```c
#define CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <conio.h>

#define MAX_STACK_SIZE 25u

void Push(unsigned char* Stack, unsigned char *SP, unsigned char ValueToAdd);
unsigned char Pop(unsigned char *Stack, unsigned char *SP);
unsigned char IsStackFull(unsigned char *SP);
unsigned char IsStackEmpty(unsigned char *SP);
void createStack(unsigned char *Stack, unsigned char *SP);
unsigned char ManaPnuelli(unsigned char Val);
//unsigned char MyStack[MAX_STACK_SIZE];


typedef struct pentru_stiva {
	unsigned char x;
	unsigned char y;
} tip_stiva;

tip_stiva MyStack[MAX_STACK_SIZE];


unsigned char StackPointer;

int main()
{
	unsigned char Val;
	/*Mana Pnuelli*/
	/*f(6)*/
	createStack(&MyStack[0u], &StackPointer);
	Val = ManaPnuelli(6);

	_getch();
	return 0;
}

void createStack(unsigned char *Stack, unsigned char *SP)
{
	unsigned char count;
	*SP = 0u;

	for (count = 0; count < MAX_STACK_SIZE; count++)
	{
		Stack[count] = 0x00;
	}

}

unsigned char IsStackEmpty(unsigned char *SP)
{
	return ((*(SP)) == 0X00u);
}

unsigned char IsStackFull(unsigned char *SP)
{
	return ((*SP) == (MAX_STACK_SIZE - 1u));
}
```

```
void Push(unsigned char* Stack, unsigned char *SP, unsigned char ValueToAdd)
{

	if(!IsStackFull(SP))
	{
		Stack[*SP] = ValueToAdd;
		*SP = *SP + 1u;
	}
	else
	{
		/*stack is full*/
	}

}

unsigned char Pop(unsigned char *Stack, unsigned char *SP)
{
	unsigned char ValToReturn = 0xFF;

	if (!IsStackEmpty(SP))
	{
		*SP = *SP - 1;
		ValToReturn = Stack[*SP];
		return ValToReturn;
	}
	else
	{
		return ValToReturn;
	}

}

unsigned char ManaPnuelli(unsigned char Val) //Val=6
{
	unsigned char ManaPnuelliStop = 0;
	unsigned char PopVal_1;  //prima val pe care o scoate
	unsigned char PopVal_2;
	Push(&MyStack[0], &StackPointer, Val); //pune valoarea 6 in stiva

	while (1) //bucla infinita / do - while(1) / for(;;) - no init value, no
condition, no implementation
	{
		PopVal_1 = Pop(&MyStack[0u], &StackPointer); //il scoate

		if (PopVal_1 < 12u) // si verfica daca 6 este mai  mic decat 12
		{
			Push(&MyStack[0], &StackPointer, PopVal_1); //il baga pe 6 inapoi
			Push(&MyStack[0], &StackPointer, PopVal_1 + 2u); //si il pune si pe
6+2 in stiva
		}

		else // ajungem la un moment dat la 13
		{
			if (IsStackEmpty(&StackPointer))
			{
				return PopVal_1 - 1u; // returnam x-1 //acesta este "break-ul"
din bucla infinita
```

```
                    }

                    else
                    {
                           /*another one shall be popped*/
                           PopVal_2 = Pop(&MyStack[0u], &StackPointer);  //il scoate pe
13 / scoatem valoarea la care am ajuns
                           Push(&MyStack[0], &StackPointer, PopVal_1 - 1u); //pune inapoi
13-1 / punem inapoi valoarea - 1
                    }
                    /*popAlso to check if it is the last element*/
             }
      }
}

unsigned char Ackerman(tip_stiva Val) // unde val este perechea (x,y) (val. pt x, y sunt
date in main)
{
      tip_stiva PopVal_1;
      tip_stiva PopVal_2;


      Push(&MyStack[0], &StackPointer, Val); //pune perechea (x,y) in stiva


      while (1)
      {
             PopVal_1 = Pop(&MyStack[0u], &StackPointer); //scoate pereche (x,y)
introdusa in stiva anterior


             if ( PopVal_1.x == 0) // si verifica daca
             {
                    return PopVal_1.y + 1u; // returneaza y+1
             }

             else if (PopVal_1.y == 0)
             {
                    tip_stiva aux;
                    aux.x = PopVal_1.x - 1u; // x -> x-1
                    aux.y = 1; // y -> 1

                    Push(&MyStack[0], &StackPointer, aux); //perechea (x,y) se
transforma in (x-1,1)
             }

             PopVal_2 = Pop(&MyStack[0u], &StackPointer); //scoate ultima pereche

             if(PopVal_2.x !=0 && PopVal_2.x !=0) //verfica daca in ultima pereche
scoasa x-ul si y-ul sunt !=0
             {
                    tip_stiva aux2;
                    aux2.x = x;
                    aux2.y = y - 1;
                    Push(&MyStack[0], &StackPointer, aux2); // transforma (x,y) -> (x,y-
1)
             }
```

```
        }
}
```