كلية الهندسة قسم المعلوماتية

بنى معطيات 1

Data Structure 1

ا. د. علي عمران سليمان

محاضرات الأسبوع الرابع
المكدسات
Stacks

الفصل الثاني 2024-2025

محتوى الفصل

المكدسات	stacks
1- مقدمة.	
2- مدخل إلى المكدسات .	
3- تصميم وبناء الصنف Stack.	
4- بعض التطبيقات على المكدسات .	
5- تحويل العدد الطبيعي إلى نظام العد الثنائي.	
5- تحوي التعبير infix to RPN .	
6- program infix to RPN.	

References

- Deitel & Deitel, Java How to Program, Pearson; 10th Ed(2015)

- د.علي سليمان، بنى معطيات بلغة JAVA، بنى معطيات بلغة C++، بنى معطيات بلغة Pascal جامعة تشرين 2014، 2007، 1998

1- مقدمة:

إن تحقيق بنى المعطيات المجردة ADT يتضمن تحديد البنى التخزينية والمفاهيمية لعناصر البيانات Physical, Logical . وتعريف خوارزميات إنجاز العمليات الأساسية والعلاقات(تخزين storage، استرجاع retrieval ومعالجة manipulation).

- إن أهمية وجود أنماط البيانات المجردة Abstract Data Type (ADT) تكمن في أنك تستطيع استخدامها ودراستها بدون أن تكون معنياً بتفاصيل بنيتها وتحقيقها.
- رأينا نمط المعطيات سلسلة محرفية كما هي معرفة بلغة C++ كمصفوفة من النمط char ومكتبات من توابع معالجة هذه المصفوفات. ورأينا أيضاً الصنف string في لغة C++. إن السلاسل تعتبر مثالاً بسيطاً عن بنى المعطيات المهيكلية أو ما يدعى النمط الحاوي container type التي تخزن مجموعة من قيم البيانات والتي يوجد العديد منها.
- سنتعرف الآن على أحدها وهو المكدس stack وبعض تطبيقاته. لقد تم تحقيق هذه البنية باستخدام مصفوفة.

نركز في هذا الفصل على بناء الصنف stack وسنرى لاحقاً بعض التحسينات على هذه البنية وأهمها:

- تحويله إلى قالب صنف وبالتالي يصبح نمط حاوي عام يمكن أن يعالج أي نمط من عناصر المكدس.
- استخدام vector لتخزين الحاوي وبالتالي سعة المكدس يمكن أن تكبر بحسب الحاجة.
- التعرف على الحاوي stack المعرف في مكتبة القوالب القياسية STL (Standard Template Library).

1- مدخل إلى المكدسات 1

introduction to stacks 1

لننظر إلى المسائل التالية:

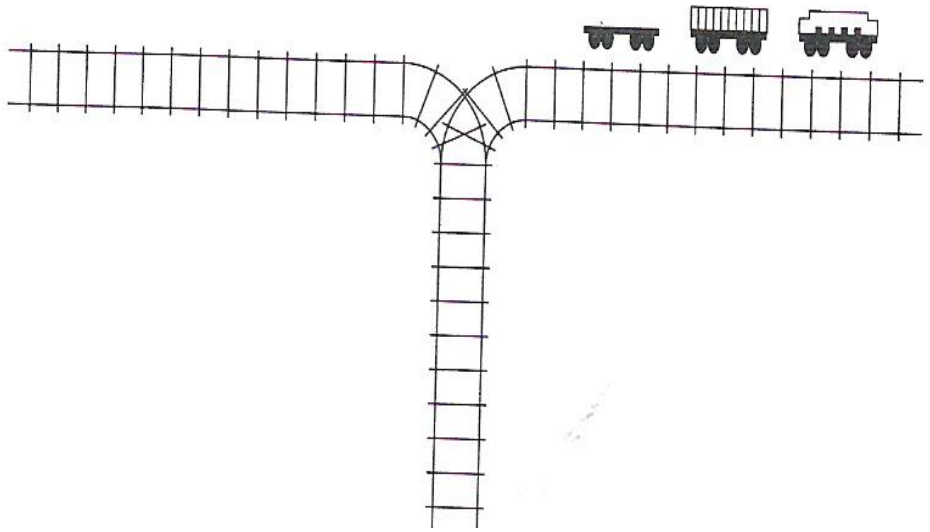
المسألة الأولى:

نريد كتابة برنامج لمحاكاة لعبة أوراق اللعب, أحد مبادئ هذه اللعبة يقوم على تكديس الأوراق المرمية, في أي دور يمكن للاعب أن يقوم برمي ورقة من يده إلى الكدسة أو أن يقوم بتناول أعلى ورقة في كدسة الأوراق. ما هو نمط البيانات اللازم لنمذجة كدسة الأوراق ؟.

المسألة الثانية:

نريد كتابة برنامج لنمذجة نقطة تحويل خط سكة حديدية (محطة رأسية)، أحد أجزاء شبكة التحويل يتضمن مساراً رئيسياً ومساراً جانبياً يمكن للقطارات أن تحول إليه في أي وقت.

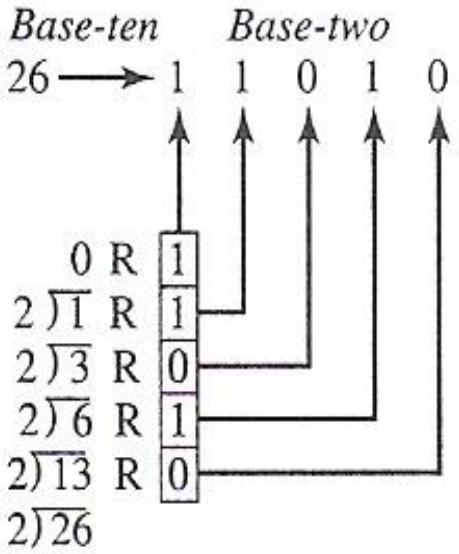
ما هو نمط البيانات الذي يمكن استخدامه لنمذجة عمل المسار الجانبي ؟.



المسألة الثالثة:إن إحدى المهام التي يجب إنجازها خلال تنفيذ تابع ( )f هي حفظ موقع المعلومات الخاصة بالتابع ( )f - قيم البارامترات, المتحولات المحلية... وهكذا – وبالتالي عند تنفيذ ( )f تمت مقاطعته باستدعاء تابع آخر ( )g, يمكن استئناف تنفيذ ( )f عندما ينتهي ( )g, في حال استدعى ( )g تابعاً آخر ( )h وبالتالي فإن معلومات ( )g يجب أن تخزن بحيث يكون من الممكن استئناف التنفيذ لاحقاً, كما أن ( )h قد يستدعي تابعاً آخر وهكذا. ما هو نمط البيانات الذي يمكن استخدامه لتخزين معلومات التوابع ؟.

المسألة الرابعة: UnDo(^Z) في تطبيقات Microsoft Office

1- مدخل إلى المكدسات 2



المسألة الخامسة:

تخزن عناصر البيانات في ذاكرة الحاسوب باستخدام التمثيل الثنائي (base-two), هذا يعني أن التمثيل العشري (base-ten) للأعداد الصحيحة الذي نستخدمه في كتابة البرامج يجب أن يتم تحويله إلى التمثيل الثنائي, أحد خوارزميات تنفيذ عملية التحويل هذه تقوم على الحصول على بواقي التقسيم المتكرر على 2 تخزيمها ومن ثم التقسيم على 2، وهذه البواقي تشكل بمجموعها الخانات الثنائية للتمثيل بالأساس 2 وتؤخذ من الأحدث إلى الاقدم وتكتب من اليسار إلى اليمين كما يبين الشكل:

ما هو نمط البيانات اللازم لتخزين بواقي القسمة ؟.

إن كلاً من المسائل السابقة تتضمن مجموعة من عناصر البيانات المترابطة، مجموعة من أوراق اللعب في المسألة الأولى, مجموعة من القطارات المتتالية في المسألة الثانية, مجموعة من معلومات التوابع في الثالثة والتراجع عن أخر عمل في الرابغة وتتالي من البواقي في الخامسة.

في كل حالة من الحالات الخمس نحتاج إلى نمط بيانات بنيوي يعتمد العمليات التالية: " المرمي أخيراً الملتقط أولاً ", " الداخل أخيراً الخارج أولاً ", " المخزن أخيراً المحذوف أولاً ", المفعول آخراً المتراجع عنه أولاً " ، " المتولد أخيراً المعروض أولاً « أي LIFO.

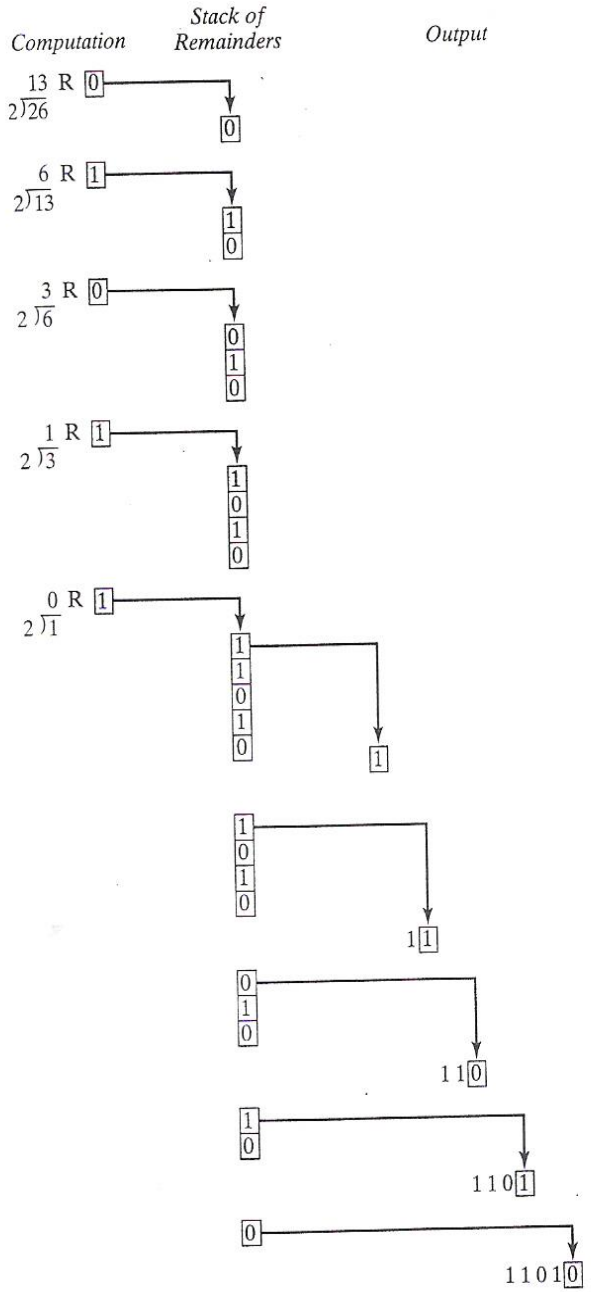
لتوضيح هذه العمليات, لنركز على المسألة الخامسة, لكي نحصل على التمثيل الثنائي للعدد 26 نحتاج لأن نقوم بجساب بواقي القسمة على 2 بتكديس البواقي ثم التقسيم المتكرر على 2 وعندما تنتهي عملية القسمة نقوم برصف بواقي القسمة من اليسار إلى اليمين وفق المبدأ التالي LIFO

1- مدخل إلى المكدسات 3

يمكن وضع خوارزمية إنجاز هذه العملية كما يلي:

1. إنشاء مكدس فارغ لتخزين البواقي.
2. طالما أن العدد لا يساوي الصفر:
   a. أحسب الباقي الناجم عن قسمة العدد على 2.
   b. قم بإضافة هذا الباقي إلى مكدس البواقي.
   c. استبدال العدد المختبر بناتج القسمة على 2.
3. طالما أن مكدس البواقي غير فارغ:
   a. أزل الرقم الموضوع في أعلى المكدس.
   b. قم بعرضه.

الشكل المجاور يوضح هذه العملية:



إن هذا النمط من المعالجة الذي يعتمد مبدأ " الداخل أخيراً يخرج أولاً " يحصل في العديد من التطبيقات, وبنية المعطيات التي تستخدم مبدأ (Last In First Out: LIFO) **تدعى المكدس stack.**

1- مدخل إلى المكدسات 4

المكدس كنمط بيانات مجرد

مجموعة من عناصر البيانات:

مجموعة مرتبة من عناصر البيانات يمكن الوصول فقط إلى العنصر الذي في نهايتها والذي يدعى قمة top المكدس.

العمليات الأساسية:

عند التخوين:

- بناء المكدس (عادة يكون فارغ).
- اختبار كون المكدس غير ممتلئ.
- إضافة عنصر إلى أعلى المكدس (**push**).

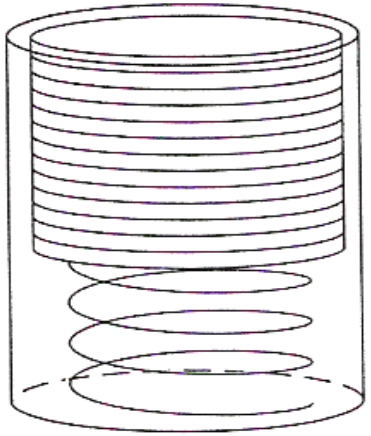
عند التفريغ:

- اختبار كون المكدس غير فارغ.
- عرض العنصر الأعلى من المكدس (**top**).
- إزالة العنصر الأعلى في المكدس (**pop**).

سيكون من المفضل أن نقوم ببناء الصنف stack لأن ذلك سيجعل من السهل كتابة برامج قصيرة لإنجاز العمليات التي تتطلب مكدس.

1- مدخل إلى المكدسات 5

introduction to stacks 5

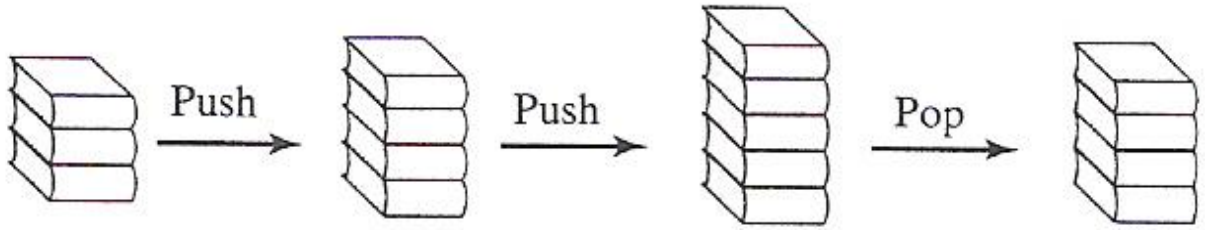


مقاربة أولى:

تقوم هذه المقاربة للمكدس على نموذج مكدس صحون ذو نابض كما هو مبين في الشكل المجاور. هنا مشكلة عند إضافة أو سحب أية طبيق سيتم إزاحة باقي الاطباق للاسفل أو للأعلي على الترتيب. تعتبر عملية غير مجدية ومستهلكة للوقت.

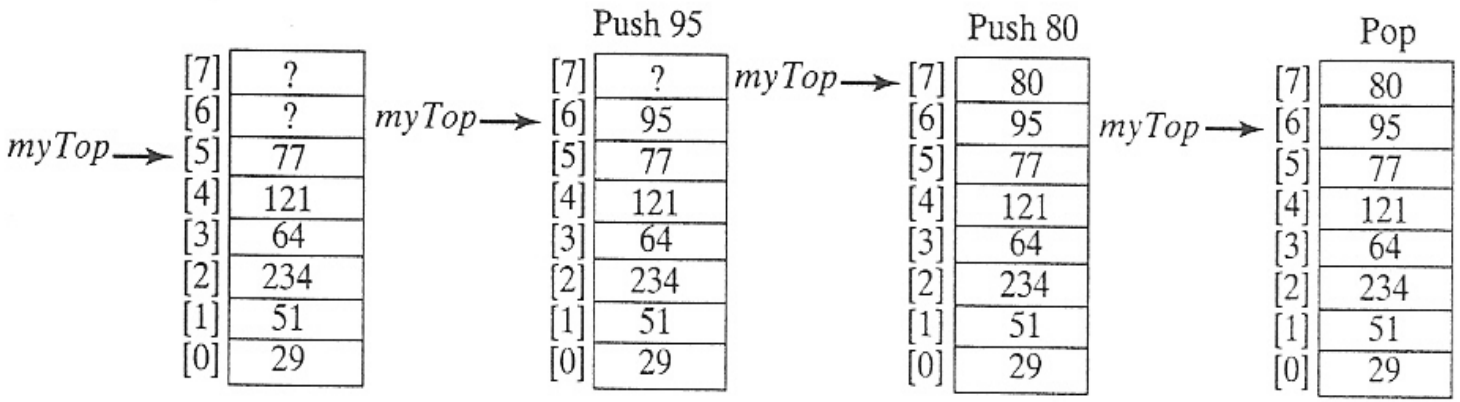
مقاربة ثانية:

مقاربة بنية المكدس بنموذج مكدس كتب موضوعة على طاولة, في هذه المقاربة يمكن إضافة وإزالة كتب إلى ومن قمة المكدس بدون الحاجة لتحريلك الكتب الأخرى.



في المقاربة الأولى, كانت قمة المكدس ثابتة وأسفل المكدس تتغير. في المقاربة الثانية العملية معكوسة أسفل المكدس ثابت وقمته تتغير.

لنمذجة المكدس وفق هذه المقاربة, نأخذ المثال السابق وننفذ العمليات الثلاث فنلاحظ أن العملية



يمكن البدء ببناء المكدس في الصنف stack باختيار أعضاء البيانات كما يلي:

- مصفوفة array لاحتواء عناصر المكدس.
- عدد صحيح integer للإشارة إلى قمة المكدس.

يأخذ التصريح عن المصفوفة الشكل التالي:

int array [CAPACITY]; أو استخدام الاسم المرادف من int مثل typedef int ArrayElementType عندها يمكن أن نكتب :

ArrayElementType array[CAPACITY];

لبناء مكدس من القيم الحقيقية : typedef double ArrayElementType ; في هذا التعريف تذكر ArrayElementType بدل double

1- مدخل إلى المكدسات 7

سنقوم بوضع التصريح typedef خارج التصريح عن الصنف لتسهيل تغيير النوع عندما نريد, وسهل فيما بعد بناء قالب template لتعريف المكدس.
أما بالنسبة لـ CAPACITY فتمثل سعة المكدس ويمكن تعريفها كما يلي:

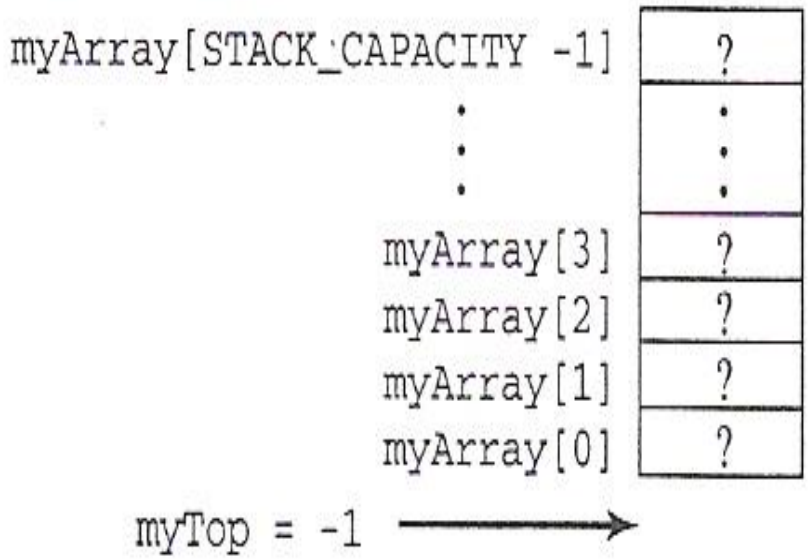
```
const int STACK_CAPACITY=..... ;
```

وإذا أردنا تعريفها ضمن الصنف فمن الممكن جعلها قيمة ساكنة static بحيث تستطيع جميع أغراض الصنف stack استخدام نفس القيمة.

```
static const int STACK_CAPACITY=..... ;
```

يمكن أن يكون التصريح عن الصنف stack كما يلي:

- الملف الرأسي stack.h للتصريح عن الصنف,
- ملف تنفيذ الصنف stack.cpp لتعريف توابع الصنف)



<u>سيتم إدراج التابع الباني constructors:</u> سيقوم المنقح بتخصيص الذاكرة لأعضاء البيانات myArray وmyTop وبالتالي يجب على التوابع البانية للصنف Satck تحتاج فقط أن تقوم بالتهيئة اللازمة لإنشاء مكدس فارغ, وبالتالي يكفي أن تقوم بإعطاء myTop القيمة 1- للإشارة إلى مكدس فارغ.

1- يضاف تعريف التابع الباني Stack( ); ضمن الصنف وكتابة التابع الباني ضمن أو خارج الصنف.

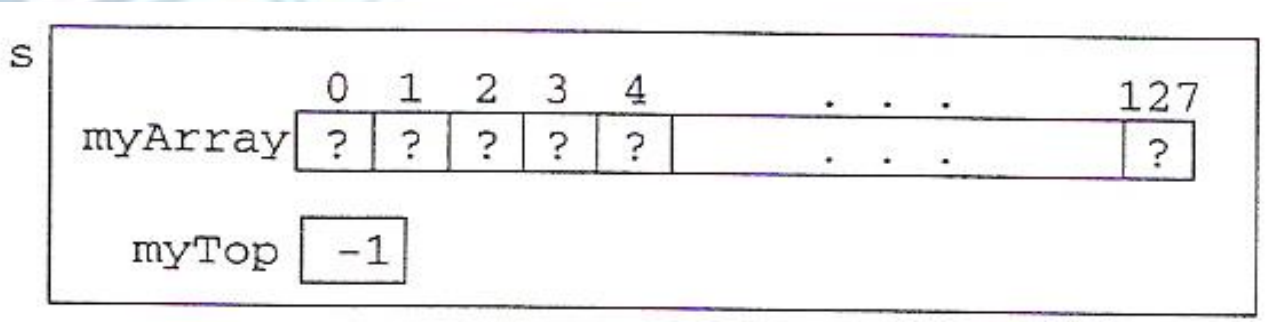
1- مدخل إلى المكدسات 8

```
#ifndef STACK
#define STACK
const int STACK_CAPACITY=128;
typedef int StackElement;
class Stack
{
/**** function members ****/
public:
Stack( );
/**** data members ****/
private:
        StackElement myArray[STACK_CAPACITY];
        int myTop;
}; //end of class declaration
inline Stack::Stack( ) { myTop= -1; }
#endif
```

فإن التصريح التالي:

```
Stack s;
```

سيقوم ببناء s كما يلي:



سنقوم بشرح وكتابة الكود الضروري لكل تابع من تو ابع المكدس وجمعها في النهاية ضمن الملفين المذكورين.

2-تابع اختبار هل المكدس فارغ empty: إن عملية اختبار فيما إذا كان المكدس فارغاً أصبحت الآن بديهية إذ أننا لا نحتاج لأكثر من اختبار عضو البيانات myTop فيما إذا كان مساوياً لـ 1-, سنجعل هذا التابع تابعاً عضواً سطرياً ثابتاً .

```
bool empty( ) const ;
 inline bool Stack::empty() const {return (myTop==-1);}
```

كتابة التابع خارج الصنف كما يلي:

ويتم إختبارة من قبل التابع الرئيس:

```
#include "stack.h"
#include<iostream>
using namespace std;
int main( )
{
Stack s;
cout<<boolalpha<<" s empty ? "<<s.empty()<<endl;
return 0;
}
```

3- تابع الإضافة إلى المكدس push: يمكن تعريف عملية الإضافة على أنها تتلقى بارامترين الأول هو غرض من نوع Stack (يمكن أن يكون ذلك بشكل ضمني), والثاني القيمة value التي نريد إضافتها. وتعيد المكدس المعدل (يمكن أن يكون ذلك بشكل ضمني) حيث القيمة value مضافة إلى قمته.

3-1- ضمن الصنف :

```
void push (const StackElement & value);
```

يتم إنجاز عملية الإضافة وفقاً للخوارزمية التالية:

1. اختبار فيما إذا كانت المصفوفة ممتلئة من خلال اختبار myTop إذا كانت مساوية لنهاية المصفوفة.
2. إذا لم تكن ممتلئة عندئذ:
   a. زيادة قيمة myTop بمقدار 1.
   b. تخزين القيمة value في الموقع myArray[myTop].

وإلا إعطاء إشارة خطأ تشير إلى أن المكدس ممتلئ.يتم وضع التابع في ملف التحقيق:

```
void Stack::push(const StackElement &value)
{if (myTop<STACK_CAPACITY-1)
{++myTop;    myArray[myTop]=value;     }
else
cerr<<"**** stack is full or overflow... can't add new value ***** \n
must increase value of STACK_CAPACITY in stack.h\n"; }
```

2-3- ننادي التابع من ملف التابع الرئيس main وفق التالي:

```
for (int i=1;i<=128;i++)
      s.push(i);   cout<<"stack should now be full\n";
```

4- تابع عرض محتوى المكدس output: إن عملية إخراج محتوى المكدس ليست عملية أساسية من العمليات المعرفة على المكدس كنمط بيانات مجرد ولكنها عموماً عملية ضرورية للتعامل مع المكدسات في برامج الاختبار, يمكن تحقيق هذه العملية من خلال التابع الثابت display( ) الذي يقوم ببساطة بالتجول عبر المصفوفة وعرض القيم المخزنة فيها.

1-4- ضمن الصنف :

```
void display( ) const ;
```

2-4- يتم وضع التابع في ملف التحقيق:

```
void Stack::display( ) const
{ for (int i=myTop ; i>=0 ; i--) cout<<myArray[i]<<endl;}
```

3-4- ننادي التابع من ملف التابع الرئيس main وفق التالي:

```
s.display();
```

5 - تابع إخراج القيمة في أعلى المكدس top: يمكن تعريف العملية top التي تحدد القيمة المخزنة في أعلى المكدس على أنها تتلقى بارامتراً واحداً وهو غرض من نوع Stack (يمكن أن يكون ذلك بشكل ضمني) وتعيد القيمة المخزنة في أعلى المكدس إذا كان المكدس غير فارغ.
من الواضح أن التابع top يجب أن يكون تابعاً عضواً ثابتاً .

1- مدخل إلى المكدسات 12

```
StackElement top( ) const;
```

1-5- ضمن الصنف :

2-5- يتم وضع التابع في ملف التحقيق:

```
StackElement Stack::top( ) const
{if (myTop >=0)return myArray[myTop];
cerr<<"*** stack is empty or underflow *** \n";}
```

3-5- ننادي التابع من ملف التابع الرئيس main وفق التالي:

```
cout<<"top value: "<<s.top( )<<endl;
```

6 - تابع تابع الحذف من المكدس pop: يمكن تعريف عملية الحذف من المكدس pop على أنها تتلقى كبارامتر لها غرض من نوع Stack (ربما بشكل ضمني) وتنتج إزالة القيمة المخزنة في أعلى المكدس وإذا كان المكدس فارغاً تعيد رسالة خطأ.

كما هو ملاحظ فإن هذا التابع يجب أن يكون تابعاً عضوا غيرثابت

```
void pop( );
```

1-6- ضمن الصنف :

2-6- وتكون خوارزمية إخراج قيمة من المكدس كما يلي:

1. اختبار فيما إذا كان المكدس فارغاً.
2. إذا لم يكن المكدس فارغاً عندئذ:
   a. إنقاص قيمة myTop بمقدار 1.

وإلا عرض رسالة تفيد بأن المكدس فارغ.

يمكن تحقيق هذه العملية من خلال إضافة التصريح التالي إلى تعريف الصنف Stack

2-6- يتم وضع التابع في ملف التحقيق:

```
void Stack::pop()
{if (myTop>=0) myTop--;
Else cerr<<"*** stack is empty... can't remove a value ***\n";
}
```

6-3- نادي التابع من ملف التابع الرئيس main وفق التالي:

```
while(!s.empty())
{cout<<"poping "<<s.top()<<endl;
s.pop();
}
```

وبالتالي نكون قد وصلنا للصيغة النهائية التالية:

stack.h

ملف الصنف stack.h

```
#ifndef STACK
#define STACK
const int STACK_CAPACITY=128;
typedef int StackElement;

class Stack
{       /****  function members ****/

public:         Stack( ); bool empty( ) const ;
void push (const StackElement & value);
void display() const ;
StackElement top( ) const;
void pop();    /****  data members ****/

private:        StackElement myArray[STACK_CAPACITY];        int myTop;
}; //end of class declaration

inline Stack::Stack( ) { myTop= -1; }
inline bool Stack::empty( ) const { return (myTop==-1 ) ; }
#endif
```

stack.cpp

ملف تنفيذ توابع الصنف stack.h

```
#include<iostream>
#include<fstream>
#include "stack.h"
#include<string>
using namespace std;
 void Stack::push(const StackElement &value)
{       if (myTop<STACK_CAPACITY-1)
        {++myTop;               myArray[myTop]=value; }
        else
        cout<<"**** stack is full... can't add new value ***** \n
                must increase value of STACK_CAPACITY in stack.h\n";
}
void Stack::display() const
{       for (int i=myTop ; i>=0 ; i--)          cout<<myArray[i]<<endl;}
StackElement Stack::top( ) const
{       if (myTop >=0) return myArray[myTop];cout<<"***stack is empty*** \n";return 0 ;}
void Stack::pop()
{       if (myTop>=0)           myTop--;
        else            cout<<"*** stack is empty... can't remove a value ***\n";}
```

Main.cpp

ملف التابع الرئيس لاختبار الصنف stack.h

```
#include "stdafx.h«
#include "stack.h"
#include<iostream>
using namespace std;

int main( )
{   Stack s;
    cout<<boolalpha<<"s empty ? "<<s.empty()<<endl;

    for (int i=1;i<=4;i++)
     {int ep= i*2; s.push(ep); cout<<"pushing  "<<ep<<endl;}
    cout<<" s empty ? "<<s.empty()<<endl;

    while(!s.empty())
    { cout<<"poping "<<s.top()<<endl;
    s.pop(); }
    cout<<"s empty ? "<<s.empty()<<endl;
    system("pause");return 0;
}
```

نتيجة تنفيذ البرنامج الرئيس main

نتيجة تنفيذ البرنامج الرئيس main

```
s empty ? true
pushing  2
pushing  4
pushing  6
pushing  8
 s empty ? false
poping 8
poping 6
poping 4
poping 2
s empty ? true
Press any key to continue . . .
```

بعض التطبيقات على المكدسات1

التحويل من النظام العشري إلى النظام الثنائي

base-ten to base-two conversion

```
#include "stdafx.h"
#include<iostream>
#include "stack.h"
using namespace std;
int main( )
{       unsigned number,remainder;
        Stack stackofRemainders;
        char response;
        do
        {
                cout<<"Enter positive integer to convert: ";
                cin>>number;
                while ( number>0)
```

```
        {       remainder=number%2;
                stackofRemainders.push(remainder);
                number/=2;
        }
        cout<<"Base-two representation: ";
        while (!stackofRemainders.empty())
        {       remainder=stackofRemainders.top();
                stackofRemainders.pop();
                cout<<remainder;
        }
        cout<<endl;
        cout<<"\nMore (Y or N )?";
        cin>>response;
    }   while (response=='Y' || response=='y');
    return 0;
}
```

الأنواع الأكثر شيوعاً بلغة C++

نتيجة تنفيذ البرنامج الرئيس main

```
Enter positive integer to convert: 32
Base-two representation: 100000

More (Y or N )?y
Enter positive integer to convert: 444
Base-two representation: 110111100

More (Y or N )?y
Enter positive integer to convert: 8888
Base-two representation: 10001010111000

More (Y or N )?y
Enter positive integer to convert: 2048
Base-two representation: 100000000000

More (Y or N )?n
Press any key to continue . . .
```

Reverse Polish Notation(RPN)

الترميز البولندي العكسي

Reverse Polish Notation(RPN)

إن مهمة المنقح compiler هي توليد تعليمات الآلة اللازمة لتنفيذ التعليمات للبرنامج المكتوب بلغة عالية المستوى.

تكتب التعابير الحسابية في أغلب لغات البرمجة بالترميز infix أي (a*b+c), في هذا الترميز توضع الرموز لكل عملية ثنائية بين المعاملات, وتقوم الكثير من المنقحات بداية بتحويلها إلى الترميز postfix وفيه توضع المعاملات بعد الرموز, ومن ثم تقوم بتوليد تعليمات الآلة لتقييم هذه التعابير.

دعي الترميز postfix باسم الترميز البولندي العكسي Reverse Polish Notation(RPN), في هذا الترميز يمكن كتابة التعبير 2*(3+4) بالشكل التالي *+234.

تقييم التعابير المكتوبة بالترميز RPN: يمكن تبيان ذلك من خلال المثال التالي

1 5 + 8 4 1 - - *

هذا التعبير يقابل التعبير infix التالي:

(1+5)*(8-(4-1))

ويتم ذلك وفق الآلية التالية: يتم مسح التعبير من اليسار حتى يصادف معاملاً, عند هذه النقطة يكون هذا المعامل متوسطاً للقيمتين السابقتين, أي:

Reverse Polish Notation(RPN)

1 5 + 8 4 1 - - *

(1+5) 8 4 1 - - *

(1+5) 8 (4-1) - *

(1+5) (8-(4-1)) *

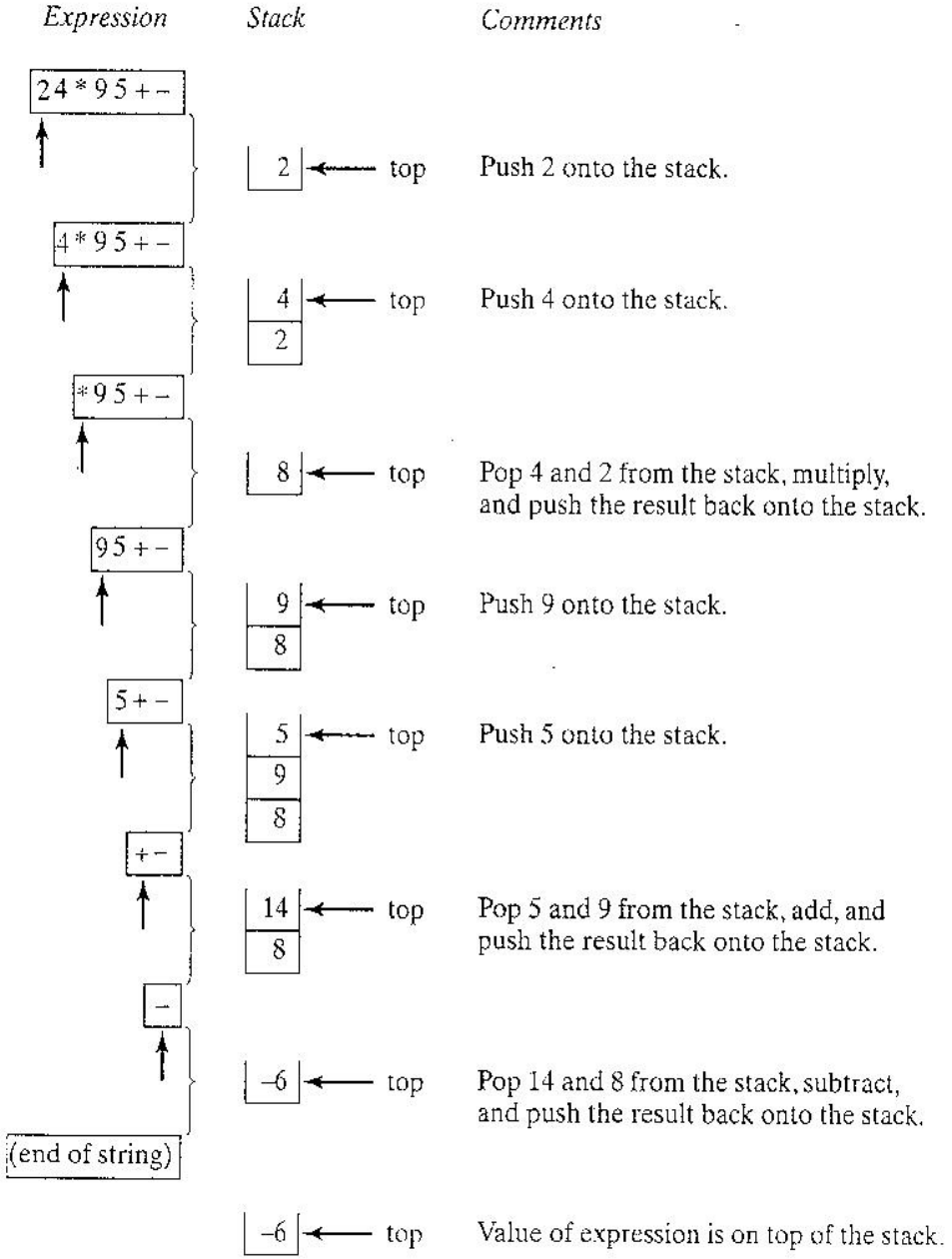
(1+5)*(8-(4-1))

إن طريقة التقييم لتعبير RPN تتطلب كون القيم مخزنة حتى تتم مصادفة معامل في عملية المسح من اليسار إلى اليمين, عند هذه النقطة يجب أن يتم جلب القيمتين الأخيرتين وتنفيذ المعامل عليهما. إن ذلك يتطلب بنية تعمل وفق المبدأ LIFO (أي المكدس) لتخزين القيم, حيث في كل مرة يصادف فيها أثناء المسح قيمة يتم إضافتها إلى المكدس, ثم عندما يصادف عملية يتم إخراج القيمتين الأخيرتين وإجراء العملية التي تمت مصادفتها عليهما ويتم إدخال الناتج إلى المكدس. يمكن تلخيص هذه العملية من خلال الخوارزمية التالية:

1. تهيئة مكدس فارغ.
2. تكرار ما يلي حتى يتم الوصول إلى نهاية التعبير:
   a. الحصول على الرمز التالي (ثابت, متحول, معامل حسابي) في التعبير RPN.
   b. إذا كان الرمز قيمة, يدفع إلى المكدس. وإذا كان معامل يتم القيام بما يلي:
      i. إخراج القيمتين العلويتين من المكدس (إذا لم يحتو المكدس قيمتين عندئذ يحصل خطأ يدل على تعبير RPN غير صالح ويتم إنهاء التنفيذ).
      ii. تطبيق المعامل على القيمتين المخرجتين.
      iii. إدخال الناتج في المكدس.
3. عندما يتم الوصول إلى نهاية التعبير, تكون قيمته موجودة على قمة المكدس. (يجب أن تكون هذه القيمة هي القيمة الوحيدة في المكدس).

يبين الشكل التالي خطوات هذه العملية:

بعض التطبيقات على المكدسات1



infix to RPN 1

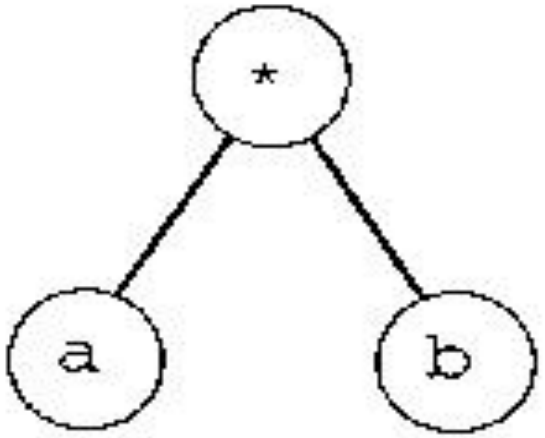
تحويل التعابير ذات الترميز infix إلى RPN:

هناك العديد من الطرق لتحويل التعابير ذات الترميز infix, نبدأ بطريقة تستخدم التمثيل الرسومي الخالي من الأقواس للتعابير على شكل شجرة التعابير expressions tree.

يتم تمثيل الأشجار باستخدام عقد nodes دائرية لتخزين البيانات وهذه العقد توصل فيما بينها من خلال خطوط مستقيمة تدعى الحواف edges.

إحدى العقد تدعى الجذر root ليس لها أي حواف قادمة إليها, ويمكن الوصول لكل عقدة انطلاقاً من الجذر من خلال مسار وحيد هو عبارة عن الحواف المتصلة, يوضع الجذر عادة في قمة الشجرة.

لتوضيح ذلك, ليكن لدينا شجرة التعبير التالية الخاصة بالتعبير a*b:



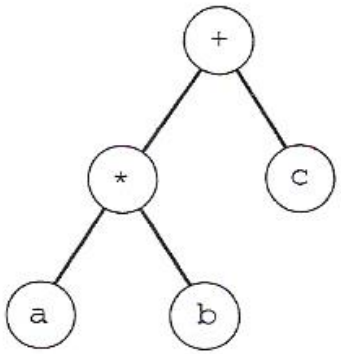
تحتوي العقدة الجذرية هنا المعامل* ولها عقدتين اثنتين, اليسرى تحتوي القيمة التي إلى يسار المعامل واليمنى تحتوي القيمة إلى يمين المعامل .

بعض التطبيقات على المكدسات1

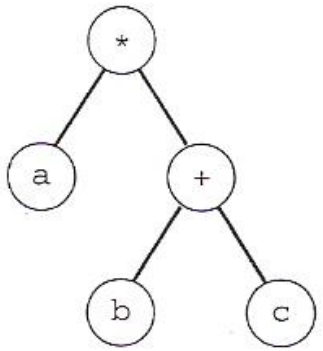
infix to RPN 2

فيما يلي بعض الأمثلة الأخرى:

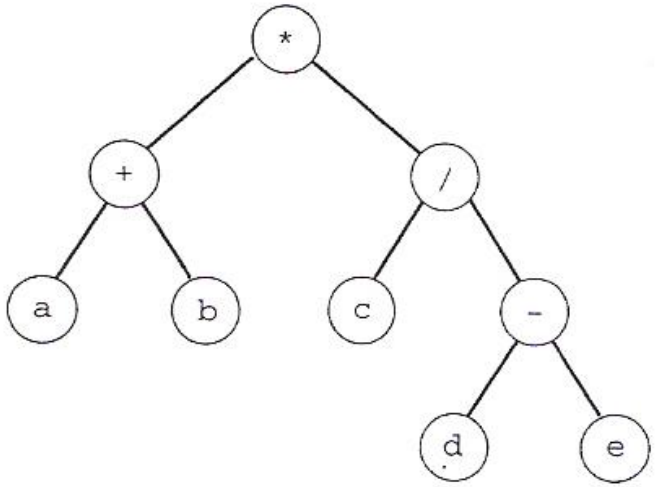
a * b + c



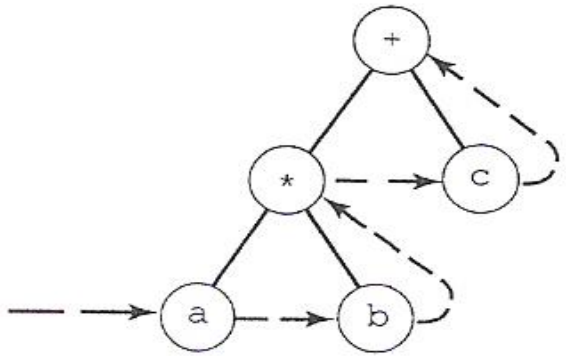
a * (b + c)



(a + b) * (c / (d - e))



a b * c +



للحصول على الترميز RPN المقابل للتعبير, يجب التجول عبر الشجرة بالترتيب التالي يسار،يمين, أب (Left-Right-Parent), هذا يعني أن لكل عقدة, لكي نستطيع المرور بها يجب أولاً المرور بالعقدتين الأبناء. يوضح الشكل التالي هذه العملية:

بعض التطبيقات على المكدسات1

infix to RPN 3

هناك طريقة ثانية لإجراء التحويل من infix إلى RPN, وهي مناسبة للتحويل اليدوي وهذه الطريقة تدعى طريقة الانتقال والحذف المحصورة كلياً بالأقواس:

1. ضع الأقواس بشكل كامل لكامل أجزاء التعبير.
2. استبدل كل قوس يميني بالمعامل المناسب من الداخل إلى الخارج).
3. احذف جميع الأقواس اليسارية.

فيما يلي بعض الأمثلة: 1- لتحويل التعبير a*b+c:

1. ((a * b) + c)
2. ((a * b) + c) ⟶ ((a b * c +
3. a b * c +

2- أما التعبير a*(b+c)

1. (a * (b + c))
2. (a * (b + c)) ⟶ ((a b c + *
3. a b c + *

بعض التطبيقات على المكدسات1

infix to RPN 3

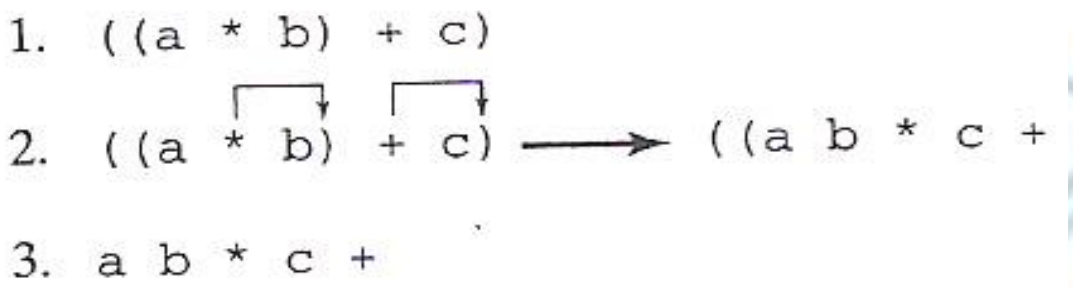
هناك طريقة ثانية لإجراء التحويل من infix إلى RPN, وهي مناسبة للتحويل اليدوي وهذه الطريقة تدعى طريقة الانتقال والحذف المحصورة كلياً بالأقواس:

1. ضع الأقواس بشكل كامل لكامل أجزاء التعبير.
2. استبدل كل قوس يميني بالمعامل المناسب من الداخل إلى الخارج).
3. احذف جميع الأقواس اليسارية.

فيما يلي بعض الأمثلة:

1- لتحويل التعبير a*b+c:

2-أما التعبير a*(b+c):



1. (a * (b + c))
2. (a * (b + c)) ⟶ (a * (x b c +
3. (a * (x b c +) ⟶ (x a (x b c + *
4. a b c + *

بعض التطبيقات على المكدسات1

infix to RPN 4

ومن أجل التعبير ((a+b)*(c/(d-e)):

1. ((a + b) * (c / (d - e)))

2. ((a + b) * (c / (d - e))) ⟶ ((a b + (c (d e - / *

3. a b + c d e - / *

وأخيراً, يمكن استخدام الطريقة التالية التي تستخدم المكدس للتحويل, ليكن لدينا التعبير التالي:

7+2*3

بمسح التعبير من اليسار إلى اليمين, نصادف أولاً 7 نقوم مباشرة بعرضها على الشاشة, ثم نصادف + يتم دفعها إلى المكدس, ثم نصادف 2 فنقوم بعرضها, في هذه اللحظة يجب أن نحدد فيما إذا كانت 2 هي القيمة اليمينية للمعامل + أو اليسارية للمعامل *, يكون ذلك بمقارنة المعامل الموجود في أعلى المكدس بالمعامل التالي *, وبما أن * ذو أولوية أعلى من + فإن 2 هي القيمة اليسارية للمعامل * وبالتالي يتم دفع المعامل * إلى أعلى المكدس ومن ثم عرض القيمة 3, بما أننا وصلنا إلى نهاية التعبير يتم إخراج القيم من المكدس وعرضها فنحصل على التعبير المطلوب.

بعض التطبيقات على المكدسات1

infix to RPN 4

ومن أجل التعبير ((a+b)*(c/(d-e)):

1. ((a+b)*(c/(d-e)))→((a+b)*(c/($_x$de-))

2. ((a+b)*(c/($_x$de-))⟶((a+b)*($_x$c/($_x$de-/)

3. ((a+b)*($_x$c/($_x$de-/)⟶($_x$(a+b) ($_x$c/($_x$de-/*

4. ($_x$(a+b) ($_x$c/($_x$de-/* ⟶($_x$($_x$ab+ ($_x$c/($_x$de-/* ⟶ab+cde-/*

وأخيراً, يمكن استخدام الطريقة التالية التي تستخدم المكدس للتحويل, ليكن لدينا التعبير التالي: 7+2*3

بمسح التعبير من اليسار إلى اليمين, نصادف أولاً 7 نقوم مباشرة بعرضها على الشاشة, ثم نصادف + يتم دفعها إلى المكدس, ثم نصادف 2 فنقوم بعرضها, في هذه اللحظة يجب أن نحدد فيما إذا كانت 2 هي القيمة اليمينية للمعامل + أو اليسارية للمعامل *, يكون ذلك بمقارنة المعامل الموجود في أعلى المكدس بالمعامل التالي *, وبما أن * ذو أولوية أعلى من + فإن 2 هي القيمة اليسارية للمعامل * وبالتالي يتم دفع المعامل * إلى أعلى المكدس ومن ثم عرض القيمة 3, بما أننا وصلنا إلى نهاية التعبير يتم إخراج القيم من المكدس وعرضها فنحصل على التعبير المطلوب.

infix to RPN 5

بعض التطبيقات على المكدسات1

الخرج	المكدس
7	+
72	* +
723	* +
723*	+
723*+	

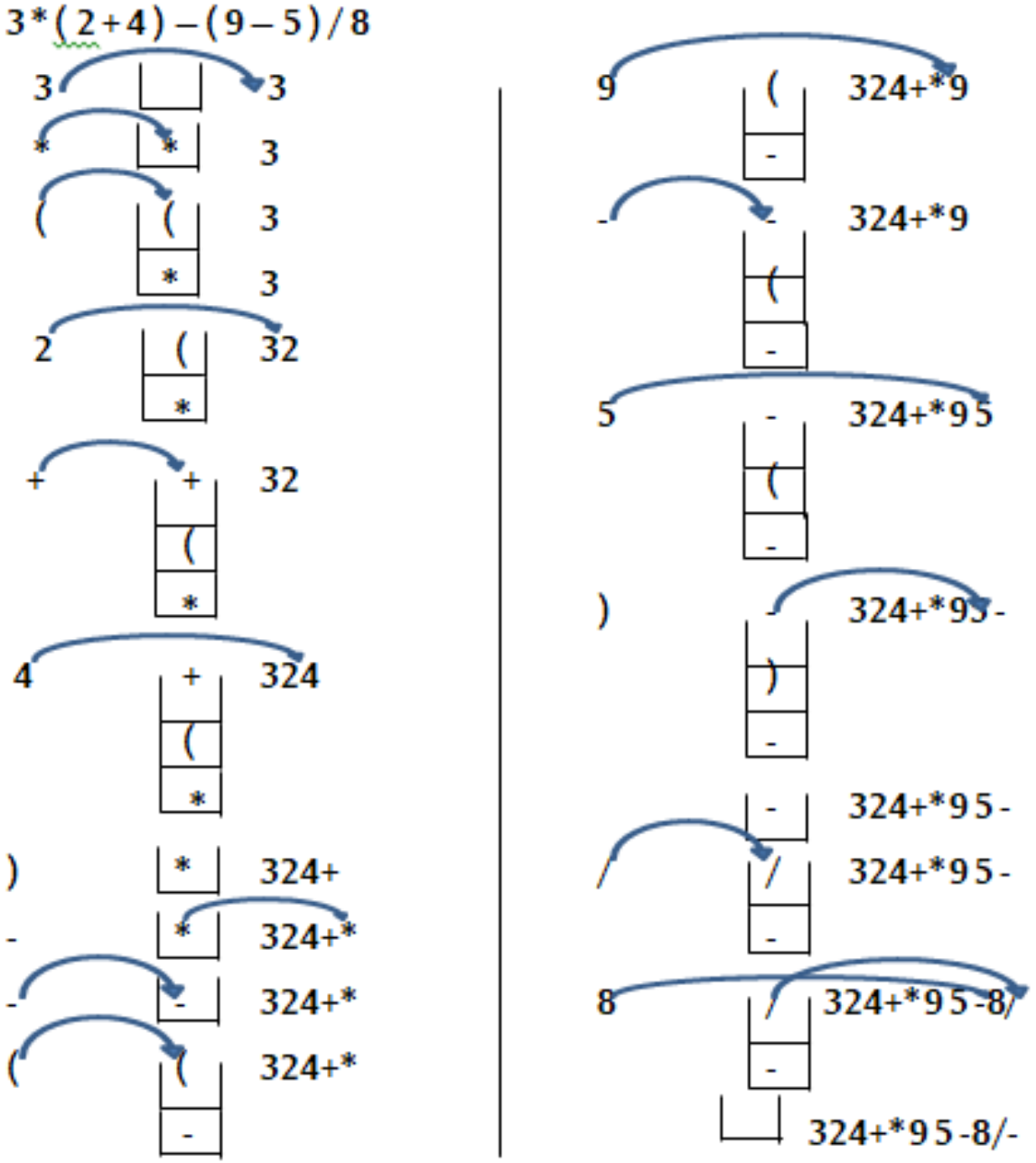
ومن أجل حساب قيمة التعبير 7+2*3

infix to RPN 6

يمكن تلخيص خوارزمية التحويل من الترميز infix إلى الترميز RPN كما يلي:

1. تهيئة مكدس فارغ للمعاملات.
2. طالما أنه ليس هناك خطأ ولم نصل إلى نهاية التعبير الحسابي نقوم بما يلي:
   a. جلب الرمز التالي (ثابت, متحول, معامل, قوس يميني, قوس يساري).
   b. إذا كان الرمز:
      i. قوس يساري: قم بدفعه إلى المكدس.
      ii. قوس يميني: قم بإخراج القيم من المكدس وقم بعرضها على الشاشة لحين الوصول إلى قوس يساري فلا تقم بعرضه.
      iii. معامل: إذا كان المكدس فارغاً أو الرمز ذو أولوية أعلى من المعامل الموجوده في قمة المكدس, قم بدفع الرمز في المكدس.
      وإلا أخرج المعامل الموجوده في أعلى المكدس واعرضها. (ملاحظة القوس اليساري في المكدس يملك أولوية دنيا).
      iv. قيمة: قم بعرضها.
3. عند الوصول إلى نهاية التعبير قم بإخراج وعرض عناصر المكدس إلى أن يصبح فارغاً.

يمكن كتابة البرنامج التالي بلغة C++ لحل هذه المسألة:


3*(2+4)-(9-5)/8
3 3
* * 3
((3
* 3
2 (32
*
+ + 32
(
*
4 + 324
(
*
) * 324+
- * 324+*
- - 324+*
((324+*
-
9 (324+*9
-
- 324+*9
(
-
5 - 324+*95
(
-
) - 324+*95-
)
-
- 324+*95-
/ / 324+*95-
-
8 / 324+*95-8/
-
324+*95-8/-

program infix to RPN 1

برنامج تحويل infix to RPN

```
#include "stdafx.h"
#include<string>
#include<cassert>
#include "Stack.h"
using namespace std;
string RPN(string exp);
int main()
{       string exp;
        cout<<"NOTE:Enter # for infix expression to stop\n ";
        for (;;)
        {       cout<<"\nInfix Expression ?";
                cin>>exp;
                if (exp=="#") break;
                cout<<"RPN Expression is "<<RPN(exp)<<endl;
        }
system("pause");    return 0;
}
```

program infix to RPN 2

برنامج تحويل infix to RPN

```
string RPN(string exp)
{       char token,topToken;
        Stack opStack;
        string RPNexp;
        const string BLANK=" ";
        RPNexp="";
        for (int i=0;i<exp.length();i++)
        {       token=exp[i];
                switch(token)
                {case ' ':break;
                case '(':opStack.push(token);    break;
                case ')':for ( ; ; )
                                {assert(!opStack.empty());
                                topToken=opStack.top(); opStack.pop();
                                if (topToken== '(') break;
                                  RPNexp.append(BLANK+topToken);
                                } break;
```

```
        case '+':case '-':case '*':case '/':
              for( ; ; )
              {     if (opStack.empty() ||opStack.top()=='(' ||
                    (token=='*' || token=='/') &&
                    (opStack.top()=='+' || opStack.top()=='-'))
                    {opStack.push(token);break;
                    }
              else  {
                    topToken=opStack.top();   opStack.pop();
                    RPNexp.append(BLANK+topToken);
                    }
              }
        break;
        default:RPNexp.append(BLANK+token);
        }
}
```

```
for ( ; ; )
{
        if (opStack.empty()) break;
        topToken=opStack.top();
        opStack.pop();
        if (topToken!='(')
                RPNexp.append(BLANK+topToken);
        else
        {
                cout<<"**** ERROR IN INFIX EXPRESSION *** \n";
                break;
        }
}
return RPNexp;
}
```

نتيجة تنفيذ البرنامج لتحويل infix to RPN

```
"C:\stack11\Debug\stack11.exe"
NOTE:Enter # for infix expression to stop

Infix Expression ?a+b
RPN Expression is  a b +

Infix Expression ?a*b+c
RPN Expression is  a b * c +

Infix Expression ?a*(b+c)
RPN Expression is  a b c + *

Infix Expression ?(a+b)*(c/(d-e)):
RPN Expression is  a b + c d e - / : *

Infix Expression ?a-(b-(c-(d-(e-f))))
RPN Expression is  a b c d e f - - - - -

Infix Expression ?a-b-c-d-e-f
RPN Expression is  a b - c - d - e - f -

Infix Expression ?a+b+d
RPN Expression is  a b + d +

Infix Expression ?#
Press any key to continue
```

النهاية

انتهت المحاضرة الرابعة