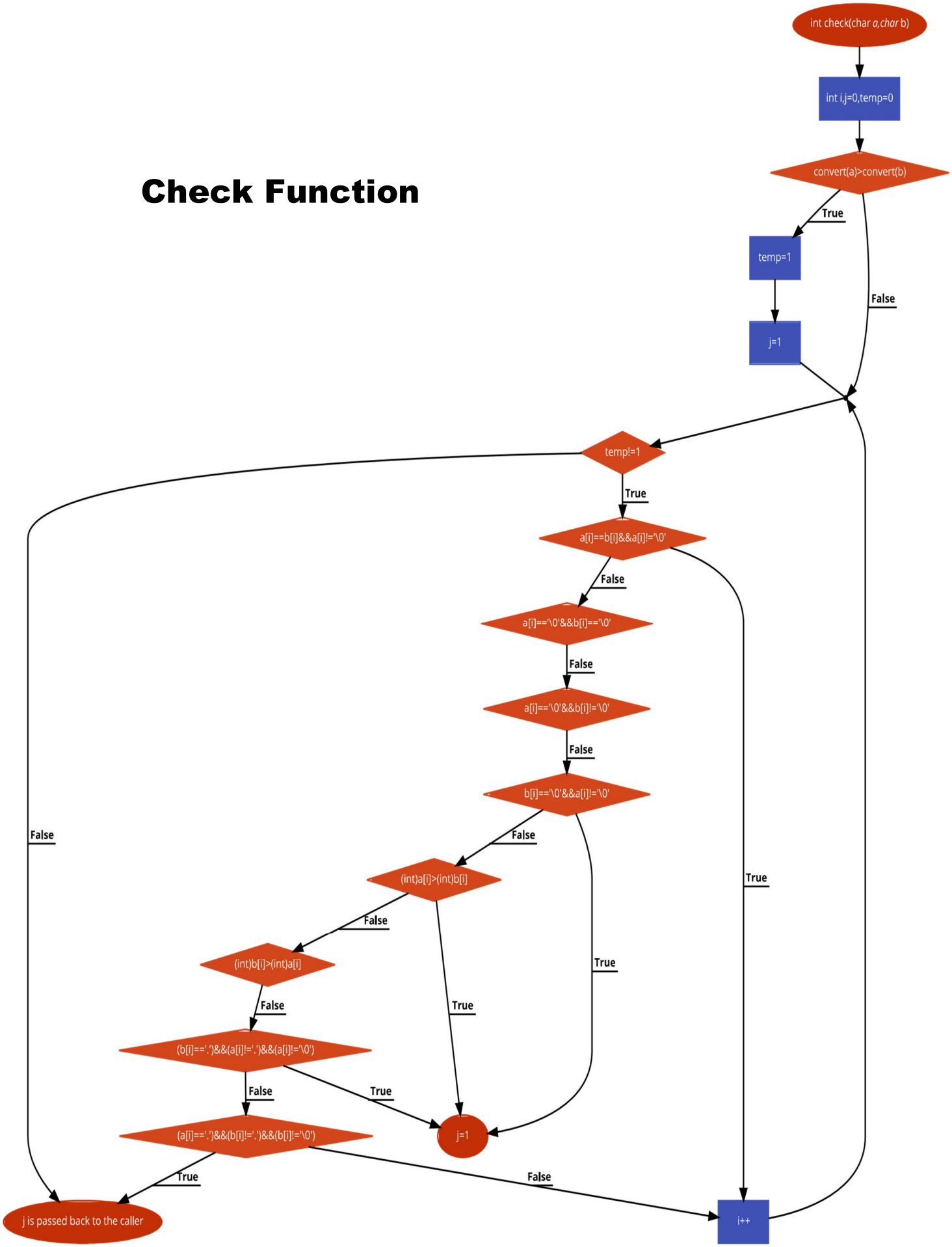
Convert Function



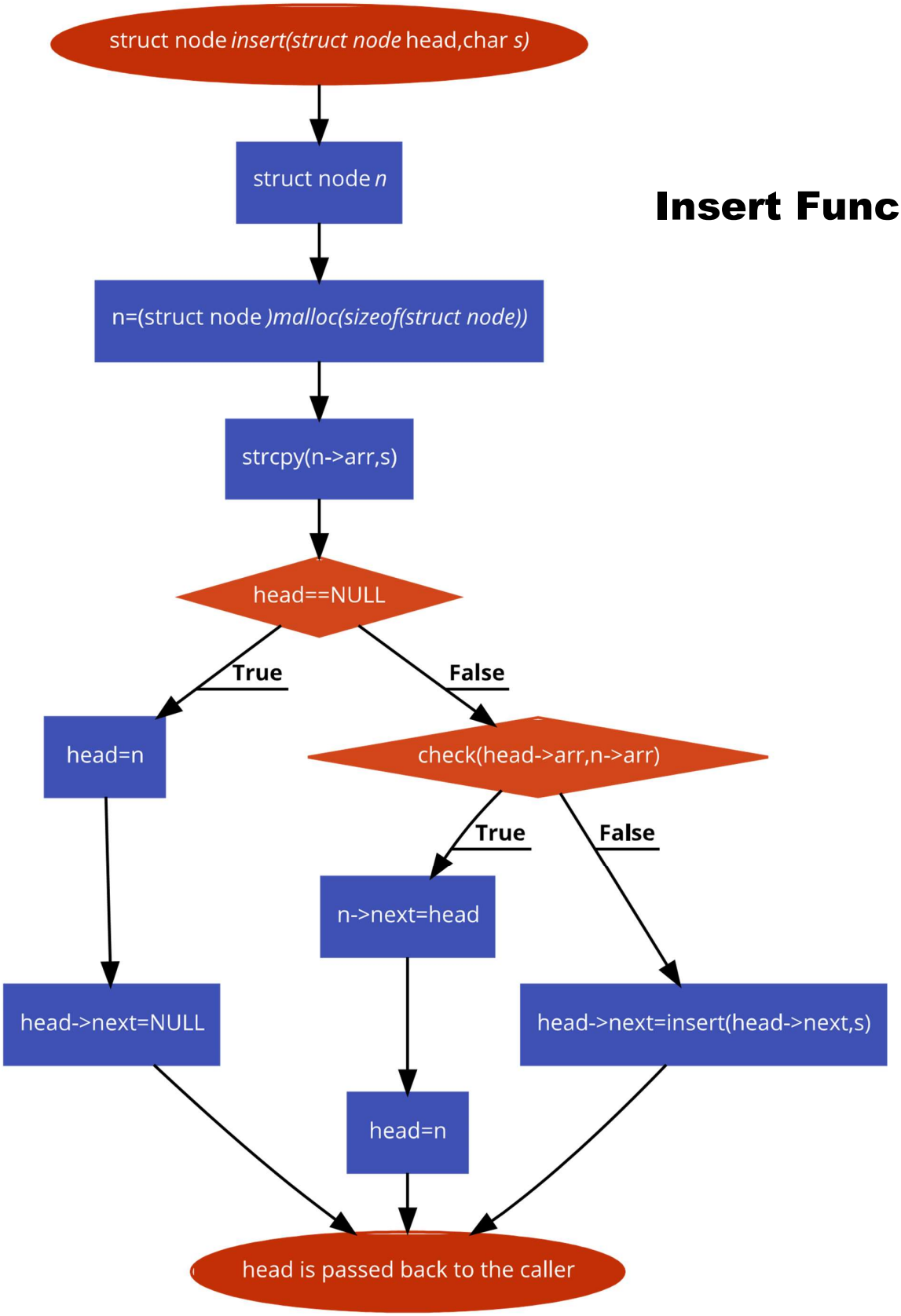
Display Function



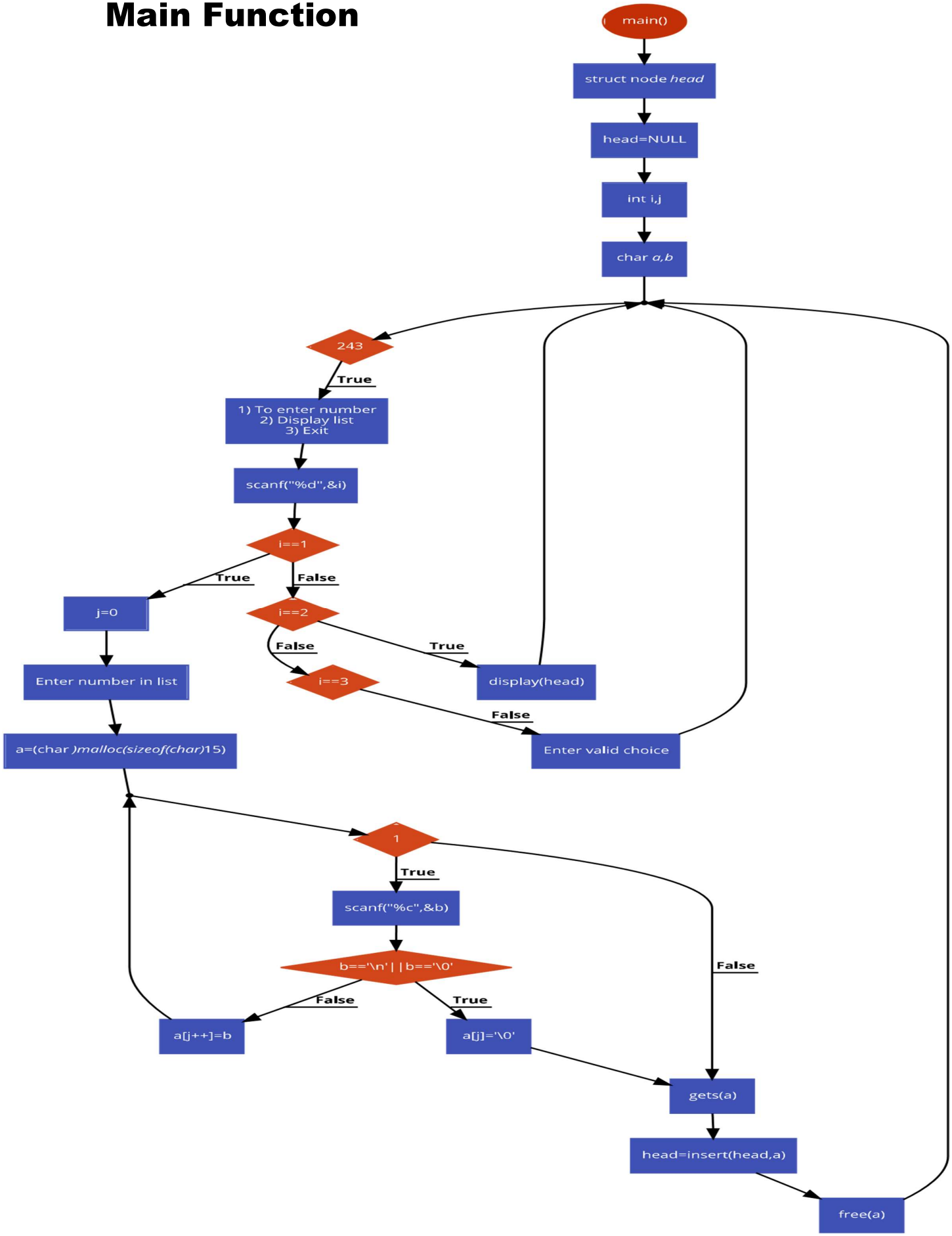
Check Function



Insert Function



Main Function



Applications: -

1. Lexicographic order is the complex activity concerned with the development of theories and principles for the design, compilation, use, and evaluation of dictionaries.
2. It is also used in note-making and segregation of information into sections, for various books.

Contribution: -

1) **Parth Shah (17BIT051) – Algorithm, Report**
2) **Vitrag Shah (17BIT052) – Code**
3) **Nisarg Soni (17BIT053) – PPT**
4) **Darshit Vachhani (17BIT054) – Code**
5) **Mitesh Vaghela (17BIT055) – PPT**