Programming Homework 2

(1)

/\*\*

 \* file phw2.c

 \*

 \* author : Jeon Sihyeon (wjstlgus13@gachon.ac.kr)

 \*

 \* Date : 03/20/2022

 \* Partnet : I worked alone

 \* Course : Data Structures(14461\_004)

 \*

 \* Summary of File :

 \*      This file contains code which reads a decimal number, and using a stack,

 \*      converts it to its binary equivalent, and prints it.

 \*

\*\*/

#include <stdio.h>

#include <stdlib.h>

struct q{

    int num1;       // 0 or 1

    struct q \*next;

}stack[40];

int push (int givenNum, struct q head);

int dec2bi(struct q head, int num);

void printReversedStack(struct q \*ptr);

void clearStack(struct q \*ptr);     //function declaration

int main(){

    for (int i = 0; i<39; i++){

        stack[i].next = &stack[i+1];

        stack[i].num1 = -1;

    }

    stack[39].num1 = -1;

   struct q head;

   head.next = &stack[0];   //head of stacks

   stack[39].next = NULL;

    int num[5];

    for (int i = 0; i<5; i++){

        scanf("%d", &num[i]);

        if(num[i]<=0){

            printf("input a positive number\n");

            i--;

            continue;

        }

        if(dec2bi(head, num[i])==0){

            printf("failed to convert -- too big number\n");

            i--;

        }

        else{

            struct q \*ptr = head.next;

            printReversedStack(ptr);    //calls printReversedStack function

            printf("\n");

        }

        struct q \*ptr = head.next;

        clearStack(ptr);                //calls clearStack function

    }

    return 0;

}

/\*\*

 \* int push (int givenNum, struct q head)

 \*

 \* Summary of the push :

 \*      The push function pushes 0 or 1

 \*

 \* Parameters   : integer number

 \*                  struct q

 \*

 \* Return Value : integer

 \*                  return 0 : fail to push the number

 \*                  return 1 : success to push the number

 \*

 \* Description:

 \*

 \*

 \*      This function utilizes linked list.

 \*

 \*/

int push (int givenNum, struct q head){

    struct q \*ptr = head.next;

    while(ptr){

        if (ptr->num1 == -1){

            ptr->num1 = givenNum;

            break;

        }

        ptr = ptr->next;

    }

    if(ptr==NULL){

        return 0;   //failed to push -- the stack is full

    }

    return 1;   //success to push

}

/\*\*

 \* int dec2bi(struct q head, int num)

 \*

 \* Summary of the dec2bi :

 \*      The dec2bi function converts a decimal number to its binary equvalent

 \*      by calling push function

 \*

 \* Parameters   : integer number

 \*                  struct q

 \*

 \* Return Value : integer

 \*                  return 0 : fail to convert

 \*                  return 1 : success to convert the number

 \*

 \* Description:

 \*

 \*

 \*      This function utilizes converting decimal into binary equivalent algorithm.

 \*

 \*/

int dec2bi(struct q head, int num){

    int bin;

    int push\_res;

    printf("input number : %d\n", num);

    while(num>0){

        bin = num%2;        // 0 or 1

        num/=2;

        push\_res = push(bin, head); //call push function to push 0 or 1

        if (push\_res==0){   // failed   (the stack is full)

            return 0;       // failed

        }

    }

    return 1;   //successfully converted a number to its binary equivalent

}

/\*\*

 \* void printReversedStack(struct q \*ptr)

 \*

 \* Summary of the printReversedStack :

 \*      print stack from the end(which has 0 or 1) to the front

 \*

 \* Parameters   : struct's pointer

 \*

 \* Return Value : nothing

 \*

 \* Description:

 \*

 \*      This function utilizes recursive sequence

 \*

 \*/

void printReversedStack(struct q \*ptr){

    if (ptr->num1 == -1){

        return;         //exit condition

    }

   printReversedStack(ptr->next);

   printf("%d", ptr->num1);

}

/\*\*

 \* void clearStack(struct q \*ptr)

 \*

 \* Summary of the push :

 \*      The clearStack function changes all the num1 in stacks into -1

 \*

 \* Parameters   : struct's pointer

 \*

 \* Return Value : nothing

 \*

 \*/

void clearStack(struct q \*ptr){

    while(ptr){

        ptr -> num1 = -1;   //clear the ptr

        ptr = ptr -> next;

    }

}

텍스트이(가) 표시된 사진

자동 생성된 설명

(2)

/\*\*

 \* file phw2-2.c

 \*

 \* author : Jeon Sihyeon (wjstlgus13@gachon.ac.kr)

 \*

 \* Date : 03/20/2022

 \* Partnet : I worked alone

 \* Course : Data Structures(14461\_004)

 \*

 \* Summary of File :

 \*      This file contains code that reads information about a general tree

 \*       and constructs a binary tree, using the leftmost-child-right-siblings

 \*      representation. Then traverse the binary tree and print the keys of the nodes

 \*      visited in preorder.

\*\*/

#include <stdio.h>

struct node{

    int treeLevel;

    int key;

    int child\_key

};

struct data{

    int num;

    struct data \*leftChildPTR;

    struct data \*rightChildPTR;

};

void LeftmostChildRightSiblings(struct node nodes[], struct data linkedNodes[], int size);

void preorder(struct data \*ptr);    //function declarations

int main(){

    struct node a[200] = {{1, 100, 200}, {2, 200, NULL}, {2, 75, 25}, {2, 300, NULL},

    {3, 25, NULL}, {3, 50, NULL}, {3, 30, 120}, {3, 150, NULL}, {4, 120, NULL}, {4, 55, NULL}}; //tree 1

    int size1 = 10;

    struct data nodeData[200];

    printf("Tree 1 : ");

    LeftmostChildRightSiblings(a, nodeData, size1);    //calls the function LeftmostChildRightSiblings

    struct node b[200] = {{1, 100, 200}, {2, 200, NULL}, {2, 75, 25}, {2, 300, NULL}, {2, 95, 150},

                        {3, 25, NULL}, {3, 50, NULL}, {3, 30, 120}, {3, 150, NULL}, {4, 120, NULL}};    //tree 2

    int size2 = 10;

    printf("\nTree 2 : ");

    struct data nodeData2[200];

    LeftmostChildRightSiblings(b, nodeData2, size2);        //calls the function LeftmostChildRightSiblings

    return 0;

}

/\*\*

 \* void LeftmostChildRightSiblings(struct node nodes[], struct data linkedNodes[], int size)

 \*

 \* Summary of the LeftmostChildRightSiblings :

 \*      The LeftmostChildRightSiblings function reads information about a general tree and

 \*      constructs a binary tree.

 \*

 \* Parameters   : struct array, integer number

 \*

 \* Return Value : nothing

 \*

 \* Description :

 \*

 \*

 \*      This function utilizes Leftmost Child-Right Siblings.

 \*

 \*/

void LeftmostChildRightSiblings(struct node nodes[], struct data linkedNodes[], int size){

    for (int i = 0; i<size; i++){

        linkedNodes[i].num = nodes[i].key;

    }

    for (int i = 0; i<size; i++){

        linkedNodes[i].leftChildPTR = NULL;

        linkedNodes[i].rightChildPTR = NULL;        //initializes certain leftchildPTR and rightChildPTR

    }

    for (int i = 0; i<size; i++){

        if (nodes[i].child\_key!=NULL){      //if nodes[i] has a child\_key

            for (int j = i; j<size; j++){

                if (nodes[i].child\_key==nodes[j].key && ((nodes[i].treeLevel)+1 == nodes[j].treeLevel)){

                    linkedNodes[i].leftChildPTR = &linkedNodes[j];      // left most child

                }

            }

        }

    }

    for (int i = 0; i<size-1; i++){

        if (nodes[i].treeLevel == nodes[i+1].treeLevel)

            linkedNodes[i].rightChildPTR = &linkedNodes[i+1];   //linkedNodes[i].rightChildPTR has linkedNodes[i+1]'s address

    }

    for (int i = 0; i<size-1; i++){

        for (int j = i+1; j<size; j++){

            if(linkedNodes[i].leftChildPTR == linkedNodes[j].rightChildPTR){

                linkedNodes[j].rightChildPTR = NULL;    //cut the connection

            }

        }

    }

    struct data \*ptr = &linkedNodes[0];     //ptr has linkedNodes[0]'s address

    preorder(ptr);  //calls preorder function

}

/\*\*

 \*

 \* void preorder(struct data \*ptr)

 \*

 \* Summary of the preorder :

 \*      The preorder function prints the keys of the nodes visited in preorder.

 \*

 \* Parameters   : struct pointer

 \*

 \* Return Value : nothing

 \*

 \* Description:

 \*

 \*      This function utilizes preorder traversal

 \*

 \*

 \*/

void preorder(struct data \*ptr){

    if (ptr){

        printf("%d ", ptr->num);

        preorder(ptr->leftChildPTR);

        preorder(ptr->rightChildPTR);

    }

}

