الدرس الرابع: المصفوفات

Arrays

1- تعريف المصفوفات

بشكل عام، المصفوفة هي مجموعة من العناصر المرتبة والمنظمة بطريقة معينة كأن تكون على شكل قائمة، جدول ثلاثي الأبعاد، ...الخ.

في لغات البرمجة، المصفوفة هي مجموعة من الكائنات التي لها نفس الصفات والتي يمكن عنونتها واحدة تلو الأخرى.

لإنشاء مصفوفة في لغة C++، يجب استخدام الصيغة التالية:

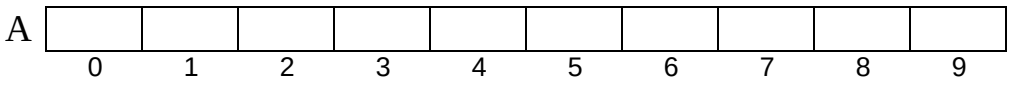
```
Array_Type  Array_Name[Array_size];
```

حيث أن Array_Type يمثل نوع العناصر، Array_Name يمثل اسم المصفوفة و Array_size يمثل عدد عناصر المصفوفة وهو قيمة ثابتة.

على سبيل المثال، يقوم الأمر:

int A[10];

بإنشاء مصفوفة اسمها A تتكون من 10 عناصر من فئة الأعداد الصحيحة مرقمة من 0 إلى 9.



يمكن الوصول على أي عنصر في المصفوفة باستخدام الصيغة التالية:

```
Array_Name[Index]
```

حيث أن Index يمثل دليل أو رقم العنصر في المصفوفة.

تبدأ عناصر المصفوفة من الرقم 0 وبالتالي إذا كانت المصفوفة A تحتوي على عدد n عنصر فإن:

- A[0] يعني العنصر الأول
- A[1] يعني العنصر الثاني
- ...
- A[n-1] يعني العنصر الأخير.

مثال رقم 1

أكتب برنامجا بلغة C++ يقوم بقراءة 5 أعداد صحيحة وتخزينها في مصفوفة اسمها A ومن ثم طباعتها على الشاشة.

- **البرنامج**

```
#include <iostream>
using namespace std;
int main( )
{
      int  A[5];    //  الاعلان عن المصفوفة
      int  i;

      // قراءة عناصر المصفوفة من لوحة المفاتيح
      for(i = 0; i <= 4; i++)
      {
            cout<<"Enter an integer :";
            cin>>A[i];
      }

      // طباعة عناصر المصفوفة على الشاشة
      cout<<endl;
      for(i = 0; i <= 4; i++)
            cout<<"A["<<i<<"] = "<<A[i]<<"\t";
      cout<<endl;
      return 0;
}
```

- **تتبع تنفيذ البرنامج**

```
Enter an integer :5
Enter an integer :15
Enter an integer :10
Enter an integer :20
Enter an integer :25

A[0] = 5      A[1] = 15      A[2] = 10      A[3] = 20      A[4] = 25

Press any key to continue . . .
```

مثال رقم 2

أكتب برنامجا يقوم بقراءة درجات 15 طالبا في مادة البرمجة وتخزينها في مصفوفة اسمها Mark، ومن ثم يقوم بحساب متوسط الدرجات (average) وكم طالب حصل على درجة أكبر من المتوسط.

- **البرنامج**

```
#include <iostream>
using namespace std;

int main( )
{
      float Mark[15];  // التعريف بالمصفوفة
      float sum = 0;
      float average;
      int i, count;
```

```
	for (i=0; i <= 14; i++)		// قراءة الدرجات وتعبئة المصفوفة
	{
		cout<<"Enter mark["<<i<<"] : ";
		cin>>Mark[i];
		sum = sum + Mark[i];
	}
	average = sum / 15;			// حساب متوسط الدرجات
	cout<<"Average = "<<average<<endl;

	count = 0;
	for (i=0; i <= 14; i++)
	{
		if (Mark[i] > average)
			count++;
	}

	cout<<count<<" students have a mark greater than the average";
	cout<<endl;
	return 0;
}
```

- **تتبع تنفيذ البرنامج**

```
Enter mark[0] : 56
Enter mark[1] : 58
Enter mark[2] : 78
Enter mark[3] : 96
Enter mark[4] : 28
Enter mark[5] : 74
Enter mark[6] : 81
Enter mark[7] : 55
Enter mark[8] : 56
Enter mark[9] : 52
Enter mark[10] : 58
Enter mark[11] : 94
Enter mark[12] : 82
Enter mark[13] : 73
Enter mark[14] : 64

Average = 67

7 students have a mark greater than the average

Press any key to continue . . .
```

تمرين رقم 1

كيف سيكون محتوى المصفوفة F بعد تنفيذ الكود التالي ؟

```
int F[10];
F[0] = 1;
F[1] = 1;

for (int i=2; i<=9; i++)
{
	F[i] = F[i-2] + F[i-1];
}
```

الحل

F	1	1	2	3	5	8	13	21	34	55
	0	1	2	3	4	5	6	7	8	9

تمرين رقم 2

كيف سيكون محتوى المصفوتين A و B بعد تنفيذ الكود التالي ؟

```
int A[10] = {2, 4, 8, 16, 32, 64, 128, 256, 512, 1024};
int B[10];
int i;

for (i = 0; i<=9; i++)
      B[i] = A[9-i];
```

الحل

B	2	4	8	16	32	64	128	256	512	1024
	0	1	2	3	4	5	6	7	8	9

B	1024	512	256	128	64	32	16	8	4	2
	0	1	2	3	4	5	6	7	8	9

تمرين رقم 3

أكتب برنامجا بلغة C++ يقوم بحساب مجموع عناصر المصفوفة A و المصفوفة B ووضع الناتج في المصفوفة C.

مثال

A	1	2	3	4	5
	0	1	2	3	4

B	2	4	6	8	10
	0	1	2	3	4

C	3	6	9	12	15
	0	1	2	3	4

- **الحل**

```
#include <iostream>
using namespace std;
int main()
{
      int A[5] = {1, 2, 3, 4, 5};
      int B[5] = {2, 4, 6, 8, 10};
      int C[5];

      for (int i = 0; i<=4; i++)
            C[i] = A[i] + B[i];

      for(i=0; i<=4; i++)
            cout<<"C["<<i<<"] = "<<C[i]<<"\t";
      cout<<endl;
      return 0;
}
```

- **مخرجات البرنامج**

```
C[0] = 3      C[1] = 6      C[2] = 9      C[3] = 12      C[4] = 15
Press any key to continue . . .
```

2- البحث الخطي في المصفوفات (Linear Searching)

تستخدم هذه الطريقة للبحث عن قيمة معينة داخل المصفوفة بطريقة متتالية (Sequentially) حيث تتم مقارنة العنصر المراد البحث عنه بالعنصر الأول ثم العنصر الثاني ... إلى حين العثور على هذا العنصر أو بلوغ نهاية المصفوفة.

على سبيل المثال، يقوم البرنامج التالي بتعبئة المصفوفة A بعشرة أعداد صحيحة ثم يطلب من المستخدم إدخال قيمة العدد x الذي يتم البحث عنه في المصفوفة باستخدام طريقة البحث الخطي.

- **البرنامج**

```
#include <iostream>
using namespace std;

int LinearSearch(int SearchKey, int Array[], int ArraySize)
{
      for (int i=0; i < ArraySize; i++)
      {
            if (Array[i] ==SearchKey)     // if found
                  return i;               // return location of SearchKey
      }
      return -1;                          // SearchKey not found
}


int main( )
{
      int A[10] = {2, 0, -6, 8, 5, 12, 15, 5, 0, 20};
      int x;
      cout<<"Enter integer search key: "; cin>>x;

      int element = LinearSearch(x, A, 10);

      if (element != -1)
            cout<<"Found value in element "<<element<<endl;
      else
            cout<<"Value not found"<<endl;
      return 0;
}
```

- **تتبع تنفيذ البرنامج**

```
Enter integer search key : 10
Value not found
Press any key to continue . . .
```

```
Enter integer search key : 12
Found value in element 5
Press any key to continue . . .
```

تمرين

قم بتعديل البرنامج السابق ليقوم بحساب كم مرة يوجد العنصر x داخل المصفوفة A.

- **البرنامج**

```
#include <iostream>
using namespace std;

int Frequency(int SearchKey, int Array[], int ArraySize)
{
      int freq = 0;
      for (int i=0; i < ArraySize; i++)
      {
            if (Array[i] ==SearchKey)     // if found
                  freq++;
      }
      return freq;
}


int main( )
{
      int A[10] = {2, 0, -6, 8, 5, 12, 15, 5, 0, 20};
      int x;
      cout<<"Enter integer search key : ";      cin>>x;

      int f = Frequency(x, A, 10);

      cout<<"Number of occurrences = "<<f<<endl;
      return 0;
}
```

- **تتبع تنفيذ البرنامج**

```
Enter integer search key : 5
Number of occurrences = 2
Press any key to continue . . .
```

3- البحث الثنائي (Binary searching)

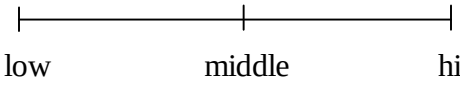
لنفترض أن المصفوفة A مرتبة بشكل تصاعدي (من الصغير إلى الكبير) وأننا نريد معرفة هل أن الرقم key موجود داخل المصفوفة أم لا.

A	2	3	4	5	5	6	8	10	12	15
	0	1	2	3	4	5	6	7	8	9

Key = 9

الهدف من طريقة البحث الثنائي هو قسمة مجال البحث على 2 في كل دورة ليتواصل البحث في النصف الأيمن أو النصف الأيسر للمصفوفة الحالية.

لنفترض أن المتغيرات low ، high و middle تمثل على التوالي الحد الأدنى و الحد الأعلى ووسط مجال البحث.



تتم في كل مرة مقارنة العدد key بوسط المصفوفة A[middle] حيث توجد 3 حالات:

1. إذا كان (key < A[middle]) يتواصل البحث في النصف الأيسر أي [low .. middle-1]
2. إذا كان (key > A[middle]) يتواصل البحث في النصف الأيمن أي [middle +1 .. high]
3. إذا كان (key = A[middle]) يتوقف البحث بما أنه تم العثور على العنصر key.

- **البرنامج**

```
#include <iostream>
using namespace std;

void BinSearch(int x, int AR[ ], int low,int up)
{
      int middle;
      while (low <= up)
      {
            middle = (low + up) / 2;
            if (x < AR[middle])  up = middle - 1;
            if (x > AR[middle])  low = middle + 1;
            if (x == AR[middle])  break;
      }
      if (x == AR[middle])
            cout<<"Index = "<<middle;
      else
            cout<<"Not found..";    // العنصر غير موجود
}

int main( )
{
      int A[10] = {2, 3, 4, 5, 5, 6, 8, 10, 12, 15};
      int key;
      cout<<"Enter the number to search : ";
      cin>>key;
      BinSearch(key,A,0,9);
      cout<<endl;
      return 0;
}
```

- **تتبع تنفيذ البرنامج**

```
Enter the number to search : 20
Not found...
Press any key to continue . . .
```

```
Enter the number to search : 10
Index = 7
Press any key to continue . . .
```

4- طرق فرز المصفوفات

1.4 طريقة الفرز الفقاعي (Bubble sort)

لنفترض أننا نريد ترتيب عناصر المصفوفة A من الأصغر إلى الأكبر باستخدام طريقة الفرز الفقاعي:

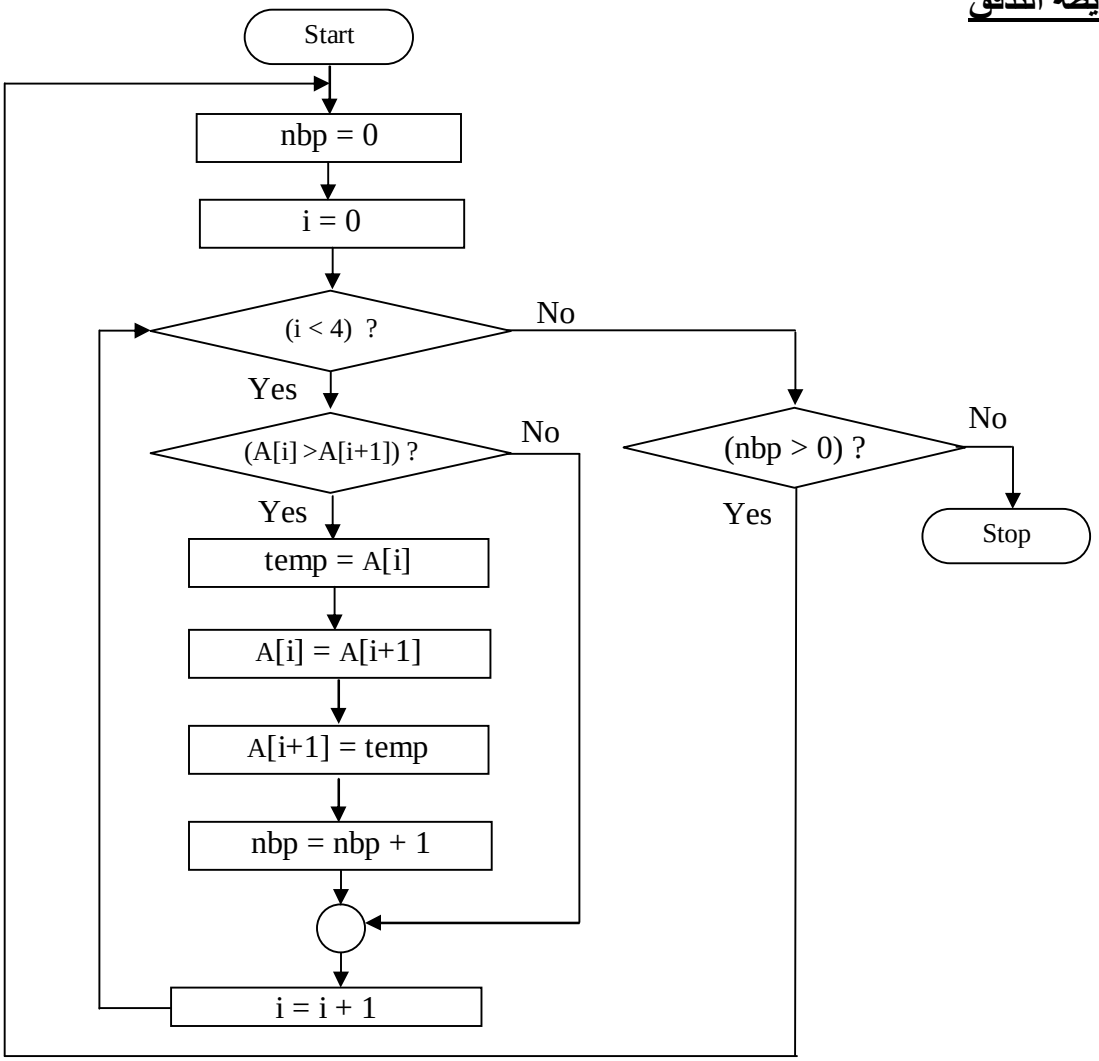
A	6	4	2	1	4
	0	1	2	3	4

- **خطوات الفرز الفقاعي**

1- كرر ما يلي:
   - § قارن كل عنصر في المصفوفة بالعنصر الذي يليه
   - § إذا تبين أن العنصر الحالي A[i] أكبر من A[i+1]، قم بإجراء عملية استبدال (swap).
   - § قم بعد عمليات الإستبدال

2- إذا تمت عملية استبدال واحدة على الأقل فعد إلى الخطوة الأولى وإلا توقف.

- **خريطة التدفق**



- **البرنامج**

```
#include <iostream>
using namespace std;
int main()
{
	int  A[5] = {6, 4, 2, 1, 4};
	int  i,  nbp,  temp;
	do
	{
		nbp = 0;
		for (i = 0; i < 4; i++)
		{
			if (A[i] > A[i+1])
			{
				temp = A[i];
				A[i] = A[i+1];
				A[i+1] = temp;
				nbp = nbp + 1;
			}
		}
	} while (nbp > 0);

	// طباعة عناصر المصفوفة المرتبة على الشاشة
	for (i=0; i<= 4; i++)
		cout<<A[i]<<"\t";
	cout<<endl;

	return 0;
}
```

- **تتبع تنفيذ البرنامج**

*** المصفوفة الأولية:**

A	6	4	2	1	4
	0	1	2	3	4

*** بعد الحلقة الأولى:**

A	4	2	1	4	6
	0	1	2	3	4

*** بعد الحلقة الثانية:**

A	2	1	4	4	6
	0	1	2	3	4

*** بعد الحلقة الثالثة:**

A	1	2	4	4	6
	0	1	2	3	4

*** بعد الحلقة الرابعة:**

A	1	2	4	4	6
	0	1	2	3	4

2.4 طريقة الفرز عن طريق الاختيار (Selection sort)

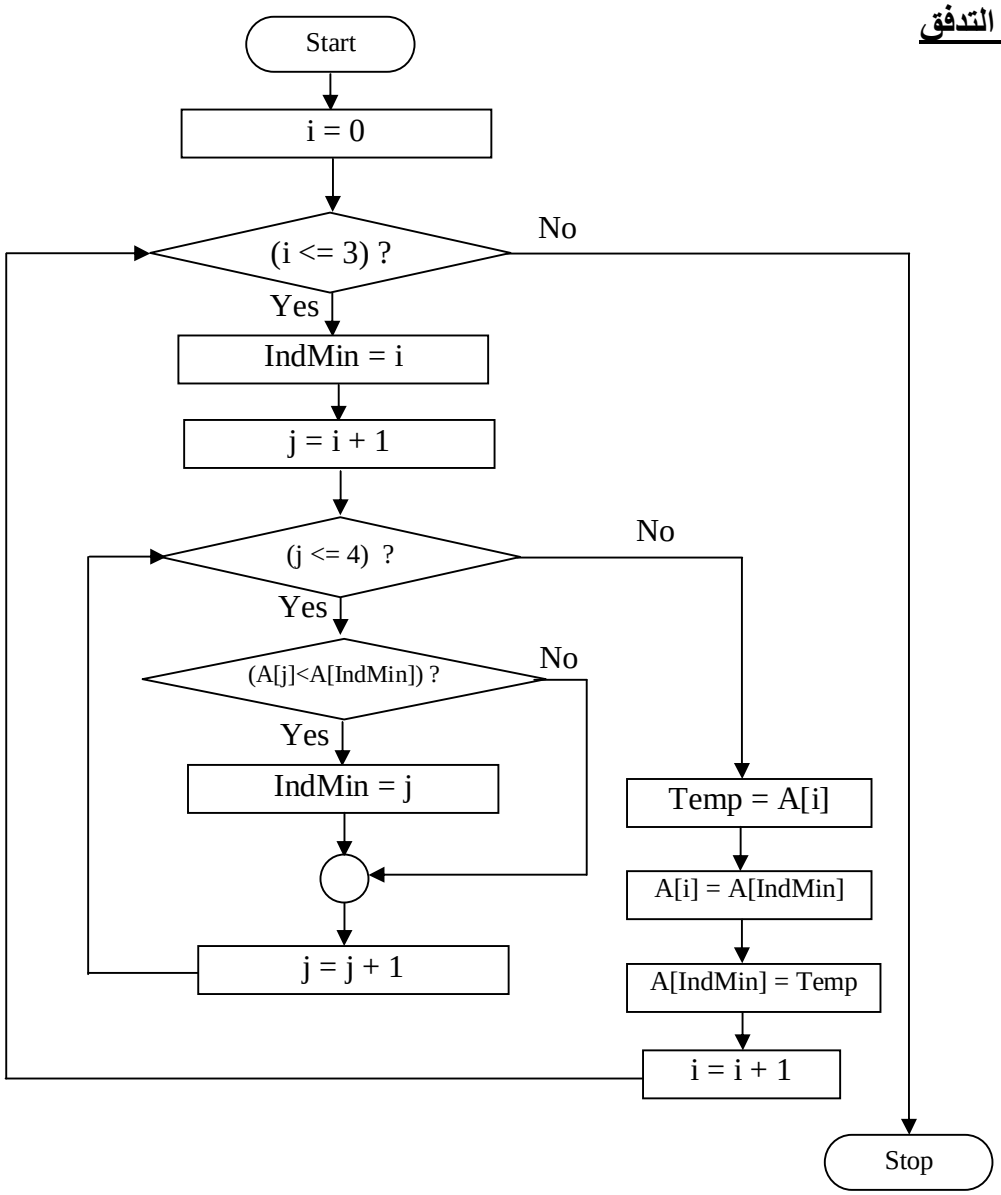
لنفترض أننا نريد ترتيب عناصر المصفوفة A من الأصغر إلى الأكبر باستخدام طريقة الفرز عن طريق الاختيار:

A	6	4	2	1	4
	0	1	2	3	4

- **خطوات الفرز عن طريق الاختيار**

1- البحث عن أصغر عنصر في المصفوفة A[0..4] واستبداله مع A[0]
2- البحث عن أصغر عنصر في المصفوفة A[1..4] واستبداله مع A[1]
3- ... الخ
4- البحث عن أصغر عنصر في المصفوفة A[3..4] واستبداله مع A[3]

- **خريطة التدفق**



- **البرنامج**

```
#include <iostream>
using namespace std;
int main()
{
	int  A[5] = {6, 4, 2, 1, 4};
	int  i,  j,  IndMin, temp;
	for (i = 0; i <= 3; i++)
	{
		IndMin = i;
		for (j = i+1; j <= 4; j++)
		{
			if (A[j] < A[IndMin])
				IndMin = j;
		}
		temp = A[i];
		A[i] = A[IndMin];
		A[IndMin] = temp;
	}

	// طباعة عناصر المصفوفة المرتبة على الشاشة
	for (i=0; i<= 4; i++)
		cout<<"A["<<i<<"] = "<<A[i]<<"\t";
	cout<<endl;
	return 0;
}
```

- **تتبع تنفيذ البرنامج**

*** المصفوفة الأولية:**

A	6	4	2	1	4
	0	1	2	3	4

*** بعد الحلقة الأولى:**

A	1	4	2	6	4
	0	1	2	3	4

*** بعد الحلقة الثانية:**

A	1	2	4	6	4
	0	1	2	3	4

*** بعد الحلقة الثالثة:**

A	1	2	4	6	4
	0	1	2	3	4

*** بعد الحلقة الرابعة:**

A	1	2	4	4	6
	0	1	2	3	4

3.4 الفرز عن طريق الإدراج (Insertion sort)

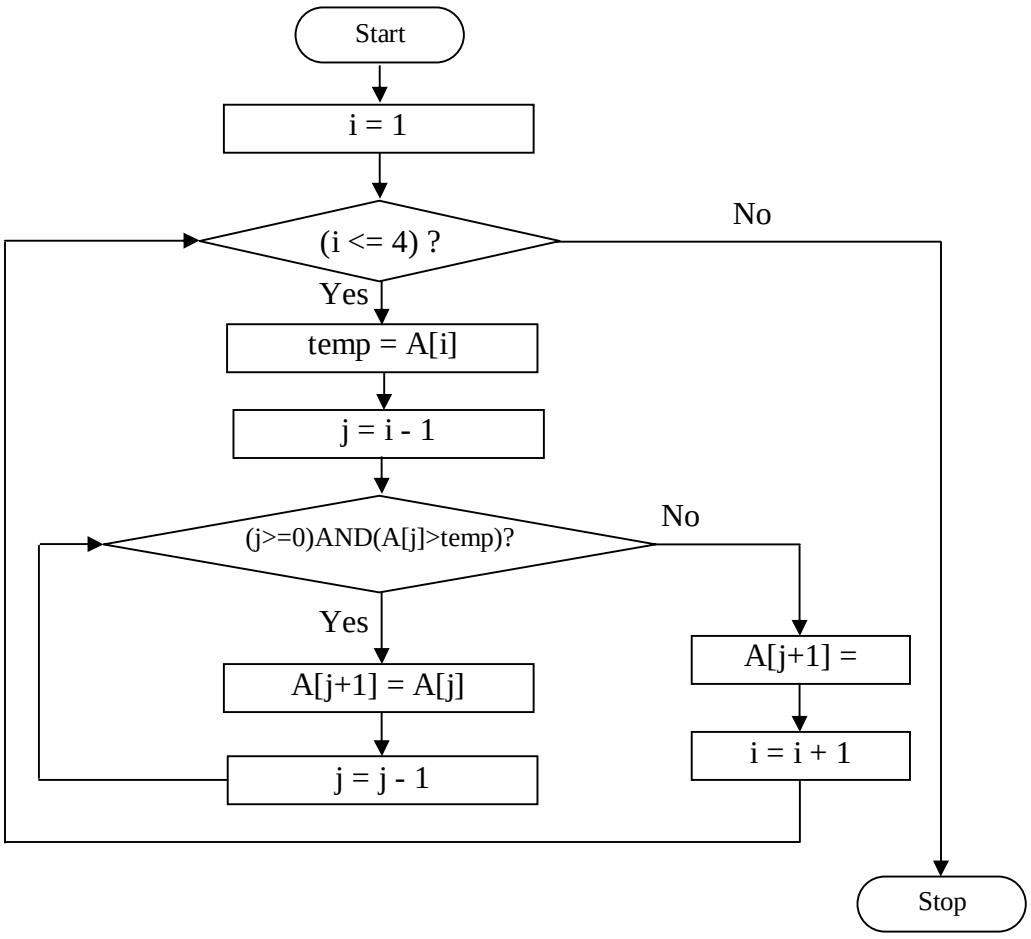
لنفترض أننا نريد ترتيب عناصر المصفوفة A من الأصغر إلى الأكبر باستخدام طريقة الفرز عن طريق الإدراج:

A	6	4	2	1	4
	0	1	2	3	4

- **<u>خطوات الفرز عن طريق الإدراج</u>**

1- مقارنة A[1] بالعنصر A[0] وإجراء عملية استبدال عند اللزوم للحصول على القائمة المرتبة A[0..1]
2- مقارنة A[2] بالعنصر A[1] ثم A[0] وإجراء عمليات استبدال عند اللزوم للحصول على القائمة المرتبة A[0..2]
3- ... الخ
4- مقارنة A[4] بالعناصر التي تسبقه وإجراء عمليات استبدال عند اللزوم للحصول على المصفوفة المرتبة A[0..4]

- **<u>خريطة التدفق</u>**



- **البرنامج**

```
#include <iostream>
using namespace std;

int main()
{
	int A[5] = {6, 4, 2, 1, 4};

	int i, j, temp;
	for (i = 1; i <= 4; i++)
	{
		temp = A[i];
		j = i - 1;
		while ((j >= 0) && (A[j] > temp))
		{
			A[j+1] = A[j];
			j = j - 1;
		}

		A[j+1] = temp;
	}
	// طباعة عناصر المصفوفة المرتبة على الشاشة
	for (i = 0; i <= 4; i++)
		cout<<"A["<<i<<"] = "<<A[i]<<"\t";
	cout<<endl;
	return 0;
}
```

- **تتبع تنفيذ البرنامج**

*** المصفوفة الأولية:**

A	6	4	2	1	4
	0	1	2	3	4

*** بعد الحلقة الأولى:**

A	4	6	2	1	4
	0	1	2	3	4

*** بعد الحلقة الثانية:**

A	2	4	6	1	4
	0	1	2	3	4

*** بعد الحلقة الثالثة:**

A	1	2	4	6	4
	0	1	2	3	4

*** بعد الحلقة الرابعة:**

A	1	2	4	4	6
	0	1	2	3	4

4.4 طريقة الفرز غير المباشر (Indirect sort)

لنفترض أننا نريد ترتيب عناصر المصفوفة A من الأصغر إلى الأكبر باستخدام طريقة الفرز غير المباشر:

A	6	4	2	1	4
	0	1	2	3	4

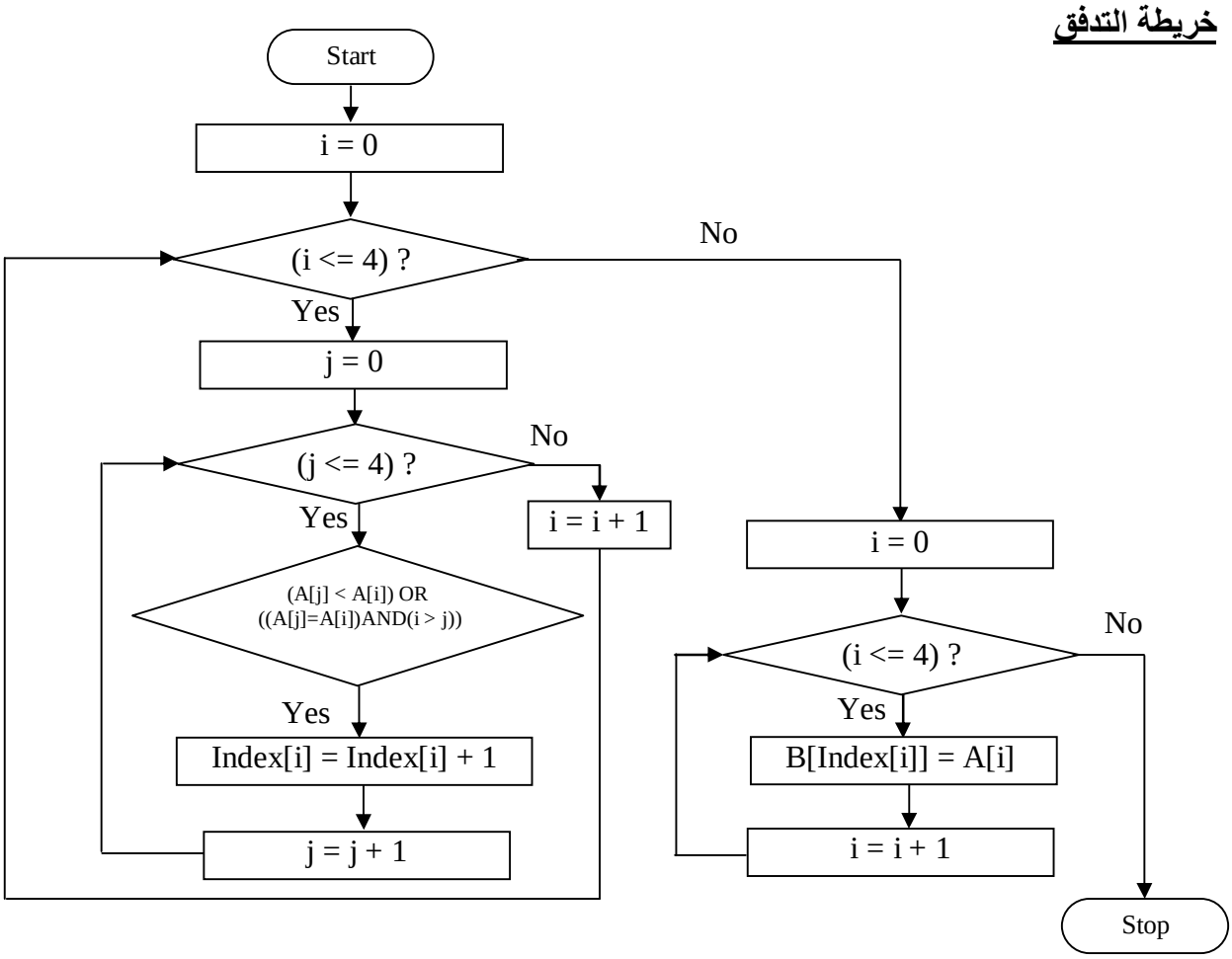
تقضي هذه الطريقة بإنشاء مصفوفة أخرى Index تحتوي في البداية على أصفار. ثم يقع تحديث هذه المصفوفة بحيث يشير كل عنصر إلى المكان الذي ينبغي أن يتواجد فيه العنصر المقابل في المصفوفة Aحتى تكون هذه الأخيرة مرتبة . يتم ذلك بحساب عدد العناصر التي هي أقل من العنصر الحالي أو مساوية له وتوجد عن يساره.

Index	4	2	1	0	3
	0	1	2	3	4

ثم يتم بعد ذلك نقل عناصر المصفوفة A إلى المصفوفة الجديدة B حسب الأماكن المبينة في المصفوفة Index.

B	1	2	4	4	6
	0	1	2	3	4

- **<u>خريطة التدفق</u>**



- **البرنامج**

```
#include <iostream>
using namespace std;

int main()
{
	int A[5] = {6, 4, 2, 1, 4};
	int index[5] = {0, 0, 0, 0, 0};
	int B[5];
	int i,j;

	for(i=0; i <= 4; i++)
	{
		for(j = 0; j <= 4; j++)
		{
			if ((A[j] < A[i]) || ((A[j] == A[i]) &&(i > j)))
				index[i]++;
		}
	}

	for(i = 0; i <= 4; i++)
		B[index[i]] = A[i] ;

	cout<<"A: \t";
	for(i = 0; i <= 4; i++)
		cout<<A[i]<<"\t";

	cout<<"\n\nIndex: \t";
	for(i = 0; i <= 4; i++)
		cout<<index[i]<<"\t";

	cout<<"\n\nB: \t";
	for(i = 0; i <= 4; i++)
		cout<<B[i]<<"\t";

	cout<<endl<<endl;
	return 0;
}
```

- **تتبع تنفيذ البرنامج**

```
A:	6	4	2	1	4

Index:	4	2	1	0	3

B:	1	2	4	4	6

Press any key to continue . . .
```

5.4 الفرز بطريقة شال SHELL

قام الباحث Donald Shell بإدخال تحسينات على طريقة الفرز بالإدراج (Insertion Sort) تتمثل أساسا في تقليص عمليات تحريك البيانات وذلك بإجراء عمليات مقارنة بين عناصر بعيدة نوعا ما ثم تقليص هذا المسافة تدريجيا حتى الوصول إلى طريقة الفرز بالإدراج (مقارنة العناصر المتجاورة).

تحسب المسافة باستخدام التتابع:

$$\begin{cases} h_1 = 1 \\ h_2 = 4 \\ h_3 = 13 \\ \dots \\ h_n = 3 \times h_{n-1} + 1 \end{cases}$$

- **<u>خطوات الفرز بطريقة شال</u>**

لنفترض أننا نريد ترتيب عناصر المصفوفة التالية تصاعديا:

A	6	5	4	9	1	0	7	8	3	2
	0	1	2	3	4	5	6	7	8	9

1. نبحث عن أكبر قيمة ممكنة للخطوة h حسب التتابع المذكور، في هذه الحالة **(h = 4)**
2. نقوم بترتيب المصفوفة المكونة من العناصر (A[0], A[4], A[8]) باستخدام طريقة الإدراج:

A	1	5	4	9	3	0	7	8	6	2
	0	1	2	3	4	5	6	7	8	9

3. نقوم بترتيب المصفوفة المكونة من العناصر (A[1], A[5], A[9]) باستخدام طريقة الإدراج:

A	1	0	4	9	3	2	7	8	6	5
	0	1	2	3	4	5	6	7	8	9

4. نقوم بترتيب المصفوفة المكونة من العناصر (A[2], A[6]) باستخدام طريقة الإدراج:

A	1	0	4	9	3	2	7	8	6	5
	0	1	2	3	4	5	6	7	8	9

5. نقوم بترتيب المصفوفة المكونة من العناصر (A[3], A[7]) باستخدام طريقة الإدراج:

A	1	0	4	8	3	2	7	9	6	5
	0	1	2	3	4	5	6	7	8	9

6. نقوم بتقليص المسافة باستخدام الصيغة

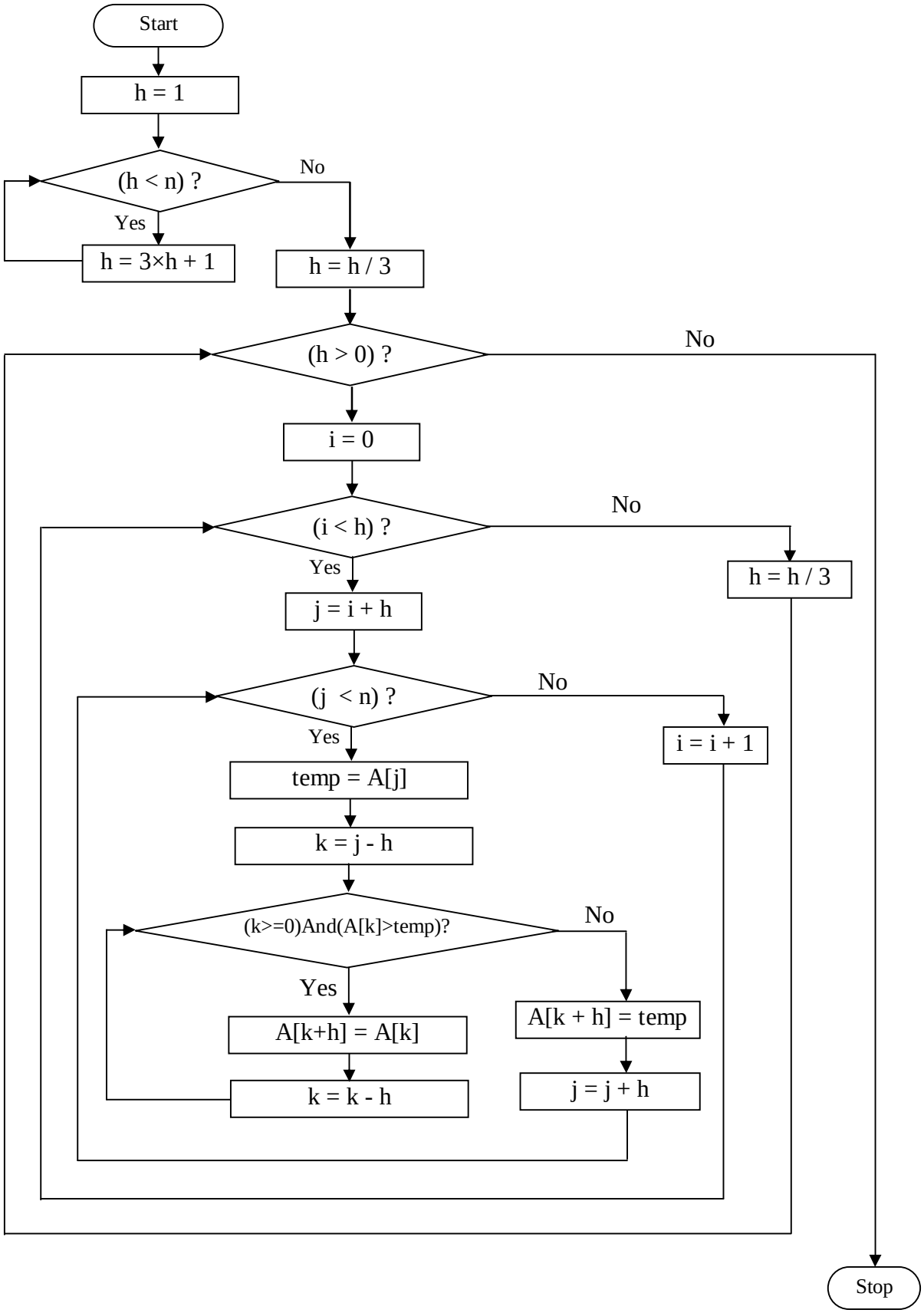
$$h = h / 3$$

مما يجعل **h = 1**.

7. نقوم بترتيب المصفوفة المكونة من العناصر (A[0..9]) باستخدام طريقة الإدراج بحيث تكون النتيجة كما يلي:

A	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

- **خريطة التدفق**



- **البرنامج**

```
#include <iostream>
using namespace std;
const int n = 10;          //عدد عناصر المصفوفة
int main()
{
	int A[n] = {6, 5, 4, 9, 1, 0, 7, 8, 3, 2};
	int i, j, k, temp;

	// calculation of the first value of h
	int h = 1;
	while (h < n)
		h = 3*h +1;
	h = h / 3;

	while ( h > 0)
	{
		cout<<"h = "<<h<<endl;

		// sort by insertion of the h arrays
		for(i = 0; i < h; i++)
		{
			j = i + h;
			while (j < n)
			{
				temp = A[j];
				k = j - h;
				while ((k >= 0) && (A[k] > temp))
				{
					A[k+h] = A[k];
					k = k - h;
				}
				A[k+h] = temp;
				j = j + h;
			}
			for(int cnt = 0; cnt <= 9; cnt++)
				cout<<A[cnt]<<"\t";
			cout<<endl;
		}
		h = h / 3;
	}
	cout<<endl;
	return 0;
}
```

- **تتبع تنفيذ البرنامج**

```
h = 4
1	5	4	9	3	0	7	8	6	2
1	0	4	9	3	2	7	8	6	5
1	0	4	9	3	2	7	8	6	5
1	0	4	8	3	2	7	9	6	5

h = 1
0	1	2	3	4	5	6	7	8	9

Press any key to continue . . .
```

5- المصفوفات متعددة الأبعاد

المصفوفة متعددة الأبعاد هي مصفوفة من المصفوفات.

يمكن تعريف المصفوفة متعددة الأبعاد بنفس طريقة تعريف المصفوفات ذات البعد الواحد مع إضافة حجم الأبعاد التي تلي البعد الأول.

على سبيل المثال، يقوم الأمر التالي:

```
int A[4][3];
```

بإنشاء مصفوفة ذات بعدين (Matrix). يحتوي البعد الأول على عدد الصفوف (4 في هذه الحالة) بينما يحتوي البعد الثاني على عدد الأعمدة في كل صف (3 في هذه الحالة).

A

	0	1	2
0			
1			
2			
3			

يمكن أيضا إعطاء قيما ابتدائية لعناصر المصفوفة كما في المثال التالي:

```
int A[4][3] = {0,1,2, 3,4,5, 6,7,8, 9,10,11};
```

A

	0	1	2
0	0	1	2
1	3	4	5
2	6	7	8
3	9	10	11

للوصول إلى أحد عناصر المصفوفة ذات البعدين يجب تحديد رقم الصف ورقم العمود، كل منهما داخل قوسين مستقلين كما في العبارة التالية:

```
A[1][2] = 100;
```

التي تقوم بإسناد القيمة 100 إلى العمود الثالث بالصف الثاني (لأن ترقيم عناصر المصفوفة يبدأ من الصفر كما ذكرنا).

A

	0	1	2
0	0	1	2
1	3	4	**100**
2	6	7	8
3	9	10	11

تمرين رقم 1

كيف سيكون محتوى المصفوفة M بعد تنفيذ الكود التالي:

```
int M[10][10];
int i, j;
for (i = 0; i <= 9; i++)
      for (j = 0; j <= 9; j++)
            M[i][j] = i * j;
```

الحل

M	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	10	12	14	16	18
3	0	3	6	9	12	15	18	21	24	27
4	0	4	8	12	16	20	24	28	32	36
5	0	5	10	15	20	25	30	35	40	45
6	0	6	12	18	24	30	36	42	48	54
7	0	7	14	21	28	35	42	49	56	63
8	0	8	16	24	32	40	48	56	64	72
9	0	9	18	27	36	45	54	63	72	81

تمرين رقم 2

أكتب برنامجا بلغة C++ يقوم بإنشاء وتعبئة المصفوفتين A و B ذاتا البعدين ثم يقوم بجمع عناصر المصفوفتين وحفظ الناتج داخل المصفوفة C.

مثال

A

0	4	8
1	5	9
2	6	10
3	7	11

B

1	2	3
1	2	3
1	2	3
1	2	3

C

1	6	11
2	7	12
3	8	13
4	9	14

- **الحل**

```
#include <iostream>
using namespace std;

int main()
{
      int A[4][3]={0,4,8,1,5,9,2,6,10,3,7,11};
      int B[4][3]={1,2,3,1,2,3,1,2,3,1,2,3};
      int C[4][3];

      int i,j;
```

```
	cout<<"Matrix C:\n\n";
	for (i=0; i <= 3; i++)
	{
		for (j = 0; j <= 2; j++)
		{
			C[i][j] = A[i][j] + B[i][j];
			cout<<C[i][j]<<"\t";
		}
		cout<<endl;
	}
	cout<<endl;
	return 0;
}
```

- **مخرجات البرنامج**

```
Matrix C:

1       6       11
2       7       12
3       8       13
4       9       14

Press any key to continue . . .
```

تمرين رقم 3

أكتب برنامجا بلغة C++ يقوم بإيجاد تبادل عناصر المصفوفة A بحيث تصبح الصفوف أعمدة والأعمدة صفوف (transposition).

مثال

A

0	1	2
3	4	5
6	7	8
9	10	11

ð

B

0	3	6	9
1	4	7	10
2	5	8	11

- **الحل**

```
#include <iostream>
using namespace std;
int main()
{
	int A[4][3] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};
	int B[3][4];
	int i, j;
	cout<<"Matrix B:\n\n";
	for (i = 0; i <= 2; i++)
	{
		for (j = 0; j <= 3; j++)
		{
			B[i][j] = A[j][i];
			cout<<B[i][j]<<"\t";
		}
		cout<<endl;
	}
	return 0;
}
```

- **مخرجات البرنامج**

```
Matrix B:

0       3       6       9
1       4       7       10
2       5       8       11

Press any key to continue . . .
```

تمرين رقم 4

أكتب برنامجا بلغة C++ لإنشاء الجزء الاول من مثلث باسكال على النحو التالي:

```
1     0       0       0       0       0       0       0       0       0
1     1       0       0       0       0       0       0       0       0
1     2       1       0       0       0       0       0       0       0
1     3       3       1       0       0       0       0       0       0
1     4       6       4       1       0       0       0       0       0
1     5       10      10      5       1       0       0       0       0
1     6       15      20      15      6       1       0       0       0
1     7       21      35      35      21      7       1       0       0
1     8       28      56      70      56      28      8       1       0
1     9       36      84      126     126     84      36      9       1

Press any key to continue . . .
```

الحل

```
#include <iostream>
using namespace std;
const int n = 10;
int main()
{
	int A[n][n];
	int i, j;

	for (i = 0; i < n; i++)
		for (j = 0; j < n; j++)
			A[i][j] = 0;
	A[0][0] = 1;
	for (i = 1; i < n; i++)
	{
		A[i][0] = 1;
		for (j = 1; j < n; j++)
		{
			A[i][j] = A[i-1][j-1] + A[i-1][j];
		}

	}
	for (i = 0; i < n; i++)
	{
		for (j = 0; j < n; j++)
			cout<<A[i][j]<<"\t";
	}
	cout<<endl;
	return 0;
}
```