게임 프로그램 이해를 통한 "프로그래밍" 학습

신 재 훈　김 종 훈†
제주교육대학교 초등교육과　†제주교육대학교 컴퓨터교육과
HUN0124@chollian.net　jkim@jejue.ac.kr

Programming Learning by Understanding of Game Programs

Jaehun Shin, Jong-Hoon Kim
Jeju National University of Education

요 약

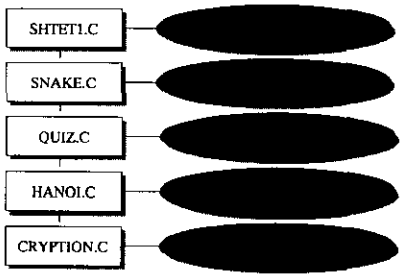
'컴퓨터 교육'이라는 개념은 컴퓨터를 이용한 교육과 컴퓨터를 배우는 교육 두 가지 의미를 동시에 가지고 있기 때문에 사용하는데 주의를 기울여야 한다. 후자의 개념으로서 '컴퓨터 교육'은 전산학에 대한 개론 및 프로그래밍과 관련된 학습이 핵심을 이룬다. 특히 프로그래밍의 학습의 경우는 코딩, 컴파일링, 디버깅 등의 복잡한 과정과 함께 운영체제 및 하드웨어에 대한 지식 같은 프로그래밍 외적인 분야에 대해서도 다양한 이해가 요구된다. 기존의 프로그래밍 학습은 프로그래밍 언어의 문법에 대한 기계적 암기나 사용법을 익히는 것에 치중하여 논리적인 사고를 키우는데 적합하지 못했다. 이에 본 논문에서는 프로그래밍에 대한 기본적인 통찰력을 기를 수 있도록 프로그래밍 언어의 기초가 되는 C로 코딩된 게임 프로그램들을 분석·이해한다.

1. 서론

프로그래밍 언어를 배우는 것은 어학이 아니지만 실제 글을 쓰는 과정처럼 창조가 수반된 어려운 분야이다. Basic, Java, C, C++, C# 및 수많은 스크립트 언어들을 배우는 과정들이 모두 그러하다. 그러나 대부분의 경우 프로그래밍과 관련된 학습은 특정 언어와는 독립적이다. 어떻게 테스트하고 디버깅하느냐에 대한 세부사항은 언어마다 다르지만 문제를 해결하기 위한 전략, 방법은 결국 동일하다는 말이다. 따라서 본 논문에서는 프로그래밍 언어의 기초가 되는 C를 통해 코딩 실력을 강화하고 프로그래밍에 대한 기본적인 통찰력을 기를 수 있도록 적절한 소스들을 분석한다. 단순한 소스의 설명이 아닌 예제와 설명을 통한 접근방법을 사용할 것이다. 우선 개발자의 의도를 파악하고 전반적인 윤곽을 그린 뒤에 소스에 대해 간단히 설명하고자 한다. 본 논문에서 추구하는 목적은 프로그래밍의 테크닉을 배우는 것이 아니라는 것을 미리 밝혀두는 바이다. 테크닉은 단순히 컴퓨터를 속이는 기술에 지나지 않는다. 그보다 학습자에게 프로그래밍 경험을 쌓기 위한 도움을 주는데 의의를 둔다. 학습자는 프로그래밍 경험을 통해 어떠한 아이디어가 떠올랐을 때 결과를 예상할 수 있는 직관을 기르게 된다.

2. 프로그래밍 학습 접근 방법

프로그래밍 학습은 주로 문법을 숙지하고 사용법을 익힌 후, 다른 사람이 제작한 소스를 분석하고 수정하는 과정을 통해 이를 자신의 코드로 만들어 가는 과정으로 요약된다. 이것은 결코 단순 독해나 암기를 의미하는 것이 아니다. 그러한 단계를 뛰어넘어 적극적인 학습, 즉 문제를 해결하기 위한 소스 작성자의 접근 방법이나 철학을 익히는 과정이라고 할 수 있다[2]. 그러한 과정에 게임 프로그램을 분석의 대상으로 선정함으로서 학습자의 흥미를 배가시킬 수 있다. 프로그래밍 학습이 문법에 대한 기계적 암기나 프로그래밍 언어의 사용법을 익히는데 치중한다면 이는 학습자의 인지부담이 크며, 논리적 사고력을 기르는데 적합한 방법이 되지 못한다. 그래서 효과적인 프로그래밍 학습이 실시되기 위해서는 학습의 주체인 학습자에게 흥미와 내적 동기를 부여하고 학습자의 수준이나 관심을 고려한 학습방법을 제공하는 데에 대한 연구가 필요한 것이다[1]. 본 논문에서는 프로그래밍에 대한 기본 제어구조 및 알고리즘 등을 익히기 위한 단순한 프로그래밍으로 그 영역을 축소시킨다. 본 논문에서 제시하는 게임 소스는 5가지이다. <그림 1>은 소스들의 목록과 각 소스를 분석하며 익혀야할 핵심 요소들을 보여준다. 물론 이것이 프로그래밍 학습을 위한 체계적인 학습 모형이 될 수는 없다. 그러나 현재의 컴퓨터 교육에서 프로그래밍 학습에 대한 다양한 접근이 이루어지지 못하고 있다는 점을 생각할 필요가 있다[1]. 본 논문에서는 C에 대한 기본적인 문법을 응용하는 데서 시작하여 프로그래밍에 꼭 필요한 기본 자료구조 및 알고리즘 등을 게임 소스를 분석함으로서 익히고자 한다. 이것은 자연스럽게 프로그래밍에 대한 개념과 절차, 방법이 습득될 수 있는 새로운 방안으로서 제시하고자 한 것이다.



<그림 1> 본 논문에서 제시된 소스들

3. 게임 프로그램의 이해

3.1 SHTET1.C

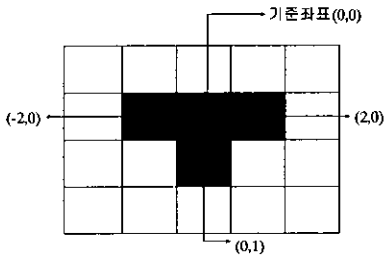
이 소스는 C 문법의 기본이 되는 데이터형, 제어문, 연산자, 함수, 배열 학습을 위해 제시되었다. 널리 알려진 테트리스 게임을 텍스트 모드에서 구현한 것으로 학습자의 포인터에 대한 부담을 덜어주기 위해 포인터를 일체 사용하지 않았다.

main 함수의 loop 구조 main 함수의 전체 루프가 실행되기 이전에 초기 작업을 수행하는 함수를 한 번 실행시킨다. 즉, 게임 화면을 출력하고 현재 화면상의 블록 모양이 기록된 tet 배열을 초기화하는 작업을 의미한다. main 함수의 loop 구조는 전체의 게임 loop안에 두 개의 for loop가 싸여있는 형식으로 되어있다.

```
void main()  {
  제어 변수 선언 및 초기화 함수 호출;
  for(;;) { /* 블록 하나가 바닥에 닿을 때까지의 loop */
      블록생성, 화면에 출력;
      게임 끝처리, 새로 만든 블록의 자리가 빈 공간이 아닐 때 게임을 끝냄;
      for(;;) { /* 하나의 블록이 한 칸 떨어지는 loop */
```

```
        switch(key) {
            case LEFT : leftmove(); break;
            . . .
        } } }
}
```

tet 배열과 pat 배열 화면에 출력되어 있는 블록의 모양을 tet이라는 이름의 2차원 배열로 저장했다. 각종 블록을 이동시키는 함수들은 이 배열을 참조하여 이동이 가능한지 판별하고 라인을 삭제하는 함수 역시 이 배열에 의해 삭제하며 삭제한 결과를 이 배열에 기록한다. pat 배열은 각 블록의 모양을 기억하는 배열로서 7개의 블록을 기억하고 있으며 블록이 회전하는 모양에 대해서도 기록하고 있다. pat 배열은 테트리스에서 사용될 블록의 형태를 기억하고 있는 배열이다. 우선 전체 블록의 모양은 7가지이며 한 블록당 최대 4개의 회전 모양을 가지고 있기 때문에 8바이트를 사용한다. 그래서 배열의 전체 크기는 pat[7][4*8], 즉 pat[7][32]가 된다. 8바이트로 블록을 표현하는 방식은 그림을 통해 알 수 있다. 배열상의 한 칸을 화면상의 두 칸으로 설정한 이유는 콘솔 한 칸의 크기는 8*16으로 직사각형이기 때문에 보기가 싫기 때문이다. 그래서 화면상의 두 칸을 배열상의 한 칸과 대응시킴으로서 정사각형을 표현한다.



<그림 2> pat 배열{0,0,-2,0,2,0,0,1} 를 참조로 나타낸 블록의 모양

<그림2>에서 8바이트는 2바이트씩 쌍을 이루어 4개의 위치를 표현하는데 중앙을 0,0으로 잡고 나머지는 그 점에서의 상대적인 좌표값으로 구현되어 있다. 따라서 그림의 블록은 배열{0,0,-2,0,2,0,0,1}로 표현된다. 0,0에 두 칸을 출력한 후 다음 좌표인 -2,0에 두 칸을 출력한다. 연속적으로 4개의 위치에 출력하면 하나의 블록 모양이 화면에 출력된다. 각 블록 당 회전한 모양을 포함하여 4개씩 블록 모양을 구현한다. 물론 4개가 아닌 것도 있다.

brick() 실제 화면에 블록을 출력하는 함수는 다음과 같다.

```
void brick(int flag) {
    char c;
    if (flag == 0) c = ' ';
    else c = 178; /* flag값에 따라 출력할 문자를 결정 */
    for(i=0;i<4;i++) {
        gotoxy(nowx+pat[nowbrick][nowrot*8+i*2],nowy+pat[nowbrick][no
        wrot*8+i*2+1]);
        putch(c);putch(c);  } /* for */
}
```

전역변수 nowbrick와 nowrot를 참조하여 현재 벽돌이 모양을 계산하고 계산된 벽돌의 모양을 nowx,nowy에 출력한다. 그런데 출력하는 방법과 삭제하는 방법이 동일하므로 main 함수로부터 brick 함수로 전달되는 인수에 의해 출력이냐 삭제냐가 결정할 수 있다. brick(1)이면 출력이고 brick(0)이면 삭제이다. 예를 들어, 블록을 한 칸 좌측으로 이동시키기 위해서는 우선 현재 위치의 블록을 삭제하고(brick(0)), x 좌표를 한 칸 좌측으로 증가시킨 후(nowx-=2), 블록을 출력하면 된다.(brick(1)) 삭제, 출력 여부를 결정한 뒤 pat 배열을 참조해 적당한 위치에 문자를 출력한다. (공백 또는 ■를 통해) 제어 변수 i가 0에서 3까지 돌아가는 동안 gotoxy에 의해 좌표가 이동하는데 이동되는 좌표는 배열에 들어있는 데이터와 가감하여 얻어진다.

pat[nowbrick][nowrot*8+i*2]

nowbrick은 현재의 블록값 0에서 6까지의 값이며 nowrot는 현재 회전 번호인데 이 값에 따라 같은 블록이라도 회전의 모양이 달라지게 된다. 한 회전 모양이 0이면 0번째 바이트부터, 1이면 8번째 바이트부터 블록의 데이터가 시작되어 회전된 블록의 모양을 보여주게 된다. i*2는 루프가 돌아감에 따라 0,2,4,6으로 변하여 순서대로 x좌표의 데이터를 가리키게 된다.

whataround() whataround 함수는 떨어지는 블록 주변을 검사하여 블록 근처에 무엇이 있는지 판별해준다. 따라서 블록을 이동시키기 전에 반드시 whataround 함수를 통해 현재 블록 상태를 검사해야 할 필요가 있다.

```
int whataround(int x, int y) { /* 빈 공간이면 0, 쌓인 블록에 부딪치면 1, 벽에 부딪치면 2를 return */
    k=0;
    for(i=0;i<4;i++) {
        j=tet[x+pat[nowbrick][nowrot*8+i*2]/2][y+pat[nowbrick][nowrot*8+i+
          1]];
        if (j>k) k=j;  }
    return k;
}
```

whataround 함수가 건네받은 x, y좌표로부터 실제 블록이 있는 부분의 배열값을 검사해주는 역할을 한다. pat 배열의 값으로 화면상에 블록이 쌓여있는 모양을 참조해 현재 블록이 있는 위치가 비어있는지 또는 쌓여있는 블록 위인지를 판별해주는 것이다. j에 그 배열값을 넣고 블록이 차지하고 있는 공간 중에 제일 큰 값을 k에 담아 리턴해준다. 즉, 빈 공간이면 0, 쌓여있는 블록 위라면 1, 벽과 부딪친 상태라면 2가 리턴되는 것이다. 따라서 main 함수의 조건 "whataround((nowx-30)/2, nowy-2) !=0)"는 블록이 처음 생성되었을 때 비어있는 상태가 아니므로 게임을 끝내기 위한 조건이 된다.

3.2 SNAKE.C

이 소스는 텍스트 모드의 문자열 입출력에 관한 학습을 정리해주는 역할을 하고 있다. 오래 전부터 소개되어온 소스이며 주로 main 함수에서 일들을 수행하므로 main 함수가 지나치게 길어진다는 점이 아쉽다. 소스에서 가장 핵심적인 사항은 뱀 꼬리의 위치를 기억하는 queue라는 자료구조를 이해하는 것이다. 꼬리는 계속해서 움직이므로 단순한 배열로는 그 위치를 다 기억할 수 없다. 따라서 snake[1000][2]라는 배열을 마련해 두고 main 함수에서 이 배열의 처음과 끝을 연결시킴으로서 원형 queue를 구현하고 있다. queue의 선두와 꼬리 포인터를 유지하기 위해 head, tail이라는 두 개의 포인터를 사용하여 두 변수의 차이가 현재 뱀 꼬리의 길이가 되는 것이다.

```
int snake[1000][2];   /* 꼬리의 위치를 기억하는 원형 queue */
int head, tail;   /*queue 내에 현재 위치를 기억하는 포인터 */
```

queue queue는 데이터의 삽입이 한쪽 끝(tail)에서 일어나고 삭제가 다른 쪽 끝(head)에서 일어나는 자료구조를 의미한다. queue의 처음 상태는 head와 tail이 모두 0을 가리키고 있다. 이때 데이터가 하나 들어오면 tail가 가리키는 위치에 들어오는 데이터를 저장하고 tail값을 증가시킨다. 데이터를 삭제하기 위해서는 head에 있는 데이터를 리턴하고 head값을 증가시키면 된다. 또한 tail값이 배열의 끝까지 증가하면 전체적으로 shift시켜야 한다. queue가 가득 찬 상태에서의 데이터 삽입은 매번 shift를 시켜야 하는 문제가 발생한다. 원형 queue는 일반적인 queue에서 발생할 수 있는 문제점을 보완한 것으로서 head와 tail을 원형으로 된 배열을 계속 돌도록 구현되었다. 이런 식으로 queue를 사용하면 shift시킬 필요없이 사용할 수 있게된다. queue를 이용해 구현된 뱀 꼬리의 실제 코드를 살펴보면 다음과 같다.

```
head++;
if (inctail == 0) tail++;
else inctail--; /* 꼬리의 증가분이 있으면 queue에서 꼬리의 위치를
                   그대로 유지함 */
if (head == 999) {   /* 원형 queue의 끝 부분 처리 */
   snake[0][0] = snake[998][0];  snake[0][1] = snake[998][1]; head =1;
}
if (tail == 999) tail = 1;
```

3.3 QUIZ.C

구조체라는 개념을 접하면서 이를 직접 실제 프로그래밍에 응용해보는 단계로 제시할 수 있는 것은 Linked List등의 자료구조가 일반적일 것이다. 그러나 문제와 보기, 정답을 하나의 구조체로 묶어 퀴즈 프로그램을 만들어 보는 것은 학습자에게 보다 흥미있는 경험이 될 것이다. 문제와 보기, 정답은 각기 형태가 다른 변수들이므로 배열이 사용될 수 없고 데이터들이 논리적, 유기적인 관계를 갖고 있는 하나의 데이터 그룹이므로 구조체로 묶어 놓으면 관리하기도 편리하다. 문제 구조체의 형태는 다음과 같다.

```
struct tagmunje {  /* 문제 data를 담을 구조체의 형틀 */
    char *jilmun;   /* 문제 문자열의 포인터 */
    char *bogi1;   /* 보기 1의 포인터 */
```

```
    char *bogi2;    /* 보기 2의 포인터 */
    char *bogi3;    /* 보기 3의 포인터 */
    int dap;        /* 정답 포인터 */
    int used;       /*한번 출제된 문제임을 표시하는 멤버변수*/
}
```

used 멤버변수 문제와는 직접적인 연관은 없으나 문제가 출제되었던 문제인지 체크하는 역할을 담당한다. 출제하는 문제를 선택하는 부분의 코드는 다음과 같다.

```
do { nowmun=random(MAXMUN); } while(munje[nowmun].used==1);
munje[nowmun].used = 1; /* 출제한 문제를 표시한다. */
```

별로 어려운 기술이라 할 수 없지만 일반적인 학습자의 경우 used를 구조체의 한 멤버로 같이 포함시킨다는 생각을 하기가 힘들다. 왜냐하면 일단 구조체의 형태를 설계할 때는 데이터의 구조만 생각하지 이후 코드가 어떤 식으로 짜여질 것이라는 것까지 생각하기 어렵기 때문이다.

3.4 HANOI.C

프로그래밍을 하다보면 복잡한 프로그램을 재귀호출을 사용하여 간단하게 나타낼 수 있는 경우가 많이 있다. 재귀호출은 사용법이 간단함과 동시에 소스를 이해하기 어렵게 만들기도 한다. 때문에 어떠한 조건에서 리턴 되어야 하는가를 잘 설정할 필요가 있다. 잘못 설정하면 무한루프에 빠질 수도 있기 때문이다. 재귀호출을 사용하는 대표적인 예로 하노이 탑을 소개한다.

재귀호출 하노이 탑은 큰 원반 위에 더 작은 원반을 쌓아 놓은 것으로 한 쪽 막대기에서 다른 막대기로 옮길 때 작은 원반 위에 큰 원반을 놓을 수 없다. 이러한 조건에서 모든 원반을 다른 막대기로 옮기는 방법을 찾는 것으로 재귀호출을 사용한다. 재귀호출을 하는 프로그램의 기본적인 형식은 다음과 같다.

```
재귀호출함수() {
    if (return 조건, ending 조건)  return 값;
    else {  재귀호출함수();  ... }
}
```

하노이 탑 알고리즘의 원리는 간단하다. n개의 원반을 처음 위치에서 목적 위치까지 이동시키기 전에 n-1개를 우선 임시 저장 위치로 이동시키고, 제일 아래에 있는 원반을 목적 위치에 이동시킨 후 n-1개를 목적 위치로 이동시키면 된다. 이 같은 과정을 재귀호출을 이용하여 원반의 개수가 1개일 때까지 반복하는 것이 기본 원리이다. 여기서 가장 밑에 있는 가장 큰 원반의 번호는 1, 가장 작은 원반의 번호는 n이다.

```
void hanoi (int n, int a, int b, int c) {
    /* n개의 원반을 a에서 b까지 c라는 임시 저장위치를 이용하여
       옮긴다는 의미 */
    int d;
    if (n&gt;0) {
        hanoi(n-1,a,c,b); /*n-1원반을 임시저장위치로 옮긴다.*/
        count++;
        push(d=pop(&stk[a]),&stk[b]); /*원반을 스택으로 옮김*/
        . . .
        print_disk();
        hanoi(n-1, c, b, a); /* n-1 원반을 목적지로 옮긴다. */
    }
}
```

stack stack은 순서화된 자료구조 중의 하나로 First In First Ou 구조로서 나중에 들어온 데이터가 먼저 나오게 되어있는 구조로 되어있다. queue와는 달리 stack에서는 삽입과 삭제가 한 군데에서 이루어진다. 따라서 하노이 탑의 저장장소인 막대기는 stack으로 구현되어야할 필요가 있다. 삽입과 삭제가 이루어지는 곳을 일반적으로 top라고 한다. top이라는 변수가 항상 stack의 맨 위의 위치를 가리키고 있다. 데이터가 하나 추가되면 top의 위치에 데이터를 저장하고 top의 값을 하나 감소시킨다. 데이터가 삭제될 때에는 top의 값을 감소시키고 그 top의 위치에 데이터를 리턴한다. 그러면 top은 다음에 들어올 데이터가 들어가게 될 위치를 가리키게된다. 초기상태의 stack은 값이 없고 top는 새로 들어올 데이터가 저장될 위치를 나타내고 있다. top의 값이 stack을 초과하면 더 이상 stack은 데이터를 추가시킬 수 없게 된다.

3.5 CRYPTION.C

Stack 구조의 구현과 알파벳을 숫자화하여 key값을 더하고 빼서 다시 문자화하는 방법을 이용한 암호화, 복호화 및 자동으로 복호화하는 알고리즘이 소개된 프로그램이다. 위와 같은 알고리즘을 Caesar's Cipher라고 하는데 Caesar's Cipher 방법을 변형하여 영문 텍스트 파일로부터 문자를 읽어들이면서 이를 Stack에 쌓고 입력이 끝나면 Stack으로부터 한 문자씩 꺼내어 암호화, 복호화 시키면서 새로운 파일을 생성한다. 물론 새로 생성된 파일의 내용은 원래의 파일 내용과 반대로 되어있다.

암호화, 복호화 알고리즘 암호화, 복호화 알고리즘을 실행하기에 앞서 파일의 내용을 한 자씩 읽어들여 모두 소문자로 변화시켜 stack에 저장한다. stack에서 꺼내어 새로운 파일에 써넣기 전에 암호화, 복호화 알고리즘을 실행하는 것이다.

```
data cryption(data d)  {
    if ((('a'<=d)&&(d>='z'))){ d = ((d-'a'+(key_value%26))+26)%26+'a'; }
    return d;
}
```

암호화, 복호화는 한 자씩 실행되며 영어 알파벳의 경우에만 암호화 및 복호화를 실행한다. 복호화의 경우는 읽어들인 key 값에 음수를 취하여 key 값을 설정한다. 계산식에서 key_value%26은 알파벳이 26자이므로 key 값도 26 이하의 값을 취하도록 하기 위해서이다. 문자들을 숫자형으로 바꾸기 위해 -'a'를 하였다. 26을 더한 것은 만약 계산의 결과가 음수인 경우는 양수로 만들어 주기 위함이고 26을 나눈 이유는 key_value%26에서와 같이 결과가 26을 넘지 않도록 하기 위해서이다. 그래서 계산식의 결과는 0에서 25 사이의 값을 갖고 여기에 다시 'a'를 더함으로서 다시 문자 변환을 하도록 한 것이다.

Linked list를 이용한 stack구조의 구현 적은 메모리를 사용하기 위해 고안된 자료구조인 선형연결리스트를 사용하여 stack을 구현하도록 한다.

```
void initialize(Stack *stk) {  /* stack의 초기화 */
    stk->cnt = 0;    stk->top = NULL;
}
void push(data d, Stack *stk) { /* stack에 데이터를 삽입 (push) */
    LinkedList p;
    새로운 리스트 P를 생성하고 메모리가 할당되지 못할 경우 에
    러문자를 출력;
    p->d =d;    p->next = stk->top;    stk->top = p;    stk->cnt++;
}
data pop(Stack *stk) {  /* stack에서 데이터를 리턴 (pop) */
    data d;    LinkedList p;    d = stk->top->d;    p = stk->top;
    stk->top = stk->top->next;    stk->cnt--;    free(p);    return d;
}
```

4. 결론

프로그래밍 학습에 대한 새로운 접근법의 일환으로 본 논문에서는 간단한 게임 소스를 제시하고 이를 분석함으로서 프로그래밍 학습에 도움이 되고자 하였다. 학습자가 프로그래밍 언어에 대한 문법을 숙지하고 사용법을 익힌 후, 다른 사람이 제작한 소스를 분석하는 과정에 본 논문에서 제시된 소스들을 사용하여 얻을 수 있는 효과는 다음과 같다. 첫 번째로, 게임 소스 분석을 통해 프로그래밍 학습에 대한 흥미와 관심을 증대시킬 수 있다는 점이다. 효과적인 프로그래밍 학습에 있어 학습자의 관심과 흥미는 대단히 중요하다. 이를 위해 게임 소스를 분석의 대상으로 선정하였다. 두 번째로, 문제를 해결하기 위한 소스 작성자의 접근 방법이나 철학을 익힐 수 있다는 점이다. 프로그래밍 학습에서 다른 사람이 만든 소스를 분석하는 것은 구체적으로 소스 작성자의 문제 해결력과 논리적 사고를 배울 수 있다는 것을 의미한다. 셋째, 교재에서 제공하는 짧고 단순한 기본 예제를 탈피하여 프로그래밍에 대한 진지한 경험을 가질 수 있다는 점이다. 교재에 제시되어 있는 레퍼런스의 함수 하나를 익히고 이에 대한 짧은 사용법을 단순히 코딩하고 암기하기보다는 이들을 응용하여 적절한 코딩 할 필요가 있는 것이다.

참고문헌

[1] 박원길, 이재무, 아동과 초보자를 위한 프로그래밍 학습 시스템의 설계, 한국정보교육학회 학술발표논문집, 2000

[2] Herbert Schildt, C++ : complete Reference, 이한출판사, 1998

[3] 김상형, C++을 내것으로, 가남사, 1995

[4] 김태훈, 조영호, 알기쉬운 C Programming, 사이버출판사, 2000

[5] 이상환, 김응석, 이승준, 백운기, 윤경구, 초보자를 위한 프로그래밍 마스터, 북마크, 1999

[6] 민승재, 대학생을 위한 C Project, 한컴프레스, 1997