**CSE2035 C 프로그래밍**

(설계 프로젝트 1 : “방 안에 최대한 많은 박스 배치하기”)

담당교수 : 서강대학교 컴퓨터공학과 김세준 교수님

20151603 전진우

**1. 설계 문제 및 목표**

이번 프로젝트에서는 “방 안에 최대한 많은 박스를 배치하는 프로그램”을 제작해야 한다. 이에 따른 요구사항은 다음과 같다.

|  |
| --- |
| 주어진 R\*C ( 1 <= R,C <= 100 ) 크기의 방에 N (1 <= N <= 9 ) 개의 박스를 최대한 많이 배치하는 프로그램을 제작한다.   * 구현 환경 : Linux (cspro 서버) 기준 * Pointer를 사용하여 주어진 문제를 해결한다. * 대 괄호 [] 사용 금지 * 전역 변수의 사용 금지 * 정적 배열의 사용 금지 |

이 프로그램의 제작과정은 C 프로그래밍 수업의 기본 과정을 따라서 제작되었음을 밝힌다.

**2. 요구사항**

**2.1 합성**

* R x C의 일정한 크기의 방에 최대한으로 넣을 수 있는 상자의 개수를 구하는 프로그램을 구현할 것이다. 또한 추가 기능으로써 방을 가득 채울 수 있는 상자의 배치가 있는지도 확인 할 것이다.
* 필요한 C언어의 문법적 지식으로는 동적으로 메모리를 할당하는 방법, 포인터의 활용법, 기초적인 흐름제어와 반복문 등이 있다.
* 학교 cspro서버의 linux 환경에서 제작이 이루어졌으며 특별한 컴파일 옵션은 필요하지 않다.
* 컴퓨터의 연산이 약 0.001 ms에 한 번 이루어지므로 약 1억번의 연산을 수행하는데 1초가 걸린다. 따라서 이 이상으로 연산을 하게 될 경우 그 만큼의 시간이 소요된다.

**2.2 분석**

이 프로젝트에서 구현해야 할 주요 기능은 다음과 같다.

* 방에 박스를 넣는 함수
* 방에 들어있는 박스를 제거하는 함수
* 입력 받은 박스를 오름차순 정렬하는 함수
* 방이 가득 차 있는지 확인하는 함수
* R, C까지 돌면서 들어간 박스의 개수를 구하는 함수

(함수의 자세한 설명은 2.3 제작으로 넘긴다.)

이번 프로젝트에서 사용될 방은 2차원 형태이므로 2차원 배열을 이용한다. 박스의 정보는 높이, 너비, 번호로 이루어져 있으므로 n X 3의 2차원 배열을 이용한다. 또한 이 때, 방과 박스는 입력 받은 값에 따라서 그 크기가 변해야 하므로 동적 할당 즉, memory allocation을 이용한다.

* 상자의 배치 방법

n x m의 방에는 n x m개의 칸이 존재한다. 따라서 n x m개의 칸을 순서대로 조사하여 정렬된 상자의 번호(이하 정. 상. 번)에 맞는 상자가 들어가는지 확인한다. 만약 들어간다면 상자를 삽입하고 다음 정. 상. 번으로 넘어가서 다시 같은 일을 반복한다. 마지막 정. 상. 번에 도달했을 때 들어가 있는 상자의 개수가 최대인지 확인하고 저장한다. 이후 이전 정. 상. 번으로 돌아가서 조사하는 방의 다음 칸을 검사한다. 이번 정. 상. 번에 대한 모든 방의 칸에 대한 검사가 완료되었을 때 다시 이전 정. 상. 번으로 돌아가 방의 다음 칸을 검사한다. 이러한 일을 반복적으로 실행하면 방에 상자가 들어가는 모든 방법에 대하여 한번씩 조사해 볼 수 있다. 따라서 그 중에서 최대의 값을 찾으면 방에 넣을 수 있는 최대의 상자의 개수를 알아 낼 수 있으며 방이 가득 채워지는 상자의 배치 방법이 존재하는지 여부도 알 수 있다.

**2.3 제작**

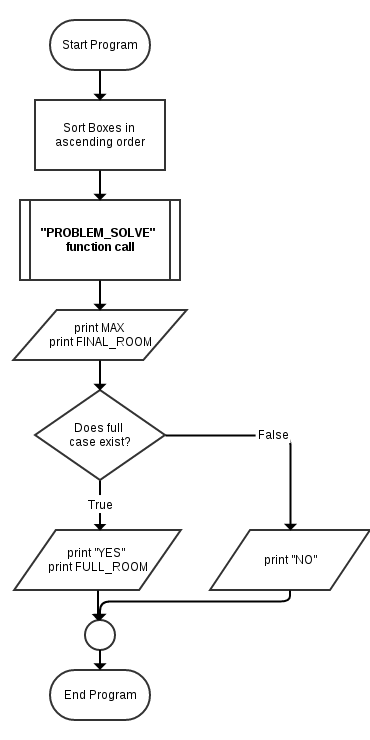
1. main 함수

프로그램이 실행되는 메인 함수이다.

리턴 값: 0

인자: 없음

기능: 프로그램의 input과 output 그리고 function call을 담당한다.



***Figure 2.3.1 Main Function Flowchart***

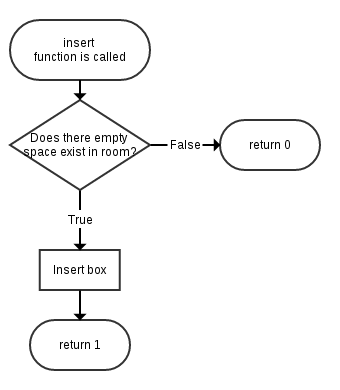
1. insert함수

박스를 주어진 방의 좌표에 삽입한다.

리턴 값: 삽입에 성공하면 1을 실패하면 0을 반환한다. 만약 상자가 room을 벗어난다면 -1을 반환한다.

인자: row(방의 높이), col(방의 너비), y(박스를 넣을 y좌표), x(박스를 넣을 x좌표), room(방), box(상자), number(정렬된 박스의 순서 번호), rotate(회전)

기능: 우선 상자가 방을 벗어나는지 검사한다. 벗어난다면 -1을 반환하고 그렇지 않다면 주어진 방의 (x, y) 좌표에서부터 (x + width - 1, y + height - 1)까지가 비어있는지를 확인하고 비어있다면 박스를 주어진 방의 빈 공간에 삽입한다.



***Figure 2.3.2 Insert Function Flowchart***

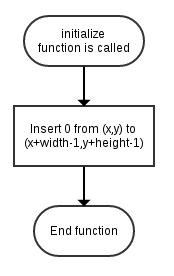
1. initialize 함수

주어진 크기만큼 방을 0으로 초기화 한다.

리턴 값: 없음

인자: y(박스를 넣을 y좌표), x(박스를 넣을 x좌표), room(방), height(초기화할 높이), width(초기화할 너비)

기능: 주어진 방의 (x, y) 좌표에서부터 (x + width – 1,y + height – 1)까지를 모두 0으로 초기화한다.



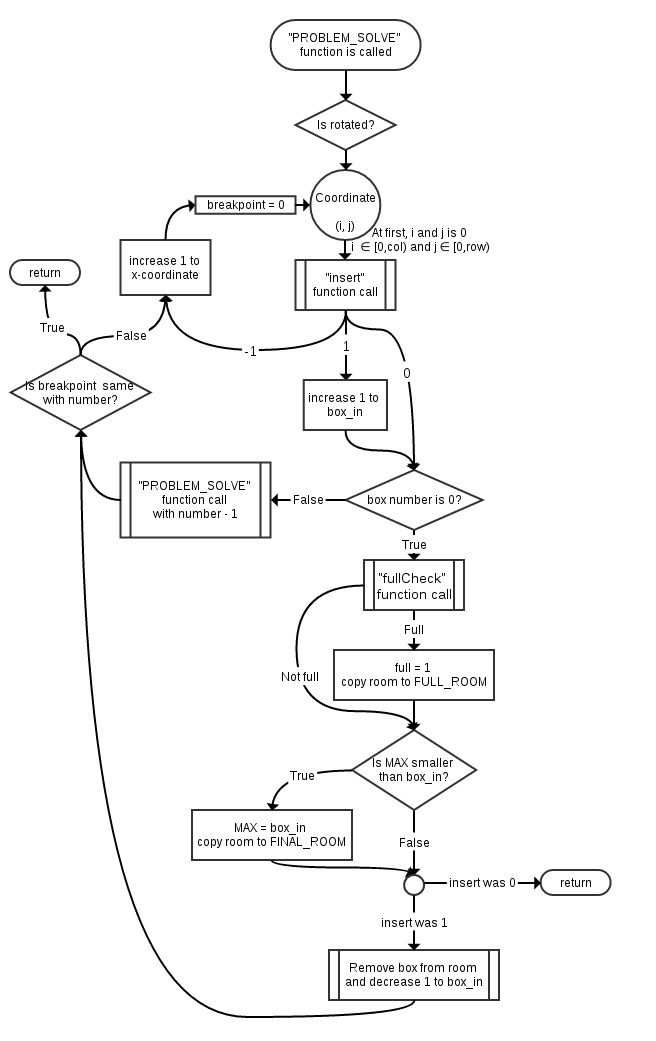
***Figure 2.3.3 Initialize Function Flowchart***

1. PROBLEM\_SOLVE 함수

리턴 값: 없음

인자: row(방의 높이), col(방의 너비), room(방), box(상자), number(정렬된 상자의 순서 번호), MAX(최대로 삽입된 상자의 수), FINAL\_ROOM(최대로 삽입되었을 때의 room의 상태), FULL\_ROOM(room이 가득 찼을 때의 room의 상태), box\_in(삽입된 상자의 수), breakpoint(1부터 상자 순서 번호 number까지 중에서 몇 개의 상자가 room에 들어가 있는지를 나타내는 값), full(방이 가득 찼는지 알려주는 값)

기능: number번째의 상자를 room에 (0,0)부터 (row-1, col-1)까지 넣어본다. 이 때, 삽입에 성공하면 다음 번호로 PROBLEM\_SOLVE를 실행한다. 만약 삽입에 모두 실패 하면 다음 번호로 PROBLEM\_SOLVE를 실행한다. 이 때 PROBLEM\_SOLVE는 삽입한 상자의 개수 box\_in을 모두 같이 가지고 있는다. 따라서 삽입에 성공할 때마다 box\_in을 1씩 늘려가며 최종적으로 box\_in의 최대값을 MAX에 저장한 채로 실행을 종료한다.



***Figure 2.3.4 PROBLEM\_SOLVE Function Flowchart***

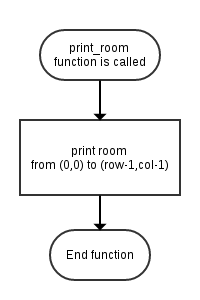
1. print\_room 함수

주어진 방(2차원 배열)을 출력한다.

리턴 값: 없음

인자: room(방), row(방의 높이), col(방의 너비)

기능: 입력 받은 room의 0에서 row – 1, 0에서 col - 1까지 출력한다.



***Figure 2.3.5 Print\_Room Function Flowchart***

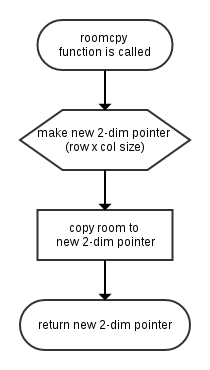
1. roomcpy 함수

주어진 방(2차원 배열)을 복사하여 반환한다.

리턴 값: 복사된 방(2차원 배열)

인자: room(방), row(방의 높이), col(방의 너비)

기능: 새로운 2차원 배열을 만들고 memory allocation을 실행한다. 그 후 인자로 받은 room을 복사하고 복사된 배열을 반환한다.



***Figure 2.3.6 Roomcpy Function Flowchart***

1. free2dim 함수

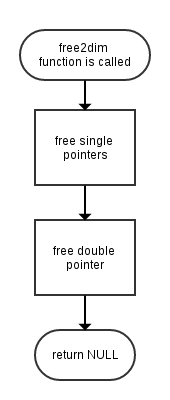
주어진 동적으로 할당된 2차원 포인터를 free한다.

리턴 값: NULL

인자: target(free할 2차원 포인터), row(2차원 포인터에 들어있는 1차원 포인터의 개수)

기능: target의 0번째부터 row - 1번째 인덱스를 돌면서 free해주고 마지막으로 target을 free한다. target을 초기화 해주기 위해서 NULL값을 반환한다. 따라서 다음과 같이 사용한다.





***Figure 2.3.7 Free2dim Function Flowchart***

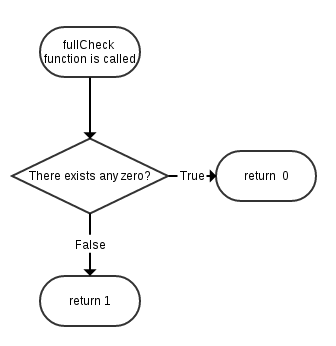
1. fullCheck 함수

room이 가득 차 있는지 확인한다.

리턴 값: room이 가득 차 있으면 1, 아니면 0을 반환한다.

인자: room(검사할 방), row(방의 높이), col(방의 너비)

기능: (0, 0)에서 (row-1, col-1)까지 돌면서 room이 가득 차 있는지 확인한다.



***Figure 2.3.8 FullCheck Function Flowchart***

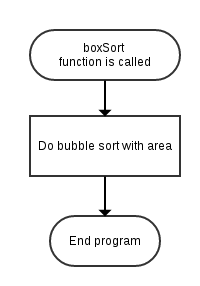
1. boxSort 함수

입력 받은 상자를 면적 순서로 정렬한다.

리턴 값: 없음

인자: box(정렬할 상자 배열), number(상자의 개수)

기능: 상자를 bubble sort를 이용하여 정렬하였다. 이 때 box를 이용함에 있어서 call by reference 방식을 채택하였다.



***Figure 2.3.9 BoxSort Function Flowchart***

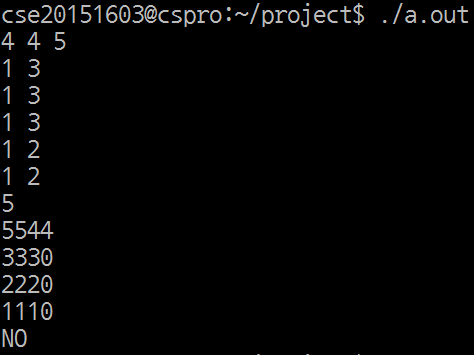
**2.4 시험**

* Test case no. 1



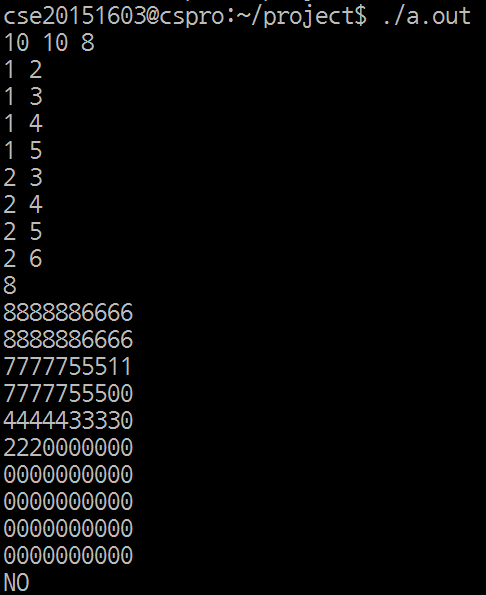
***Figure 2.4.1 Test case 1***

* Test case no. 2



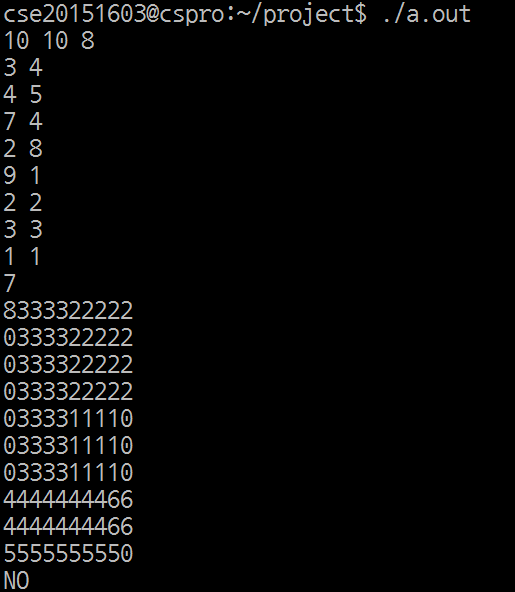
***Figure 2.4.2 Test case 2***

* Test case no. 3



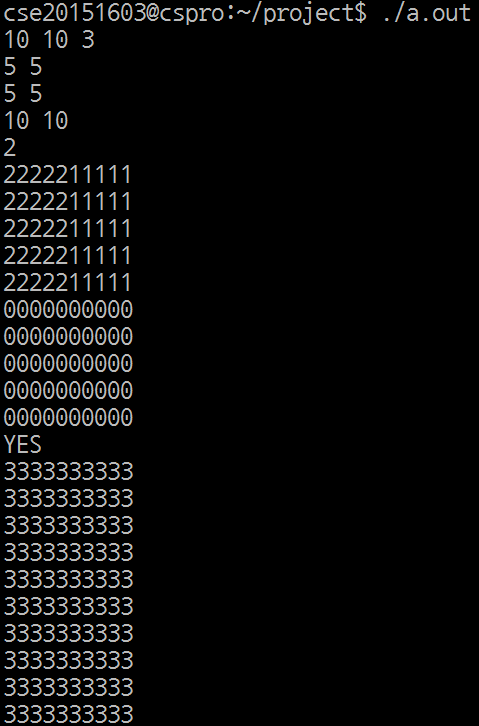
***Figure 2.4.3 Test case 3***

* Test case no. 4



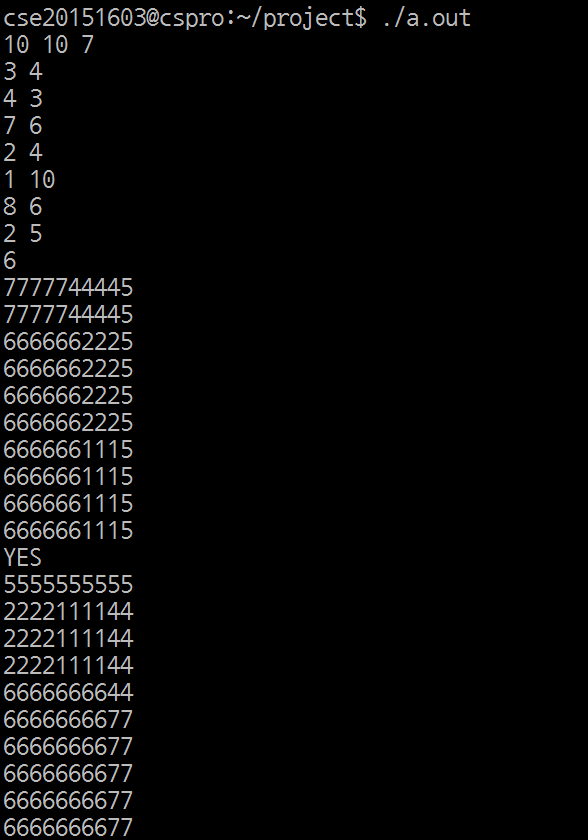
***Figure 2.4.4 Test case 4***

* Test case no. 5



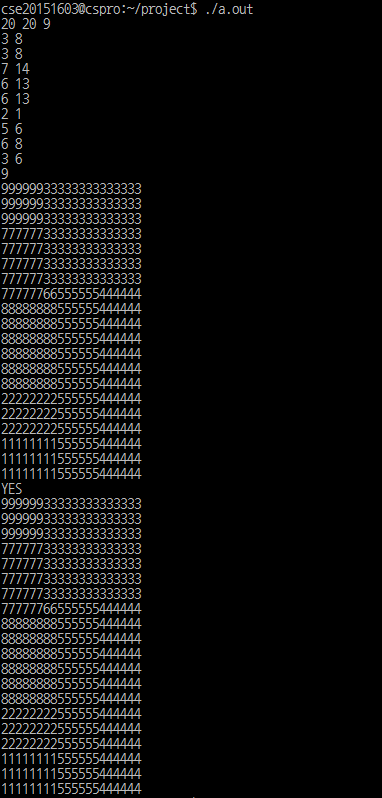
***Figure 2.4.5 Test case 5***

* Test case no. 6



***Figure 2.4.6 Test case 6***

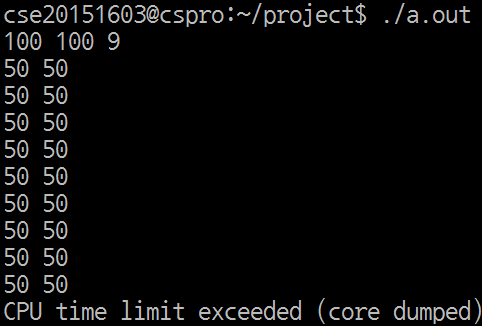
* Test case no. 7



***Figure 2.4.7 Test case 7***

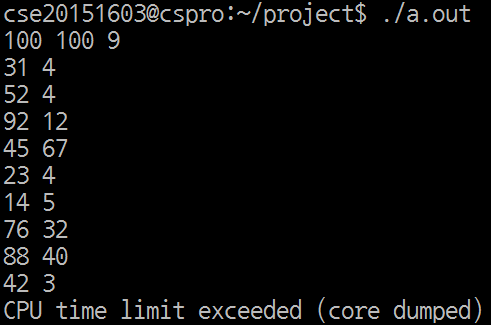
**2.5 평가**

* Failure no. 1



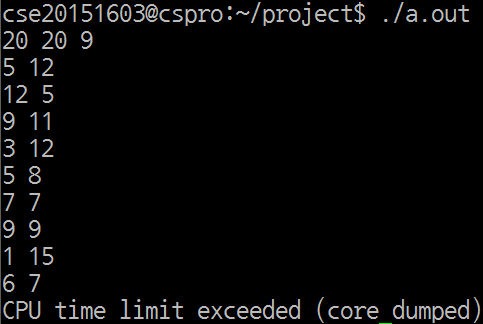
***Figure 2.5.1 Failure case 1***

* Failure no. 2



***Figure 2.5.2 Failure case 2***

* Failure no. 3



***Figure 2.5.3 Failure case 3***

**2.6 보건 및 안정**

이 프로그램에는 치명적인 단점이 존재한다. 구현한 방법이 이론상으로는 완벽해 보이지만 실제로 프로그램이 실행되면 연산의 횟수가 기하급수적으로 늘어나 원하는 답을 얻기 위해서 짧게는 몇 초, 길게는 몇 시간까지 기다려야 하는 일이 발생한다. 이러한 경우에는 cspro 서버에서 cpu사용량을 제한하기 위해 프로그램을 죽이는 일이 발생한다.

**2.7 생산성과 내구성**

이 프로그램을 처음 구상했을 때의 방법으로는 100 x 100 에 9개의 상자가 들어온다면 대략 1036이상의 연산을 수행하게 된다. 시간으로 계산해보면 1027초에 해당하는 시간이다. 따라서 불필요한 연산을 줄이고 효율적으로 값을 도출해 낼 수 있도록 케이스들을 건너 뛸 수 있는 부분을 고려해야만 했다.

첫 번째 방법은 breakpoint 변수를 이용한 방법이다.

n개의 상자를 넣는 경우 x번째 상자를 검사하고 있다고 했을 때, n ~ x+1까지의 상자는 들어가 있던 안 들어가 있던 그 위치가 고정되어 있을 것이다. 따라서 1 ~ x까지의 상자 만을 고려했을 때, x개의 상자가 들어가면 넣을 수 있는 최대한의 상자를 넣은 것이다. 우리가 구하는 값은 굳이 모든 상자 배치를 보지 않더라도 최대로 넣은 값을 구하면 되는 것이므로 더 이상 x번째 상자에 대해서는 다른 배치를 보지 않고 x+1번째의 상자를 다음 칸으로 이동시켜 배치를 확인한다. 이렇게 하면 상자의 모양과 개수에 따라서 수십 분 걸리던 것을 1초 이내로 줄일 수 있었다.

이러한 판별을 위해 breakpoint 변수를 사용하였으며 box\_in 변수와 다르게 상자를 counting 한다. box\_in 변수는 처음 들어오는 상자인 n번째 상자부터 수를 세지만 breakpoint 변수는 1번째 상자부터 수를 세어나간다.

두 번째 방법은 box\_jump변수를 이용한 방법이다.

insert함수에서 만약 상자가 방을 벗어나는 크기를 가진다면 -1을 반환하게 하여 box\_jump에 저장하였다. 이러한 경우 해당 좌표의 row에서의 이후의 col값은 모두 상자가 방을 벗어나게 되므로 검사하는 연산이 낭비가 된다. 따라서 이 때에 한하여 break를 사용하여 다음 row로 넘어가게 하였다.

**2.8 산업 표준**

기본적으로 GNU 컴파일러의 gcc를 이용하여 ANSI C 표준을 준수하여 제작되었다. 사용된 gnu 컴파일러는 서강대학교 실습 서버 cspro.sogang.ac.kr에 있는 것을 사용하였으며 버전은 gcc version 4.6.3 Ubuntu/Linaro 4.6.3-1ubuntu5이다.

**3. 기 타**

**3.1 환경 구성**

ssh 접속 프로그램 PuTTY.exe를 사용하여 서강대학교 실습 서버인 cspro.sogang.ac.kr로 접속하여 프로젝트를 진행하였다. vim text editor를 사용하여 작성하고 gnu 컴파일러를 이용하여 컴파일 하였다. (gcc version 4.6.3 Ubuntu/Linaro 4.6.3-1ubuntu5)

**3.2 참고 사항**

C언어를 사용함에 있어 각종 문법에 대한 내용은 서강대학교 C 프로그래밍 3반의 강의 자료를 참고하였다.

또한 문제의 보다 확실한 이해를 위해 Wikipedia.org의 bin packing problem 항목을 부분적으로 읽어보았다.

**3.3 팀 구성**

1인 1팀으로 구성되었다.

팀 원: 전진우 (20151603), 서강대학교 컴퓨터공학과

**3.4 수행기간**

2015. 10. 28. ~ 11. 01. 자료 조사 및 구현 방법의 구상

2015. 11. 02. ~ 11. 07. 프로그램의 구현

2015. 11. 08. ~ 11. 08. 프로그램 보고서 작성