# <이산수학>_2장_프로그래밍 실습_C 코드

## 프로그래밍 실습 1

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

//가상으로 구현된 스택의 정보처리 의 한 단위를 의미하는 Node입니다.
struct Node
{
	int Value;
//합성명제 또는 피연산자의 진리값을 담는 변수입니다.
	int HeadArrive;
//Stack의 head가 비어있는지 비어있지 않은지를 판정하는 변수입니다.
	char Modify[200];
//합성명제 식 또는 피연산자, 연산자들이 들어있는 문자열 공간입니디.
	struct Node *Next;
//다음 노드를 가리키는 구조체 포인터입니다.
};
typedef struct Node Node;
Node* Push(Node *head,int value,char *modify)
{
	int i;
	Node *NewNode,*Temp = head;
	if(head->HeadArrive == 0)
	{
		head->Value = value;//비어있는 노드에 값을 대입합니다.
		head->Next = NULL;//최상단 노드이기때문에 Next는 Null입니다.
			head->HeadArrive = 1;
		i=0;

		while(modify[i]!='₩0')//합성명제를 복사합니다.
		{
			head->Modify[i] = modify[i];
			i++;
		}
		head->Modify[i] = '₩0';
		return head;//헤드값을 리턴합니다.
	}else
	{
		while(Temp->Next != NULL)//스택의 최상단까지 찾아갑니다.
	{
			Temp = Temp->Next;
		}
		NewNode = (Node*)malloc(sizeof(Node));//새로운 노드를 할당합니다.
		Temp->Next = NewNode;
		NewNode->Value = value;//값을 대입합니다.
		NewNode->Next = NULL;
		i=0;
		while(modify[i]!='₩0')//합성명제를 복사합니다.
		{
			NewNode->Modify[i] = modify[i];
			i++;
		}
		NewNode->Modify[i] = '₩0';
		return NewNode;
	}
}
```

```
//---------------------------------------------------------------------------
//Top노드를 팝시키고 Top Node 전 주소값을 리턴합니다.
Node* Pop(Node *head,char *copyModify)
{
	int i=0,j=0;//Index로 사용할 i,j를 선언합니다.
	Node *Temp=head,*Temp2;
	if(head->Next == NULL)//만약 노드가 한 개뿐일 때.
	{
		if(head->HeadArrive == 0)//만약 head가 비어있을 때.
		{
			printf("잘못된 Pop연산.₩n");
			return 0;
		}

		while(copyModify[j] !='₩0')
		{
			j++;
		}
		while(head->Modify[i] != '₩0')
		{
			copyModify[j+i] = head->Modify[i];
			i++;
		}
		copyModify[j+i] = '₩0';
		head->HeadArrive = 0;
		return head;//최상단값인 헤드를 리턴합니다.
	}else
	{
		while(Temp->Next->Next != NULL)
		{
			Temp = Temp->Next;
		}
		Temp2 = Temp;//Top노드의 전노드주소를 저장합니다.
		Temp = Temp->Next;//제거될 Top노드의 주소값을 저장합니다.

		while(copyModify[j] !='₩0')
		{
			j++;
		}
		while(Temp->Modify[i] != '₩0')
		{
			copyModify[j+i] = Temp->Modify[i];
			i++;
		}
		copyModify[j+i] = '₩0';
		Temp2->Next = NULL;
		free(Temp);//Top노드 제거
		return Temp2;//Top노드 전노드의 주소를 리턴합니다.
	}
}
//---------------------------------------------------------------------------
//이 함수는 명제에서 중복되지 않는 명제변수의 개수 즉 명제변수들을 찾아내는 함수입니다.
//찾아내서 EleAry에 원소를 넣은 후 리턴으로 원소의 개수를 반환합니다.
int checkElementry(char *Input,char *EleAry)
{
	int i=0,j,k;
	int valueNum = 0;
```

```
	while(Input[i]!='₩0')//명제가 끝날 때 까지 반복합니다.
	{
		switch(Input[i])
		{
		case '&': case '|': case '>': case '!':
			break;
		default://그 외에는 피연산자로 판별합니다.
			j=0;
			k=0;
			while(EleAry[j]!='₩0')
			{
				if(Input[i] == EleAry[j])//중복이 있을 경우 k를 1로 세팅합니다.
				{
					k = 1;
					break;
				}
				j++;
			}
			if(k==0)//끝까지 중복이 없을 경우 명제변수 배열에 문자를 추가합니다.
			{
				valueNum ++;
				EleAry[j] = Input[i];
				EleAry[j+1] = '₩0';
			}
		}
		i++;
	}
	return valueNum;//명제변수의 개수를 리턴합니다.
}
//---------------------------------------------------------------------------------
//cSwap은 스택에서 꺼낼 때 반대순서로 나오기 때문에 그걸 출력할 때는 거꾸로 출력해야 합니다.
//cSwap을 이용해서 거꾸로 만들고 출력하기 위해 만들어진 문자열을 거꾸로 바꿔주는 함수입니다.
void cSwap(char *cAry)
{
	int i=0,length;
	char swap;

	while(cAry[i]!='₩0')//문자열의 길이를 잽니다.
	{
		i++;
	}
	length = i-1;

	for(i=0;i<=length/2;i++)//문자열을 반대로 스왑합니다.
	{
		swap = cAry[i];
		cAry[i] = cAry[length-i];
		cAry[length-i] = swap;
	}
}
//---------------------------------------------------------------------------------
//Pasing Table은 기본적으로 & | > ! 순으로 되어있습니다.
int ParsingCheck(char Stack,char Input)
{
	//Stack이 우선순위가 높다 = 1,  Input이 우선순위가 높다 = 0
	int ParsingTable[4][4]={{1,1,1,0}, {1,1,1,0}, {1,1,1,0}, {1,1,1,0}};
	int i, j;
```

```
	switch(Stack)//Stack에 저장된 연산자의 Index값을 찾습니다.
	{
	case '&':
		i=0;
		break;
	case '|':
		i=1;
		break;
	case '>':
		i=2;
		break;
	case '!':
		i=3;
		break;
	}

	switch(Input)//Input되는 연산자의 Index값을 찾습니다.
	{
	case '&':
		j=0;
		break;
	case '|':
		j=1;
		break;
	case '>':
		j=2;
		break;
	case '!':
		j=3;
		break;
	}
	return ParsingTable[i][j];
	//각각의 Index에 해당하는 우선순위값을 ParsingTable에서 꺼내서 리턴합니다.
}
//-------------------------------------------------------------------------------------
//피연산자들과 연산자를 받아서 그에 해당하는 논리 연산을 마치고 리턴시켜주는 함수
int MainOperator(int value_1,char op,int value_2)
{
	switch(op)
	{
	case '!':
		if(value_1 == 0)
//만약 논리값이 0 ( false ) 일 때 1 ( True )를 리턴합니다.
		{
			return 1;
		}else
		{
			return 0;
		}
		break;
	case '|':
		if((value_1|value_2) != 0)
//두 연산자의 논리값의 Disjunction연산을 한 후 1 ( True ) or 0 ( False ) 값을 리턴합니다.
		{
			return 1;
		}else
		{
```

```
                        return 0;
                }
                break;
        case '&':
                if((value_1&value_2) == 1)
//두 연산자의 논리값의 Conjunction연산을 한 후 1 ( True ) or 0 ( False ) 값을 리턴합니다.
        {
                        return 1;
                }else
                {
                        return 0;
                }
                break;
        case '>':
                if(value_1 == 1 && value_2 == 0)
//두 연산자의 논리값의 Implication연산을 한 후 1 ( True ) or 0 ( False ) 값을 리턴합니다.
                {
                        return 0;
                }else
                {
                        return 1;
                }
                break;
        default:
//연산자로 입력되지 않은 다른 연산자의 경우 아래의 내용을 출력하고 종료합니다.
                printf("잘못된 연산자.₩n");
                return -1;
        }
}
//-------------------------------------------------------------------------------------
//거듭제곱 구하는 함수입니다. 명제변수에 따라서 출력 결과가 2^n개이기 때문에 필요합니다.
double LowPower(int under,int upper)
{
        int i;
        double result = 1.0;
        if(upper<0)//만약 승수가 0이하일 경우 나눠서 결과값을 리턴합니다.
        {
                for(i=0;i<(upper*-1);i++)
                {
                        result *= (double)under;
                }
                return result;
        }else if(upper==0)//만약 승수가 0일 경우 결과는 1이기 때문에 1.0을 리턴합니다.
        {
                return 1.0;
        }else//승수가 양의정수 일 경우 곱한 뒤 결과값을 리턴합니다.
        {
                for(i=0;i<upper;i++)
                {
                        result *= (double)under;
                }
                return result;
        }
}
//-------------------------------------------------------------------------------------
//명제변수의 개수와 그 크기에 해당하는 배열을 주면 Value Table만들어주는 함수입니다.
//명제변수가 3개면 3*2^3 배열을 입력받고 그 값에 F , T 를 의미하는 0,1을 세팅해주는 함수입니다.
```

```
void valueTableCreate(int **Ary,int valueNum)
{
	int i,j;
	for(i=0;i<valueNum;i++)
	{
		for(j=0;j<LowPower(2,valueNum);j++)
		{
			if((j&(int)LowPower(2,(valueNum-1)-i)) != 0)
//자신의 논리값이 1이여야 할 경우에 1값을 저장합니다.
			{
				Ary[i][j] = 1;
			}else
			{
				Ary[i][j] = 0;
			}
		}
	}
}
//-------------------------------------------------------------------------------
//명제변수와 대응되는 값을 리턴해주는 함수입니다.
int BackValue(int **Ary,char *EleAry,char Check,int InputNum)
{
	int i=0;

	while(EleAry[i] != '₩0')
	{
		if(EleAry[i] == Check)//만약 찾고자 하는 피연산자의 Index값을 찾았을 경우.
		{
			break;
		}
		i++;
	}
	return Ary[i][InputNum];//피연산자의 Index값에 해당하는 진리값을 리턴해줍니다.
}
//-------------------------------------------------------------------------------
//진리표 T,F찍을때 명제 길이를 재는 함수이며 길이를 리턴합니다.
int HowManyLength(char * Input)
{
	int i=0;
	while(Input[i]!='₩0')//문자열의 길이를 셉니다.
	{
		i++;
	}
	return i;
}
//-------------------------------------------------------------------------------
//콘솔창 크기가 바뀐 것과 text입출력도 설명합니다.
int main()
{
	FILE *fp;
	char InModify[200];//명제를 받기위한 문자 배열입니다.
	char TempModify[200]={'₩0'}; //임시로 합성명제가 생길 시 합성명제를 저장할 문자열입니다.
	char EleAry[10]={'₩0'};//명제변수들을 저장할 EleAry입니다.
	char TempOp;//연산하기위해 구조채에서 연산자를 추출하기 위한 변수입니다.
	int valueNum = 0;//명제변수의 개수를 셀 변수입니다.
	int EleValue = 0;//명제변수의 진리값을 저장하게될 변수입니다.
	int TempValue_1,TempValue_2,ResultValue;
```

```
//임시로 연산함수에 들어가게 될 피연산자들의 진리값과 결과 진리값을 저장할 변수들입니다.
	int i,j,k,p;//반복문을위한 index 변수들입니다.
	int modifyPrint = 0;//첫번째줄에는 합성명제들을 출력하기 위해 사용할 변수입니다.
	int modifyLength = 0;
//라인 양식을 맞추기 위해서 식들의 길이를 알아야 하는데 명제의 길이를 저장할 변수입니다.
	int HoleLineLength = 0;
//총 한줄이 몇개의 문자들로 이루어져있는지 길이를 저장할 변수입니다.
	int **valueAry;//명제변수의 진리값들을 저장할 배열을 만들기 위한 다중포인터입니다.

	Node Operator,Operand,*tail_1=&Operator,*tail_2=&Operand;//스택의 각 값들을 초기화 시켜줍니다.
	Operator.HeadArrive = 0;
	Operand.HeadArrive = 0;
	Operator.Modify[0] = '₩0';
	Operand.Modify[0] = '₩0';

	system("mode con:cols=150 lines=30");
	fp = fopen("Truth Table.txt","w");

	printf("진리표를 만듭니다. 합성명제를 입력해 주세요.₩n( 부정은 ! , 논리곱은 & , 논리합은 | , 논리함축은
> 로 표기해주세요. 괄호도 사용하시면 안 됩니다.) ₩n₩n명제 : ");
	//명제를 입력받습니다.
	scanf("%s",InModify);

	//명제변수의 개수를 세어 valueNum에 저장합니다.
	valueNum = checkElementry(InModify,EleAry);

	//명제변수의 진리값을 저장하기 위한 배열을 동적할당합니다.
	valueAry = (int**)malloc(sizeof(int*)*valueNum);
	for(i=0;i<valueNum;i++)
	{
		valueAry[i] = (int*)malloc(sizeof(int)*((int)LowPower(2,valueNum)));
	}
	//명제변수의 진리값 Table을 valueAry에 생성합니다.
	valueTableCreate(valueAry,valueNum);

	printf("₩n₩n- Truth Table -₩n₩n");
	fprintf(fp,"- Truth Table -₩n₩n");

	//명제변수의 개수에 맞춰서 2 ~ 2^n회 반복합니다.
	for(j=0;j<(int)LowPower(2,valueNum);j++)
	{
		printf("| ");
		fprintf(fp,"| ");
		HoleLineLength += 2;
//총 라인의 개수를 세기 위해 앞으로 출력되는 문자들의 개수를 셉니다.
		for(k=0;k<valueNum;k++)
		{
			//명제변수의 값들을 출력합니다.
			if(j==0 && modifyPrint == 0)
			{
				printf("%c ",EleAry[k]);
				fprintf(fp,"%c ",EleAry[k]);
				HoleLineLength += 2;
			}else
			{
				printf("%c ",valueAry[k][j]?'T':'F');
				fprintf(fp,"%c ",valueAry[k][j]?'T':'F');
			}
```

```
		}
		printf("|");
		fprintf(fp,"|");
		HoleLineLength += 1;

		i=0;
		//명제가 끝날 때까지 반복합니다.

		while(InModify[i] != '₩0')
		{
			//명제의 원소들을 검사합니다.
			switch(InModify[i])
			{
			case '!':case '&':case '|':case '>':
				while(Operator.HeadArrive == 1 && ParsingCheck(tail_1->Modify[0],InModify[i]))
				{
					TempModify[0] = '₩0';
					if(tail_1->Modify[0] == '!')
					{
				//만약 ! ( 부정 )연산자일 경우 피연산자는 한개만 Pop합니다.
						TempValue_1 = tail_2->Value;
						tail_2 = Pop(&Operand,TempModify);

						TempOp = tail_1->Modify[0];
						tail_1 = Pop(&Operator,TempModify);
					//ResultValue에 연산자의 결과값을 저장합니다.
						ResultValue = MainOperator(TempValue_1,TempOp,TempValue_1);
						tail_2 = Push(&Operand,ResultValue,TempModify);
						cSwap(TempModify);
					}else
					{
						TempValue_2 = tail_2->Value;
						tail_2 = Pop(&Operand,TempModify);
						TempOp = tail_1->Modify[0];
						tail_1 = Pop(&Operator,TempModify);
						TempValue_1 = tail_2->Value;
						tail_2 = Pop(&Operand,TempModify);
					//ResultValue에 연산자의 결과값을 저장합니다.
						ResultValue = MainOperator(TempValue_1,TempOp,TempValue_2);
						tail_2 = Push(&Operand,ResultValue,TempModify);
						cSwap(TempModify);
					}
					if(j == 0 && modifyPrint == 0)
					{
						printf("    %s    |",TempModify);
						fprintf(fp,"    %s    |",TempModify);
						HoleLineLength += HowManyLength(TempModify)+7;
					}else
					{
						modifyLength = HowManyLength(TempModify)+6;
						for(p=0;p<modifyLength/2;p++)
						{
							printf(" ");
							fprintf(fp," ");
						}
						printf("%c",tail_2->Value?'T':'F');
```

```
                                        fprintf(fp,"%c",tail_2->Value?'T':'F');
                                        for(p=0;p<(modifyLength-1)/2;p++)
                                        {
                                                printf("  ");
                                                fprintf(fp,"  ");
                                        }
                                        printf("|");
                                        fprintf(fp,"|");
                                }
                        }
                        TempModify[0] = InModify[i];
                        TempModify[1] = '₩0';
                        tail_1 = Push(&Operator,0,TempModify);
                        break;
                default:
                        TempModify[0] = InModify[i];
                        TempModify[1] = '₩0';
                EleValue = BackValue(valueAry,EleAry,InModify[i],j);
                        tail_2 = Push(&Operand,EleValue,TempModify);
                }
                i++;
        }
//문자열 끝났을 시 Operator에 남아있는 연산자들을 꺼내며 연산합니다.
        while(Operator.HeadArrive == 1)
        {
                TempModify[0] = '₩0';
                if(tail_1->Modify[0] == '!')
                {
                        TempValue_1 = tail_2->Value;
                        tail_2 = Pop(&Operand,TempModify);
                        TempOp = tail_1->Modify[0];
                        tail_1 = Pop(&Operator,TempModify);
                        //결과값을 ResultValue에 저장합니다.
                        ResultValue = MainOperator(TempValue_1,TempOp,TempValue_1);
                        tail_2 = Push(&Operand,ResultValue,TempModify);
                        cSwap(TempModify);
                }else
                {
        //만약 ! ( negation )연산자가 아닐 경우 피연산자는 두 개를 Pop합니다.
                        TempValue_2 = tail_2->Value;
                        tail_2 = Pop(&Operand,TempModify);
                        TempOp = tail_1->Modify[0];
                        tail_1 = Pop(&Operator,TempModify);
                        TempValue_1 = tail_2->Value;
                        tail_2 = Pop(&Operand,TempModify);
                        //ResultValue에 연산자의 결과값을 저장합니다.
                        ResultValue = MainOperator(TempValue_1,TempOp,TempValue_2);
                        tail_2 = Push(&Operand,ResultValue,TempModify);
                        cSwap(TempModify);
                }

                //연산자의 Pop연산이 일어났기 때문에 합성명제를 출력합니다.
                if(j == 0 && modifyPrint == 0)
                {
                        printf("    %s    |",TempModify);
                        fprintf(fp,"    %s    |",TempModify);
                        HoleLineLength += HowManyLength(TempModify)+7;
                }else
```

```
                {
                        //첫 번째 라인이 아닐 경우 합성명제의 진리값을 출력합니다.
                        modifyLength = HowManyLength(TempModify)+6;
                        for(p=0;p<modifyLength/2;p++)
                        {
                                printf("  ");
                                fprintf(fp,"  ");
                        }
                        printf("%c",tail_2->Value?'T':'F');
                        fprintf(fp,"%c",tail_2->Value?'T':'F');
                        for(p=0;p<(modifyLength-1)/2;p++)
                        {
                                printf("  ");
                                fprintf(fp,"  ");
                        }
                        printf("|");
                        fprintf(fp,"|");
                }
        }

        Operand.HeadArrive = 0;
        printf("₩n");
        fprintf(fp,"₩n");
        if(j==0 && modifyPrint == 0)
        {
                j = -1;
                modifyPrint = 1;
                for(p=0;p<HoleLineLength;p++)
                {
                        printf("=");
                        fprintf(fp,"=");
                }
                printf("₩n");
                fprintf(fp,"₩n");
        }
    }
    printf("₩n");
    fclose(fp);
    system("PAUSE");
    return 0;
}
```