University of Central Lancashire
UCLan Cyprus

CO1406: Data Structures and Algorithms

Assignment

Date Issued: 25/10/2022
Hand in Date: 21/03/2023

IMPORTANT

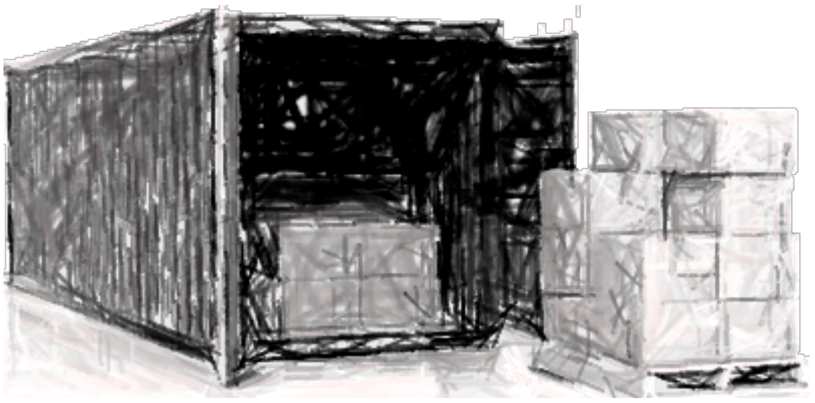
- As work is submitted on-line, **the deadline is midnight on the hand in date**.
- **Read the marking scheme carefully**.
- **This is an individual project** and no group work is permitted.

Learning Outcomes

- To develop problem-solving skills
- To evaluate common data structures
- To develop efficient algorithms to solve problems
- To enhance practical programming skills

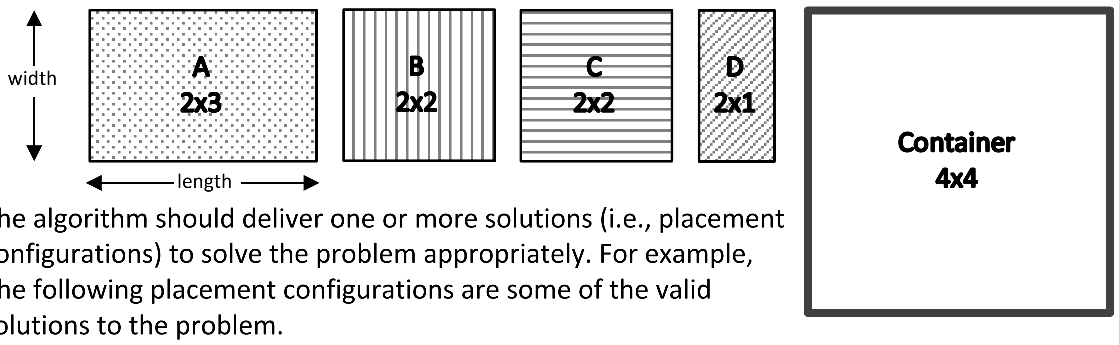
Assignment Purpose and Overview

The purpose of this assignment is to implement a container packing algorithm.

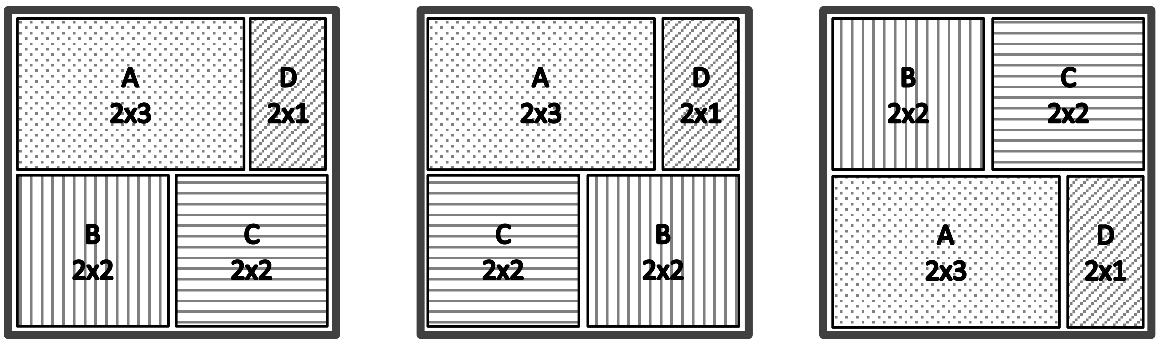


In this simplified version of container packing, we will consider a two-dimensional (2D) version of the packing problem, where given a set of boxes (i.e., rectangles) and a container (i.e., bounding box), we want to find one placement configuration that places all boxes within the container without overlap.

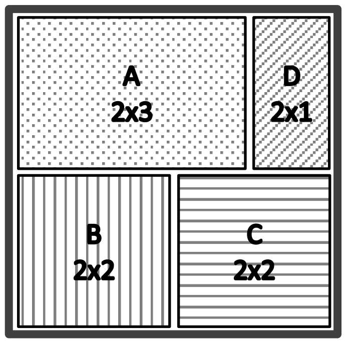
To facilitate our description, consider the following four boxes (illustrated with their respective width and length sizes (WxL), and a container of size 4x4:



The algorithm should deliver one or more solutions (i.e., placement configurations) to solve the problem appropriately. For example, The following placement configurations are some of the valid solutions to the problem.



In this assignment, the Container will be represented by a two-dimensional character array. For example, the figure below shows the first solution and how this should be represented in the container.



index	0	1	2	3
0	**A**	**A**	**A**	**D**
1	**A**	**A**	**A**	**D**
2	**B**	**B**	**C**	**C**
3	**B**	**B**	**C**	**C**

For the purposes of this assignment, a text file will be provided as input to allow your program to load box(es) and container configurations.

Overall, in this assignment, you are asked to:

- Evaluate different data structures for storing box configurations and implement the most efficient one.
- Develop a program for storing and manipulating container and box configurations and solutions (placement configurations).
- Design and develop algorithms that solve the placement configuration using backtracking, using simple and complex configurations.
- Design and implement appropriate data structures for supporting the algorithm's backtracking operation.
- Develop a program that orchestrates all the above to solve the container packing problem. Your program must also investigate if there are inconsistencies in the input file and if indeed there is at least one valid placement configuration.

Assignment Requirements and Constraints

The assignment starts by defining the Box and Packer Problem data structures to load box configurations and assign a size to the container. To this end, you will need to define the following:

- **Box** data structure: to store the width, length and name of the box
- **PackerProblem** data structure to store the width and length of the container, the number of boxes and all box configurations, and to print all box configurations.

To initialize the PackerProblem data structure you will use the following function, which will read a text file (named filename) and use dynamic memory allocation to create an instance of the PackerProblem data structure as described above:

```
PackerProblem* loadPackerProblem(string filename)
```

The structure of the input text file is described below, and an example is presented on the right:

- the first line includes two numbers, which indicate the size of the container (e.g., 4 4)
- the second number indicates the number of boxes (e.g., 4)
- the remaining lines represent the box configurations, which are triplets composed of two numbers and one character (e.g., 2 3 A), which represent the width, height, and name of the box respectively. In the example, you can observe that 4 box configurations have been provided.

```
4 4
4
2 3 A
2 2 B
2 2 C
2 1 D
```

When the file has been read properly (see section Error Control), you should then print using the console the container configuration and boxes, as illustrated below (following the above example).

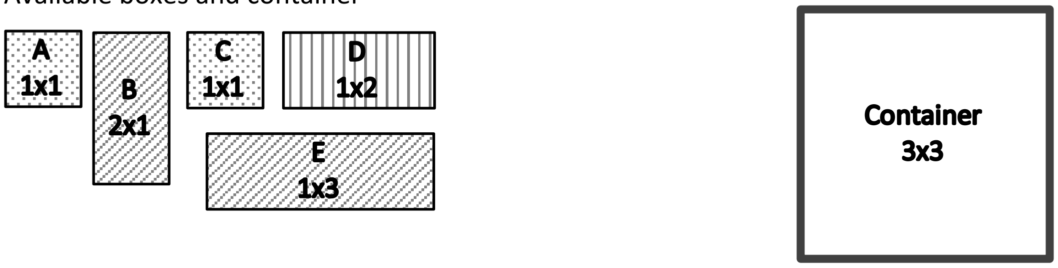
```
Container Configuration (4x4)
+----+
|XXXX|
|XXXX|
|XXXX|
|XXXX|
+----+

4 Boxes to be placed
A (2x3)
AAA
AAA

B (2x2)
BB
BB

C (2x2)
CC
CC

D (2x1)
D
D
```

A number of input files are provided in **Appendix 1** for you to test your implementation.

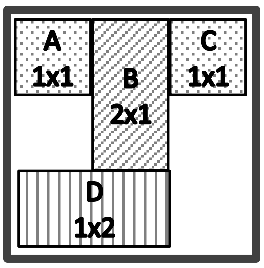
The program should initialize a PackerSolver data structure with the loaded configuration (i.e., PackerProblem) and call the solveProblem method in order to proceed to find a valid placement for all boxes, if it exists. The algorithm must employ a backtracking approach according to the following high-level concept:

- Start placing boxes in the configuration in a manner of your choice (e.g., sequentially, randomly).
- As soon as you reach a state where you cannot place any of the remaining boxes, the algorithm should backtrack to a previous state and try a different placement route. See below an example of backtracking to better understand the assignment concept.

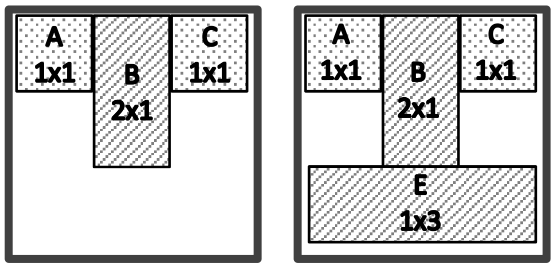
Available boxes and container



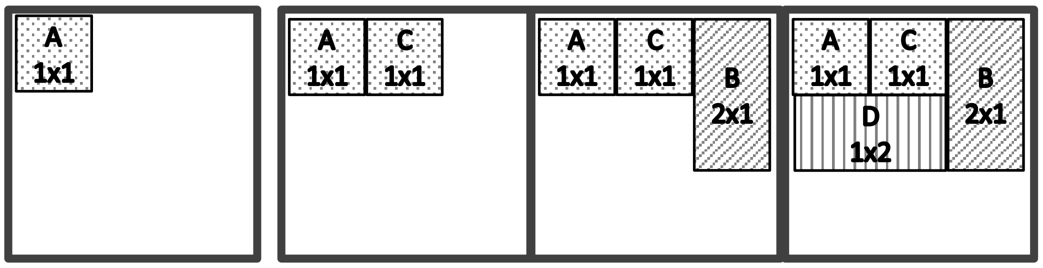
Sequential Placement will place Box A, then Box B, Box C and D and will not have place for E.



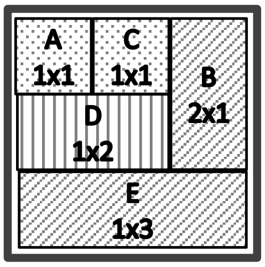
Depending on your backtracking strategy, one can select to make the last step before the dead-end (i.e., placing Box D) invalid and proceed with a different configuration.



This of course leads us to a new dead-end, which means we need to back-track again to the previous valid configuration and back-track even more. In the presented example, we will need to go back to where we placed the first box (i.e., Box A) and proceed with a different Box (i.e., Box C).



The resulting valid configuration should be exported to the user in a format similar to the printing of the initial empty container. Below we see an example of a valid configuration and the required console printing.



```
+---+
|ACB|
|DDB|
|EEE|
+---+
```

If your algorithm cannot find a solution, then the program should print that there is no solution.

It is compulsory that you implement a Stack data structure to enable the backtracking functionality. The Stack data structure must be designed and implemented using dynamic memory allocation and generic programming techniques and should implement the following functions:

- isEmpty: checks if the stack is empty
- push: inserts an element to the top of the stack
- pop: removes an element from the top of the stack, if it exists, without returning it
- top: returns the element located on the top of the stack, if it exists

Error control

In your assignment, you will also need to implement appropriate controls to ensure that the input is correct, and the problem presented is valid. In particular, you will need to check for the following:

- The structure of the file is correct. For example, the first two number should correspond to valid width and length of the container. If you identify that there are letters (e.g., 'a') or inconsistent numbers (e.g., 0 or -1), you should stop processing and report an error.
- The number of box configurations should be equal to the number of the boxes read in the file. For example, if you read from the file that there are 5 box configurations then you should report an error and stop processing when there are less.
- If there is no solution to the problem, you should report an error that there is no solution.

Time Performance

You will need to time your solver algorithm (not the loading of the file) using the `high_resolution_clock` library taught during the labs. The total time required for the solution to be printed should be printed at after the discovered solution.

IMPORTANT: The program must satisfy all the requirements and constraints described in this section. However, the requirements may not be sufficiently defined. During the specifications, you will need to record your assumptions and how these have influenced your models.

Extra Achievements

If you want to further enhance your skills, consider the below additional tasks:

- There are multiple valid configurations that can be discovered from an initial box list. If you find all valid configuration then you will receive an additional bonus 5 marks.
- There are multiple heuristics that can be used to speed-up the solver operation. If you research, understand, document and apply at least two strategies to solve the assignment and compare their efficiency wrt. time, you will receive an additional 5 marks.
- There are scenarios where you cannot find a valid configuration unless you rotate the boxes appropriately. If you enhance your solution to test rotated boxes and you can find valid configurations, you will receive an additional bonus 5 marks.

Deliverables

You should deliver a compressed folder, named as "202223.CO1406.<GNumber>.zip" (e.g., 202223.CO1406.G1234567.zip), with the following files:

1. 1x Document (Microsoft Word format) consisting of the documentation of parts A-D
2. 1x compressed file (.zip) including your source code (C++ Header files (.h) developed for part B and C++ Source files (.cpp) developed for parts C and D) and any resources (e.g., the input files) used in your implementation.
3. 1x C++ file containing all source code (including libraries).

Failure to comply to the above deliverable specification results in a grade of 0.

Grading Criteria

Marks will be awarded based on the following criteria. Within each part, aim to complete the work for each section before moving on to the next as you will not get the full credit for later sections if there are significant defects in an earlier section. **However, do not simply stop if you are stuck on one part, ask for feedback.**

In assessing the work within a section, factors such as simplicity, quality and appropriateness of comments, and quality and completeness of the design will be considered.

Part	Description	Range	Deliverable
A	Documentation	0-10	• Documentation in Microsoft Word (.docx) format
B	Data Structures	0-20	• Implementation of efficient Data Structures • C++ code for supporting data structures required for backtracking • C++ code for loading and manipulating Packer problems and solutions
C	Algorithm	0-20	• C++ code for solving the container packing problem using backtracking
D	Program	0-10	• C++ code using the deliverables of parts B and C to load and solve container problems, check for errors and timing your application

Detailed Marking Scheme

Part	Description	Criteria
A	Documentation	0-4 **Requirements Analysis** • 0 - no attempt or Fail • 1 - Requirements are poorly described • 2 - Requirements are sufficiently described. Develops requirements for a decent solution • 3 - Provides evidence of investigative skills in problem analysis to develop requirements for an efficient solution • 4 - Identifies several potential solutions and justifies the selection of the most efficient solution. 0-4 **Complexity Analysis** • 0 - no attempt or Fail • 1-4 Evaluates the time complexity of the loadPackerProblem and solveProblem functions correctly 0-2 **Source code Documentation** • 1 - Function comments, which describe the functionality of a method/field before the function definition • 1 - Inline comments, which describe implementation decisions within a method body
B	Data Structures	0-2 Implementation of the **Box** data structure 0-8 Implementation of the **PackerProblem** data structure. Implements appropriate members for storing: • 1 - number of rows and columns in the container • 1 - the number of boxes • 4 - box configurations • 2 - function to print the box configurations and the container 0-4 Implementation of the **PackerSolver** data structure representing a PackerProblem solution. Implements appropriate members for storing: • 1 - A PackerProblem instance as this was defined above • 2 - the final solution • 1 - the number of steps required by the solver to solve the solution 0-4 Implementation of a **Stack** data structure for storing moves (i.e., positions) during backtracking • 1 - Development of an appropriate push method • 1 - Development of an appropriate pop method • 1 - Development of an appropriate top method • 1 - Application of Generic programming 0-2 Demonstration of selecting **Storage Efficient** data members for the data structures described above. • 0 - no attempt

		• 1 - Consideration of trivial or infeasible alternatives • 2 - Careful consideration of alternatives and justification of selection of data member types
C	Algorithm	0-10 **Helper Functions** to support the solver algorithm • 2 – checks if a box can be placed in a position • 2 – places a box in a position • 2 – removes a box from a position • 2 – retrieves the next position after a successful placement • 2 – prints a placement configuration in the desired format 0-10 **Backtracking Algorithm** to solve container packing problems • 8 – utilizes backtracking to find a correct solution to the problem • 2 – prints the solution when there is one including the number of steps and the time
D	Program	0-8 **PackerProblem Loading Function** • 0-6 initializes a PackerProblem data structure appropriately using the loadPackerProblem function • 0-2 checks for inconsistencies/errors in the file 0-2 **Main function** • 1 – Loads the input file into the PackerProblem data structure • 1 - Initializes a PackerSolver data structure, loads the PackerProblem and then invokes the solver

Submission of assignment work

- Anonymous marking is being used. Apart from your University ID number ("G2…"), avoid including anything that would allow you to be identified from your work.
- *Keep a <u>complete</u> copy of the work you hand in.*
- Avoid submitting work at the last minute, but if there is a technical problem uploading to Blackboard, email the zip file to me before the deadline and upload the work when Blackboard is available.

Conformance to Academic Regulations

Students and their assessment are subject to the academic regulations, which specifies what is considered as Unfair Means to Enhance Performance: Cheating, Plagiarism and Collusion and Commissioning of Assessed Work. Students are expected to conform to these regulations, or face disciplinary actions, as described in Appendix 9 of the Academic Regulations (the relevant excerpts clarifying the unfair means are listed below):

Cheating

The term cheating includes, inter alia:

a. Being in possession of notes, 'crib notes', or texts books during an examination other than an examination where the rubric permits such usage;
b. Copying from another candidate's script or work;
c. Communicating during the examination with another candidate;
d. Having prior access to the examination questions unless permitted to do so by the rubric of the examination;
e. Substitution of examination materials;
f. Unfair use of a pocket calculator;
g. Impersonation;
h. Use of a communication device during the examination;
i. Or any deliberate attempt to deceive.

Plagiarism

Material submitted for assessment through open book examination, coursework, project or dissertation must be the student's own efforts and must be his/her own work. Students are required to sign a declaration indicating that individual work submitted for assessment is their own.

Copying from the works of another person constitutes plagiarism, which is an examination offence. Brief quotations from the published or unpublished works of another person, suitably attributed, are acceptable. Every School issues guidelines on the use and referencing of quotations which students are required to follow.

Collusion

Collusion is an example of unfair means because, like plagiarism, it is an attempt to deceive the examiners by disguising the true authorship of an assignment, or part of an assignment. Its most common version is that student A copies, or imitates in close detail, student B's work with student B's consent. But it also includes cases in which two or more students divide the elements of an assignment among themselves, and copy, or imitate in close detail, one another s answers.

It is an unfair means offence to copy, or imitate in close detail, another student s work, even with his or her consent (in which case it becomes an offence of collusion). It is also an offence of collusion to consent to having one's work copied or imitated in close detail. Students are expected to take reasonable steps to safeguard their work from improper use by others. Where a student is found to have engaged in collusion, the same penalties as for plagiarism will apply.

Where it is established that student B has not engaged in plagiarism, the requirement for resubmission may be waived in the case of student B.

Collusion should not be confused with the normal situation in which students learn from one another, sharing ideas, as they generate the knowledge and understanding necessary for each of them successfully and independently undertake an assignment.

Nor should it be confused with group work on an assignment where this is specifically authorised in the assignment brief.

Commissioning of Assessed Work

Commissioning occurs where a student commissions a third party to complete all or part of an assessed piece of work and then submits it as their own. Commissioned work may be pre-written or specifically prepared for the student. It might be obtained from a company or an individual and may or may not involve a financial transaction. It includes the use of essay mills or buying work on-line or the use of a proof-reading service that includes re-writing the original assessed piece of work. Where it is suspected that a student has submitted work that has not been written by them, the student may be asked questions about the work during an interview with the Academic Integrity Officer or Academic Misconduct Committee to give them the opportunity to demonstrate appropriate knowledge of the subject matter and that they understand the content of the work. Students must keep copies of drafts and other materials used in researching and preparing the work.

Category 4: Gross Academic Misconduct

Category 4 will normally be defined as gross academic misconduct where a clear intent to deceive and gain an unfair academic advantage can be established. Examples of category 4 gross academic misconduct include, without limitation:

- A repeat instance of category 3 academic misconduct in any form
- Commissioning of assessed work
- Fabrication or falsification of data

Academic Penalties for Category 4 offense

Level failed and a requirement to withdraw from the programme. (This does not preclude the student from applying for re-admission to the University after a period of time defined by the Committee.)
Or
Expulsion from the University on a permanent basis. The Academic Misconduct Committee will advise the Assessment Board regarding the student's entitlement to any exit award or credit achieved. The student will normally be entitled to retain an exit award or any credits awarded for work that has already been passed without evidence of academic misconduct.
A flag will be placed on the student record system.

The complete set of academic regulations can be accessed online at:
http://www.uclan.ac.uk/study_here/student-contract-taught-programmes.php

Reassessment and Revision

Reassessment in written examinations and coursework is at the discretion of the Course Assessment Board and is dealt with in accordance with University policy and procedures.

Appendix 1

Example file 1

```
4 4
4
2 3 A
2 2 B
2 2 C
2 1 D
```

Example file 2

```
3 3
5
1 1 A
2 1 B
1 1 C
1 2 D
1 3 E
```

Example file 3

```
3 3
5
1 1 A
2 1 B
1 1 C
2 1 D
3 1 E
```

Example file 4

```
4 4
7
1 1 A
1 1 B
1 1 C
1 1 D
2 2 E
2 2 F
2 2 G
```

Example file 5

```
4 4
7
1 1 A
1 1 B
1 1 C
1 1 D
2 2 E
2 2 F
2 2 G
```

Example file 6

```
10 10
12
3 3 !
1 1 @
5 5 #
4 4 $
2 2 %
5 5 ^
3 3 &
1 1 *
2 2 (
2 2 )
1 1 -
1 1 +
```

Example file 7

```
5 5
4
2 4 A
1 2 B
3 3 C
2 3 D
```

Example file 8

```
20 20
25
4 4 1
3 3 2
1 1 3
7 7 4
2 2 5
3 3 6
5 5 7
6 6 8
4 4 9
4 4 !
2 2 @
4 4 #
3 3 $
1 1 %
7 7 ^
2 2 &
3 3 &
5 5 *
6 6 (
4 4 )
4 4 _
2 2 +
5 5 /
2 2 \
1 1 -
```