EDDI
Electronic Design
Development Institute

에디로봇아카데미

임베디드 마스터 Lv2 과정

제 1기

2022. 04. 16

손표훈

CONTENTS

EDDI
Electronic Design
Development Institute

- 회로 심볼라이브러리 구현
  - 구현목표
  - 구현목적
- 저항 심볼
  - 추상화
  - 구조체
- 심볼 생성 함수
  - 리드 생성함수

회로심볼 라이브러리

EDDI
Electronic Design
Development Institute

➢ 구현 목적

→ Circuit Simulation 프로그램 구현 시 회로 심볼을 사용자 선택에 따라 생성하고 삭제 할 수 있는 라이브러리 구현
→ OpenGL API 사용법 숙달
→ 자료구조 활용

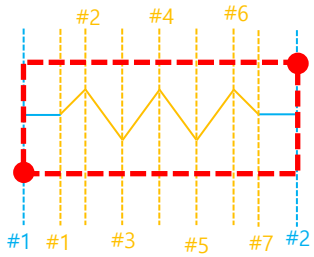
➢ 구현 목표

→ 마우스 클릭 시 해당 좌표에 심볼이 생성될 수 있는 기능이 있어야 한다
→ 마우스가 선택 여부를 파악 할 수 있는 구조를 가져야 한다
→ 각 심볼은 생성될 때 마다 순번을 갖도록 한다
→ 심볼생성시 리드가 같이 생성되어야 한다
→ 심볼의 리드는 와이어가 연결되었는지 판별할 수 있는 구조를 가져야 한다

저항 심볼

EDDI
Electronic Design
Development Institute

➢ 추상화



→ 저항 심볼은 2개의 리드좌표와 7개의 형상 좌표를 갖는다

→ 2개의 리드좌표는 와이어 연결 여부를 확인 할 수 있어야 한다

→ 저항심볼은 마우스 선택 여부를 판별하기 위한 경계선 좌표 2개를 갖는다

→ 저항심볼은 순번 값과 저항 용량 값(추가 필요)을 가져야 한다

➢ 구조체

```
/*
   @variables
   1. num : 리드의 번호
   2. vertex2D vertex : 리드의 2차원 좌표 값
   3. vertex2D *link : 리드의 연결된 와이어 좌표 값을 저장하는 포인터
                       (와이어 연결 기능 구현 후 연결 여부 판별하기 위해 필요함)
*/
typedef struct _symbol_lead symbol_lead;
struct _symbol_lead
{
        unsigned int num;
        vertex2D vertex;
        vertex2D *link;
};

/*
   @variables
   1. vertex2D *pVtx : 심볼형상의 2차원 좌표 값
   2. symbol_lead *lead : 심볼의 리드 좌표 값
*/
typedef struct _symbol2D symbol2D;
struct _symbol2D
{
        vertex2D *pVtx;
        symbol_lead *lead;
};
```

```
/*
   @variables
   1. num : 심볼의 번호
   2. sumbol2D symbol : 심볼형상의 좌표 값과 리드의 좌표 값
   3. symbol_queue *list : 다음 심볼을 가리키는 포인터
   4. boundary[2] : 심볼의 선택 영역으로 사각형 형상을 가지며, 2개의 좌표점을 이용한다
                    -----*
                                        |     | "*"이 영역을 구분하는 좌표점이 된다
                                        *-----
   5. create_symbol : 심볼의 종류별로 심볼을 생성하기 위한 함수 등록용 함수포인터
*/
typedef struct _symbol_queue symbol_queue;
struct _symbol_queue
{
        unsigned int num;
        symbol2D symbol;
        symbol_queue *list;
        vertex2D boundary[2];

        void (*create_symbol)(symbol2D *);
};
```

심볼 생성 함수

EDDI
Electronic Design Development Institute

➢ 리드 생성 함수

```
symbol_lead *create_lead(unsigned int num_lead)
{
        symbol_lead *tmp;
        tmp = malloc(sizeof(symbol_lead)*num_lead);
        tmp->link = NULL;
        return tmp;
}
```

→ 리드 선의 개수를 입력으로 받아 리드 선 개수대로 배열을 동적할당 한다

→ link는 리드선과 연결되는 와이어를 가리키는 2차원 좌표구조체 포인터

→ link는 리드선과 연결되는 와이어를 가리키는 2차원 좌표구조체 포인터

```
/*
  @variables
  1. num : 리드의 번호
  2. vertex2D vertex : 리드의 2차원 좌표 값
  3. vertex2D *link : 리드의 연결된 와이어 좌표 값을 저장하는 포인터
                (와이어 연결 가능 구현 후 연결 여부 판별하기 위해 필요함)
*/
typedef struct _symbol_lead symbol_lead;
struct _symbol_lead
{
        unsigned int num;
        vertex2D vertex;
        vertex2D *link;
};
/*
  @variables
  1. vertex2D *pVtx : 심볼형상의 2차원 좌표 값
  2. symbol_lead *lead : 심볼의 리드 좌표 값
*/
typedef struct _symbol2D symbol2D;
struct _symbol2D
{
        vertex2D *pVtx;
        symbol_lead *lead;
};
```

심볼 생성 함수

EDDI
Electronic Design Development Institute

➢ enqueue_symbol 함수

```
/*
   @parameter
   1. symbol_type : 저항인지 인덕터인지 판별하는 파라메터
   (1) R : 저항
   2. symbol_queue : symbol구조체 포인터 변수
   3. num : symbol의 번호
*/
void enqueue_symbol(char symbol_type, symbol_queue **pSym, unsigned int num)
{
        if(!(*pSym))
        {
                *pSym = create_symbol();
                (*pSym)->num = num;
                create_symbol_handler(symbol_type, *pSym);
                //(*pSym)->create_symbol(&(*pSym)->symbol, &basis, dx, dy);
                (*pSym)->create_symbol(&(*pSym)->symbol);
                return;
        }
        else
        {
                symbol_queue *backup;
                symbol_queue *new;

                while(backup->list)
                        backup = backup->list;

                new = create_symbol();
                new->num = num;
                create_symbol_handler(symbol_type, new);
                //new->create_symbol(&new->symbol, &basis, dx, dy);
                new->create_symbol(&new->symbol);
                backup->list = new;
                return;
        }
}
```

```
/*
   @variables
   1. num : 심볼의 번호
   2. sumbol2D symbol : 심볼형상의 좌표 값과 리드의 좌표 값
   3. symbol_queue *list : 다음 심볼을 가리키는 포인터
   4. boundary[2] : 심볼의 선택 영역으로 사각형 형상을 가지며, 2개의 좌표점을 이용한다
                    -----*
                                        |    | "*"이 영역을 구분하는 좌표점이 된다
                                        *-----
   5. create_symbol : 심볼의 종류별로 심볼을 생성하기 위한 함수 등록용 함수포인터
*/
typedef struct _symbol_queue symbol_queue;
struct _symbol_queue
{
        unsigned int num;
        symbol2D symbol;
        symbol_queue *list;
        vertex2D boundary[2];

        void (*create_symbol)(symbol2D *);
};
```

```
/*
   @parameter
   1. symbol_type : 저항인지 인덕터인지 판별하는 파라메터
   2. pSym : 심볼의 좌표값을 설정하는 함수포인터와 생성되는 위치가 저장된 구조체 포인터
      (1) void *create_symbol(symbol2D, vertex2D, int, int);
          (2) basis
*/
void create_symbol_handler(char symbol_type, symbol_queue *pSym)
{
        switch(symbol_type)
        {
                case 'r':
                case 'R':
                        pSym->create_symbol = create_symbol_R;
                        break;
        }
}
```

→ Symbol을 연결리스트 형태로 queue구조로 관리한다

→ 심볼에서 num변수는 queue 탐색 시 활용된다

→ enqueue_symbol함수는 파라메터 심볼의 타입(저항, 인덕터 등)에 따라 symbol_queue 구조체에 심볼 형상에 해당하는 좌표 값을 할당하게 한다

→ 심볼 타입 파라메터에 따라 symbol_queue구조체의 create_symbol 함수포인터에 입력되는 create_symbol 함수가 달라진다

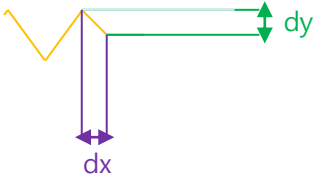
심볼 생성 함수

EDDI
Electronic Design Development Institute

➢ Create_symbol_R 함수

```
void create_symbol_R(symbol2D *R)
{
	int i;

	if(!(R->lead))
		R->lead = create_lead(2);

	//리드의 시작점을 설정한다
	R->lead[0].vertex.x = basis.x;
	R->lead[0].vertex.y = basis.y + (double)dy/2;
	R->lead[0].num = 1;
	//리드의 끝점을 설정한다
	R->lead[1].vertex.x = basis.x + (5*dx);
	R->lead[1].vertex.y = basis.y + (double)dy/2;
	R->lead[1].num = 2;

	R->pVtx = malloc(sizeof(vertex2D)*7);

	//심볼의 대각선 성분의 좌표를 얻는다
	for(i = 0; i < 7; i++)
	{
		//심볼 대각선의 시작점과 끝점
		if(i == 0)
		{
			R->pVtx[i].x = R->lead[0].vertex.x + dx;
			R->pVtx[i].y = R->lead[0].vertex.y;
		}
		else if(i == 6)
		{
			R->pVtx[i].x = R->lead[1].vertex.x - dx;
			R->pVtx[i].y = R->lead[1].vertex.y;
		}
		//심볼 대각선의 -방향
		else if(i%2 == 0)
		{
			R->pVtx[i].x = R->pVtx[0].x + (double)dx*(double)i/2;
			R->pVtx[i].y = basis.y;
		}
		//심볼 대각선의 +방향
		else
		{
			R->pVtx[i].x = R->pVtx[0].x + i*(double)dx/2;
			R->pVtx[i].y = basis.y + dy;
		}
	}
```

```
/*
  @variables
  1. num : 리드의 번호
  2. vertex2D vertex : 리드의 2차원 좌표 값
*/
symbol_queue *set_symbol_boundary(symbol_queue **pVtx)
{
	//심볼 선택을 위한 사각형 영역의 왼쪽 모서리 좌표 값을 설정한다
	(*pVtx)->boundary[0].x = (*pVtx)->symbol.pVtx[0].x;
	(*pVtx)->boundary[0].y = (*pVtx)->symbol.pVtx[0].y-dy/2;
	//심볼 선택을 위한 사각형 영역의 오른쪽 모서리 좌표 값을 설정한다
	(*pVtx)->boundary[1].x = (*pVtx)->symbol.pVtx[6].x;
	(*pVtx)->boundary[1].y = (*pVtx)->symbol.pVtx[6].y+dy/2;
}
```



→ 저항 심볼에 해당하는 2차원 좌표 값을 2차원 심볼 구조체에 저장한다

→ 생성 기저(basis 변수)와 좌표이동 최대, 최소값(dx, dy)에 따라 저항의 크기도 변경 될 수 있게 한다

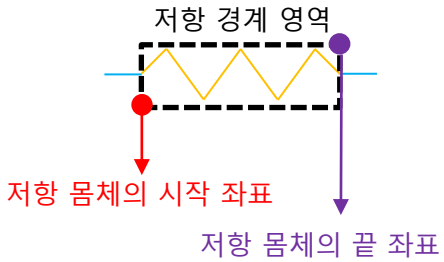
→ 저항 심볼의 리드 좌표를 추상화된 저항에 맞게 설정한다

심볼 생성 함수

EDDI
Electronic Design Development Institute

➢ set_symbol_boundary 함수

```
symbol_queue *set_symbol_boundary(symbol_queue **pVtx)
{
        //심볼 선택을 위한 사각형 영역의 왼쪽 모서리 좌표 값을 설정한다
        (*pVtx)->boundary[0].x = (*pVtx)->symbol.pVtx[0].x;
        (*pVtx)->boundary[0].y = (*pVtx)->symbol.pVtx[0].y-dy/2;
        //심볼 선택을 위한 사각형 영역의 오른쪽 모서리 좌표 값을 설정한다
        (*pVtx)->boundary[1].x = (*pVtx)->symbol.pVtx[6].x;
        (*pVtx)->boundary[1].y = (*pVtx)->symbol.pVtx[6].y+dy/2;
}
```



→ 저항 심볼을 마우스로 클릭 했을 때 마우스가 저항을 클릭한건지 판별하는 저항의 경계 값 좌표를 설정한다

→ 저항 심볼의 경계 좌표는 저항 몸체의 시작 좌표-저항 몸체의 끝 좌표로 설정한다