資料結構與演算法 I
(Data Structure and Algorithms I)
2025 fall

2025/09/26

Abstract Data Type & Chapter 2

- 第一次小考
  - 下週五(10/3), 9:10-10:00上課, 10:10-11:40 考試
  - 考試地點：管理大樓B0204/另外會借一間教室
  - 範圍：開學到今日上課結束範圍

2.1.1 C++ Class

- C++ provides the *class* to support the distinction between specification and implementation and to hide the implementation of the ADT from it users.
- The C++ class consists of four components:
  - A class name
  - Data numbers
    - i. j, k
  - Member functions
    - Get(); Set();
  - Levels of program access
    - public
    - private
    - protected

```
#ifndef RECTANGLE_H
#define RECTANGLE_H
    class Rectangle{
    public:
            Rectangle();
           ~Rectangle();
            int GetHeight();
            int GetWidth();
    private:
    int xLow, yLow, height, width;};
#endif
```

公用

私用

- public
  - any public data member (member function) can be accessed (invoked) from anywhere in the program
- private
  - a private data member (member function) can be only accessed (invoked) from within its class or by a function or a class that is declared to be a friend.
- protected
  - A protected data member (member function) can only be accessed (invoked) from within its class or its subclasses or by a friend.

```
// In the source file Rectangle.cpp
#include "Rectangle.h"

// 用"Rectangle::"來指明GetHeight()與GetWidth()是Rectangle的成員函式
//成員函式是在它們所屬類別定義的外部實作的

int Rectangle::GetHeight() {return height;}
int Rectangle::GetWidth() {return width;}
```

```
// In a source file main.cpp
#include <iostream>
#include "Rectangle.h"
main() {
        Rectangle r, s;       // 物件 r跟 s屬於類別Rectangle
        Rectangle *t = &s;  // t 是類別物件s的指標
        // 使用 "." 來存取類別物件的成員。
        // 使用 "->" 透過指標來存取類別物件的成員。
  If ( r.GetHigh ( ) * r.GetWidth ( ) > t→GetHeight ( ) * t→GetWidth ( ) )
          cout << "r";
  else cout << "s";
  cout << "has the greater area " << endl;
}
```

2.1.4 Special Class Operations

- Constructors
  - a member function which initialize data members of an object
  - if it is provided for a class, it is automatically executed when an object of that class is created.
  - If not, memory is allocated for the data members of a class object, but the data member are not initialized Program 2.1
- Destructors
  - Destructors are automatically invoked when a class object goes out of scope or when a class object is deleted
- Operator overloading
  - Program 2.6, 2.7

2.1.5 Miscellaneous Topics

- In C++, a *struct* is identical to a *class*, except that the default level of access is public
- A *union* is a structure that reserves storage for the largest of its data members so that only one of its data member can be stored, at any time.

C++ Gossip: Union

union是一種特殊的類別，使用關鍵字union來定義，union維護足夠的空間來置放多個資料成員中的「一種」，而不是為每一個資料成員配置空間， 在union中所有的資料成員共用一個空間，同時間只能儲存其中一個成員的資料，一個定義union的例子如下：

```
union StateMachine {
    char character;
    int number;
    char *str;
    double exp;
};
```

一個union只配置一個足夠大的空間以來容納最大長度的資料成員，以上例而言，最大長度是double型態，所以StateMachine的記憶體空間 就是double型態的長度，union的成員預設為public，也可以宣告為protected或private，當中可以定義建構函式、解構函式與 成員函式，例如：

http://openhome.cc/Gossip/CppGossip/Union.html

C++

```cpp
// declaring_a_union.cpp
union RecordType    // Declare a simple union type
{
    char   ch;
    int    i;
    long   l;
    float  f;
    double d;
    int *int_ptr;
};

int main()
{
    RecordType t;
    t.i = 5; // t holds an int
    t.f = 7.25; // t now holds a float
}
```

https://learn.microsoft.com/zh-tw/cpp/cpp/unions?view=msvc-170

2.1.6 ADT & C++ class

class *NaturalNumber* { // 從零開始到電腦上允許的最大整數（MAXINT）為止所形成的一串有順序的整數子範圍

pubic:

NaturalNumber Zero( );// 回傳0

bool *IsZero*();// 如果***this**是0則回傳**true**，否則回傳**false**

NaturalNumber Add(*NaturalNumber y*);
// 回傳 ***this**+*y*與MAXINT中較小的一個

bool *Equal*(*NaturalNumber y*);
// 如果 ***this** = = *y*回傳**true**，否則回傳**false**

NaturalNumber Successor();
// 如果 ***this**是MAXINT則回傳MAXINT，否則回傳 ***this**+1

NaturalNumber Subtract(*NaturalNumber y*);
// 如果 ***this** < *y*則回傳0，否則回傳 ***this**-*y*

};

2.2 ADT of Array

- ADT Array is
  - An array只是 a consecutive set of memory locations嗎?
  - A set of pairs <*index, value*> where for each value of *index* there is a *value* from the set *item*. *Index* is a finite ordered set of one or more dimensions
  - In mathematical terms, we call this a "*correspondence*" or "*mapping*"
  - Functions:
    - *Array* Create(j, list, iniValue)
    - *Item* Retrieve(i)
    - *Array* Store(i, x)

```
class GeneralArray {
/*由許多數值對 <index, value> 所乘的集合，其中對於每一個index ∈ IndexSet都有一個float 型態的value與其對應。IndexSet是一個有序且有限的一維或多維集合。例如，一維集合 {0, …, n-1}，二維集合{(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)}等。*/
public:
        GeneralArray(int j, RangeList list, float initValue = defaultValue);
  // 這個建構子會建立一個j維的浮點數陣列；第k維的範圍由list的第k個元素來決定。
  // 對於每一個在索引集合裡的索引 i，插入 <i, initValue> 到這個陣列裡。

        float Retrieve(index i);
       // 如果i屬於這個陣列的索引集，那麼回傳在這個陣列中相對應於i的浮點數；
       // 否則，丟出一個例外。

        void Store(index i, float x);
        // 如果i屬於這個陣列的索引集，那麼把跟i相對應的舊值換成x；
        // 否則，丟出一個例外。
}; //
```

GeneralArray is more general than the C++ Array as it is more flexible about the composition of the index set.

1. C++ Array requires the index set be a set of consecutive integers starting from 0;
2. C++ does not check an array index to ensure that it belongs to the range for which the array is defined.

2.3 The Polynomial ADT

- Array are not only data structure in their own right, we can also use them to implement other abstract data type
  - Ex: ordered list, or linear, list
  - Examples:
    - Days of the week
    - Values in a deck of card (Ace, 1,2,3,…, King)
    - Years in World War II (1941, 1942, 1943, 1944)
  - Perhaps the most common way to represent an ordered list is by an array where we associate the list element a_i with the array index i

We can perform many operations on lists, including:

(1) Find the length, n, of the list

(2) Read the list from left to right (or right to left)

(3) Retrieve the ith element, 0<=i<=n

(4) Store a new value into the ith position

(5) Insert a new element at the position i, causing element numbered i,i+1, i+2…to become i+1,i+2, i+3…

(6) Delete the element at position i, causing element numbered i+1, i+2…to become i,i+1, …

- Step (1)~(4)→ ***sequential mapping is OK***
- Step(5)(6) require real effort→ ***nonsequential mapping of ordered list would be better***

Polynomial

- $a(x)=3x^2+2x-4$, $b(x)=x^8-10x^5-3x^3+1$

$$A(x) = a_n x^n + \cdots + a_0 = \sum a_i x^i$$

$$A(x) + B(x) = \sum (a_i + b_i) x^i$$

$$A(x) \bullet B(x) = \sum a_i X^i \bullet \sum (b_i x^j))$$

- $3x^2 \rightarrow (3,2)$; $2x \rightarrow (2,1)$; $-4 \rightarrow (-4,0)$
- *The largest exponent of a polynomial is called its degree*
- *Coefficients that are zero are not displayed*
- *Examples*
  - $A(x) = 2x^{1000} + 1$, *degree* $= 1000$
  - $B(x) = x^4 + 10x^3 + 3x^2 + 1$, *degree* $= 4$

ADT Polynomial

```
class Polynomial {
// p(x) = ； 一個 <e_i， a_i> 的有序對之集合，
// 其中a_i是一個非零的 float 係數而是e_i一個非負的整數指數。
public:
        Polynomial( );
        // 建立多項式p(x) = 0。

        Polynomial Add(Polynomial poly);
        // 回傳 *this與poly兩個多項式之和。

        Polynomial Mult(Polynomial poly);
        // 回傳 *this與poly兩個多項式之積。

        float Eval(float f );
        // 求出當多項式 *this為 f時的值並且回傳它的結果。
};
```

Polynomial Representations 1

- **Private**
  - int degree;   // degree ≤ MaxDegree
  - float coef [MaxDegree + 1];

$a_{n-i} * x^{n-i}$

i=0			i=n
a_n		--------------	a_0

- **Examples**
  - a.degree = n
  - a.coef[i] = a_{n-i}, $0 \leqq i \leqq n$
- **Wasteful computer memory**

Polynomial Representations 2

```
private:
    int degree;
    float *coef;

    Polynomial::Polynomial(int d)
{
    degree = d;
    coef = new float [degree+1];
}
```

實際需要多少,宣告多少
不用MaxDegree

Polynomial Representations 3

```
class Polynomial;  // forward delcaration

class term {
    friend Polynomial;
    private:
        float coef;         // coefficient
        int exp;            // exponent
};

private:
    static term termArray[MaxTerms];
    static int free;
    int Start, Finish;

term Polynomial:: termArray[MaxTerms];
int Polynomial::free = 0;   // location of next free location in temArray
```

前置宣告:
http://blog.csdn.net/spritelw/article/details/965702
不要在.h檔裡include了一堆其他的.h檔, 好處是當其.h修改時, 並不會影響到太多的cpp須要重新compile, 也加快了build project的速度, 其最大的好處是, .h檔不會太雜亂.

For Sparse Polynomial

$A(x) = 2x^{1000} + 1$, 有很多項的係數為0

Figure 2.1 Array Representation of two Polynomials

Represent the following two polynomials:

$$A(x) = \boxed{2x^{1000} + 1}$$

$$B(x) = \boxed{x^4 + 10x^3 + 3x^2 + 1}$$

下一個可用
空間的位置

	a.start	a.finish	b.start			b.finish	free
coef	2	1	1	10	3	1	
exp	1000	0	4	3	2	0	
	0	1	2	3	4	5	6

對於一個有n個非零項的多項式a
a.finish=a.start+n-1

Is the representation 3 good ????
Depend on the number of nonzero

2.3.2 Polynomial Addition

- Textbook: page 91-92
- Try to figure out Program 2.8

- If you have any question, please discuss with me during the office hours.

2.4 Sparse Matrices

A general Matrix consists of **m** rows & **n** columns of numbers
→m ×n

$$
\begin{array}{c|c}
 & \begin{array}{ccc} \text{行0} & \text{行1} & \text{行2} \end{array} \\
\begin{array}{c} \text{列0} \\ \text{列1} \\ \text{列2} \\ \text{列3} \\ \text{列4} \end{array} &
\begin{bmatrix} -27 & 3 & 4 \\ 6 & 82 & -2 \\ 109 & -64 & 11 \\ 12 & 8 & 9 \\ 48 & 27 & 47 \end{bmatrix}
\end{array}
$$

(a) a[5][3]

$$
\begin{array}{c|c}
 & \begin{array}{cccccc} \text{行0} & \text{行1} & \text{行2} & \text{行3} & \text{行4} & \text{行5} \end{array} \\
\begin{array}{c} \text{列0} \\ \text{列1} \\ \text{列2} \\ \text{列3} \\ \text{列4} \\ \text{列5} \end{array} &
\begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{bmatrix}
\end{array}
$$

(b) b[6][6]

Only eight out of 36 possible elements are nonzero, and that is sparse (稀疏)
No precise definition of when a matrix is sparse!
Question:
有一個5000 * 5000矩陣(25,000,000個元素), 只有500個有值, 電腦無法一次產生5000* 5000矩陣,我們該如何處理?

2.4.2 Sparse Matrix Representation

- Use triple ***<row, column, value>***
- An array of triples to represent a sparse matrix
- These triples be stored by rows with triples for the first row first, followed by those of the second row, and so on.

	列	行	值
smArray[0]	0	0	15
[1]	0	3	22
[2]	0	5	-15
[3]	1	1	11
[4]	1	2	3
[5]	2	3	-6
[6]	4	0	91
[7]	5	2	28

(a)

	列	行	值
smArray[0]	0	0	15
[1]	0	4	91
[2]	1	1	11
[3]	2	1	3
[4]	2	5	28
[5]	3	0	22
[6]	3	2	-6
[7]	5	0	-15

(b)

Figure 2.3 sparse matrix and its transpose stored as triples

2.4.2 Sparse Matrix Representation

	行 0	行 1	行 2	行 3	行 4	行 5
列 0	15	0	0	22	0	-15
列 1	0	11	3	0	0	0
列 2	0	0	0	-6	0	0
列 3	0	0	0	0	0	0
列 4	91	0	0	0	0	0
列 5	0	0	28	0	0	0

Use triple ***<row, column, value>***

使用三元表示法

	列	行	值
smArray[0]	0	0	15
[1]	0	3	22
[2]	0	5	-15
[3]	1	1	11
[4]	1	2	3
[5]	2	3	-6
[6]	4	0	91
[7]	5	2	28

(a)

轉置矩陣

	列	行	值
smArray[0]	0	0	15
[1]	0	4	91
[2]	1	1	11
[3]	2	1	3
[4]	2	5	28
[5]	3	0	22
[6]	3	2	-6
[7]	5	0	-15

(b)

Figure 2.3 sparse matrix and its transpose stored as triples

Exercise

3	**0**	**2**	**0**
0	5	7	0
0	0	0	4
1	6	0	0

Use triple to express this matrix

Transpose

2.4.3 Transposing a Matrix

For (each row i)
Store (i,j,value) of the original matrix as (j,i,value) of the transpose;
(0, 0, 15) which becomes (0, 0, 15)
(0, 3, 22) which becomes (3, 0, 22)
(5, 0, -15) which becomes (0, 5, -15)
(1, 1, 11) which becomes (1, 1, 11) ←要先被放於轉置矩陣中
可是必須要轉完才知道它應該要被放在哪裡

For (for all elements in column j)
Store (i,j,value) of the original matrix as (j,i,value) of the transpose;

	列	行	值
smArray[0]	0	0	15
[1]	0	3	22
[2]	0	5	-15
[3]	1	1	11
[4]	1	2	3
[5]	2	3	-6
[6]	4	0	91
[7]	5	2	28

(a)

	列	行	值
smArray[0]	0	0	15
[1]	0	4	91
[2]	1	1	11
[3]	2	1	3
[4]	2	5	28
[5]	3	0	22
[6]	3	2	-6
[7]	5	0	-15

(b)

Program 2.10 Transposing a Matrix

```
SparseMatrix SparseMatrix::Transpose()
// return the transpose of  *this
{
     SparseMatrix b(cols, rows, terms);//capacity of b.smArray is terms
     {
              int CurrentB = 0;
              for (int c = 0; c < cols; c++)   // transpose by columns
                   for (int i = 0; i < terms; i++)
                   // find elements in column c
                             if (smArray[i].col == c) {
                               b.smArray[CurrentB].row = c;
                               b.smArray[CurrentB].col = smArray[i].row;
                               b.smArray[CurrentB].value = smArray[i].value;
                               CurrentB++;
                             }
     }  // end of if (Terms > 0)
      return b;
}  // end of transpose
```

$\mathbf{O(columns * terms)}$

$\mathbf{=O(columns^{2} * Rows)}$

Fast Matrix Transpose

- The O(terms*columns) time => O(rows*columns^2) when terms is the order of (rows*columns)
  - Save space, waste time!

```
for(int i=0;i<row;i++)
 for(int j=0;j<column;j++)
          b[j][i]=a[i][j];
```

→ O(rows*columns)

- A better transpose function in Program 2.11.
  - It first computes how many terms in each columns of matrix a before transposing to matrix b.
  - Then it determines where is the starting point of each row for matrix b.
  - Finally it moves each term from a to b.

	列	行	值
smArray[0]	0	0	15
[1]	0	3	22
[2]	0	5	-15
[3]	1	1	11
[4]	1	2	3
[5]	2	3	-6
[6]	4	0	91
[7]	5	2	28

(a)

	列	行	值
smArray[0]	0	0	15
[1]	0	4	91
[2]	1	1	11
[3]	2	1	3
[4]	2	5	28
[5]	3	0	22
[6]	3	2	-6
[7]	5	0	-15

(b)

a行或b列	[0]	[1]	[2]	[3]	[4]	[5]
RowSize =	2	1	2	2	0	1
RowStart =	0	2	3	5	7	7

for(i=0;i<terms;i++)
RowSize[smArray[i].col]++;

for(i=1;i<cols;i++)
RowStart [i] = RowSize[i-1]+ RowStart[i-1]

```
SparseMatrix SparseMatrix::FastTranspose( )
{// 在 O(terms + cols)的時間內回傳 *this 的轉置矩陣
  SparseMatrix b(cols , rows , terms);
  if (terms > 0)
  {// 非零的矩陣
    int *rowSize = new int[cols];
    int *rowStart = new int[cols];
    // 計算 rowSize[i] = b 的第 i 列之項數
    fill(rowSize, rowSize + cols, 0);   // 初始化
    for (int i = 0 ; i < terms ; i ++) rowSize[smArray[i].col]++;

    // rowStart[i] = b 的第 i 列之起始位置
    rowStart[0] = 0;
    for (int i = 1 ; i < cols ; i++) rowStart[i] = rowStart[i-1] + rowSize[i-1];

    for (int i = 0 ; i < terms ; i++)
    {// 從 *this 複製到 b
      int j = rowStart[smArray[i].col];
      b.smArray[j].row= smArray[i].col;
      b.smArray[j].col = smArray[i].row;
      b.smArray[j].value = smArray[i].value;
      rowStart[smArray[i].col]++;
    } // for 結束
    delete [] rowSize;
    delete [] rowStart;
  } // if 結束
  return b;
}
```

O(columns)

O(terms)

O(columns-1)

O(terms)

O(terms+columns)

O(row * column)

2.4.4 Matrix Multiplication

- Definition: Given A and B, where A is m*n and B is n * p, the product matrix Result has dimension m*p. Its [i][j] element is

$$d_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj}$$

for $0 \le i < m$ and $0 \le j < p$.

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

已知 $A = \begin{bmatrix} 1 & 4 \\ 2 & 3 \\ 3 & 2 \end{bmatrix}$、$B = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 0 & -1 \end{bmatrix}$，則：

$$A \times B = \begin{bmatrix} 1\times1+4\times2 & 1\times2+4\times0 & 1\times3+4\times(-1) \\ 2\times1+3\times2 & 2\times2+3\times0 & 2\times3+3\times(-1) \\ 3\times1+2\times2 & 3\times2+2\times0 & 3\times3+2\times(-1) \end{bmatrix} = \begin{bmatrix} 9 & 2 & -1 \\ 8 & 4 & 3 \\ 7 & 6 & 7 \end{bmatrix}$$

$$B \times A = \begin{bmatrix} 1\times1+2\times2+3\times3 & 1\times4+2\times3+3\times2 \\ 2\times1+0\times2+(-1)\times3 & 2\times4+0\times3+(-1)\times2 \end{bmatrix} = \begin{bmatrix} 14 & 16 \\ -1 & 6 \end{bmatrix}$$

2.5 Representation of Arrays

- **Multidimensional arrays** are usually implemented by **one dimensional array** via either row major order or column major order.
- It can be retrieved efficiently. And be able to retrieve array element easily

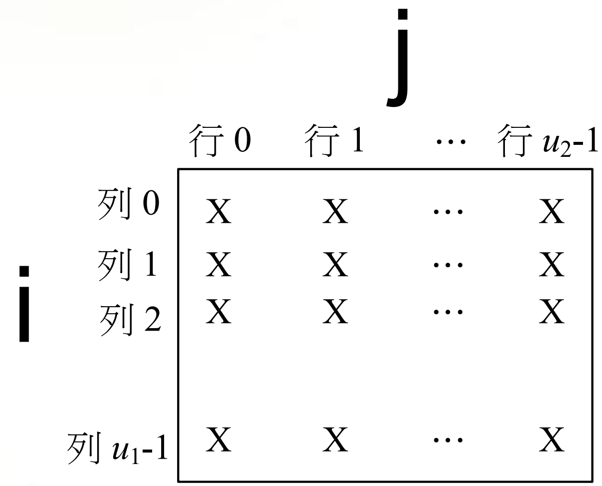
$a[u_1][u_2][u_3]\ldots[u_n] \rightarrow$ The number of element is

$$\prod_{i=1}^{n} u_i$$

Row major order

a[2][3][2][2]→ 2*3*2*2=24 elements
a[0][0][0][0] a[0][0][0][1] a[0][0][1][0] a[0][0][1][1]……………
The index at the right moves the fastest

Sequential representation of a[u_1][u_2]

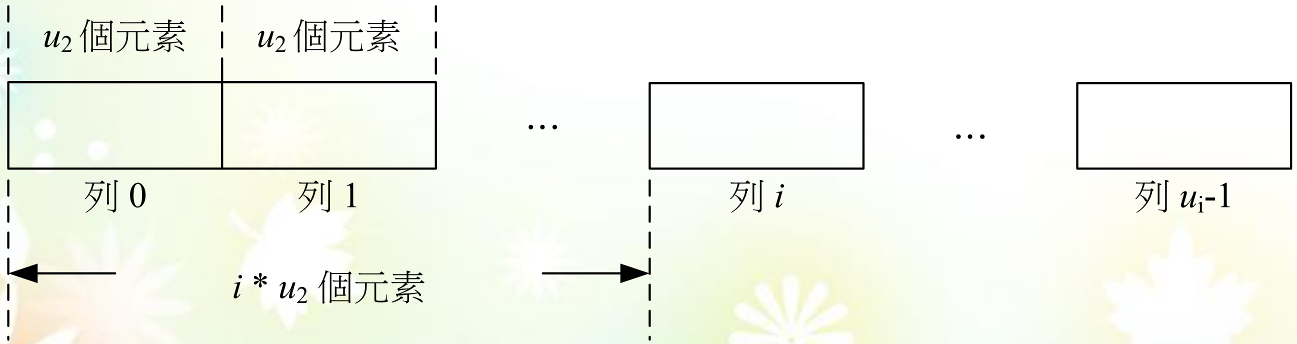


(a)

If alpha is the address of a[0][0]

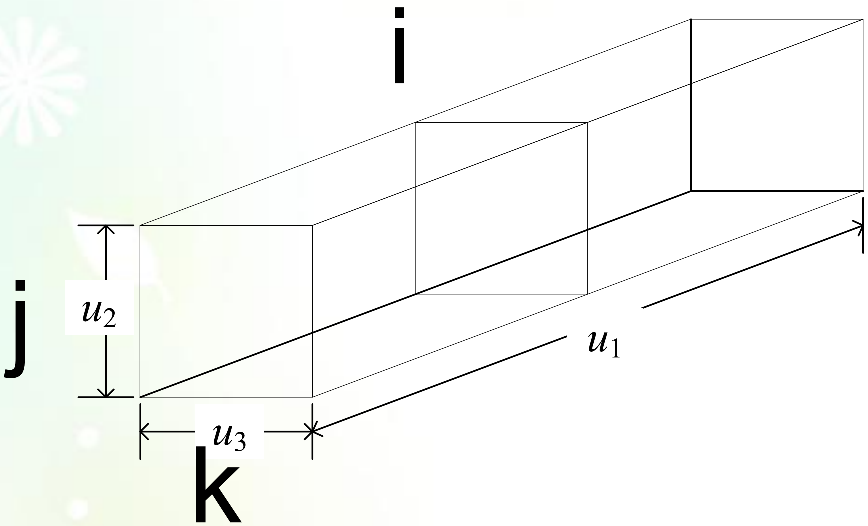
The address of a[i][0] is alpha+i*u2

The address of a[i][j] is alpha+i*u2+j



(b)

Sequential representation of a[u_1][u_2][u_3]



If alpha is the address of a[0][0][0]

The address of a[i][0][0] is alpha+i*u2*u3

The address of a[i][j][0] is alpha+i*u2*u3+j*u3

The address of a[i][j][k] is alpha+i*u2*u3+j*u3+k