BST 2

```
//successor predessor
#include <stdio.h>
#include <stdlib.h>
struct node
{
	int data;
	struct node* right;
	struct node* left;
};
struct node* insert(struct node* root,int x)
{
	if(root == NULL)
	{
		struct node* temp = (node*)malloc(sizeof(struct node*));
		temp->data = x;
		temp->left = temp->right = NULL;
		root = temp;
		return root;
	}
	else if(x <= root->data)
		root->left = insert(root->left,x);
	else
		root->right = insert(root->right,x);
	return root;
}
int minimum(struct node* root)
{
	if(root->left == NULL)
		return root->data;
	return minimum(root->left);
}
int maximum(struct node* root)
{
	if(root->right == NULL)
		return root->data;
	return maximum(root->right);
}
struct node* find(struct node* root,int x)
{
	if(root == NULL)
		return NULL;
	if(root->data == x)
		return root;
	else if(x <= root->data)
		return find(root->left,x);
	else
		return find(root->right,x);
}
```

```
int successor(struct node* root,int x)
{
	struct node* curr = find(root,x);
	if(curr == NULL)return -1;
	//case 1.
	if(curr->right != NULL)
		return minimum(curr->right);
	//case 2.
	else
	{
		struct node* success = NULL;
		while(root != curr)
		{
			if(curr->data < root->data)
			{
				success = root;
				root = root->left;
			}
			else
				root = root->right;
		}
		if(success == NULL)
			return -1;
		return success->data;
	}
}
int predecessor(struct node* root,int x)
{
	struct node* curr = find(root,x);
	if(curr == NULL) return -1;
	if(curr->left != NULL)
		return maximum(curr->left);
	else
	{
		struct node* predec = NULL;
		while(root != curr)
		{
			if(curr->data > root->data)
			{
				predec = root;
				root = root->right;
			}
			else
				root = root->left;
		}
		if(predec == NULL)
			return -1;
		return predec->data;
	}
}
int main()
{
```

```
	int x,n;
	struct node* root = NULL;
	while(1)
	{
		printf("1 insert,2 minimum,3 maximum,4 successor,5 predecessor,6
end\n");
		scanf("%d",&n);
		switch(n)
		{
			case 1:
				scanf("%d",&x);
				root = insert(root,x);
				break;
			case 2:
				printf("minimum %d\n",minimum(root));
				break;
			case 3:
				printf("maximum %d\n",maximum(root));
				break;
			case 4:
				scanf("%d",&x);
				printf("successor %d\n",successor(root,x));
				break;
			case 5:
				scanf("%d",&x);
				printf("predecessor %d\n",predecessor(root,x));
				break;
			case 6:
				return 0;
		}
	}
}
```