IUT AMU, site d'Arles	BUT Informatique – 1ère année
NOM :	PRENOM : ....

Contrôle – R1.01 - du 28/01/2025

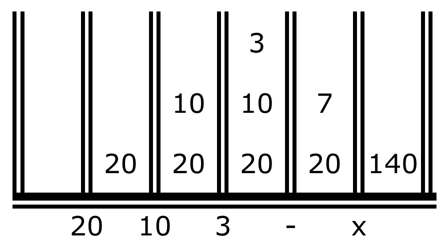
Durée 1h30, sans document, sans machine

- Lisez tout le sujet avant de commencer
- Vous pouvez répondre au crayon tant que la copie reste parfaitement lisible.
- La qualité de la rédaction et la rigueur des raisonnements seront prises en compte dans la notation.
- En plus de votre code, donnez des commentaires sur ce que vous faites, et si vous n'avez pas toutes les étapes d'un problème, rajoutez un texte explicatif de ce qui manque.
- La notation tiendra compte de la justesse et de la qualité de votre code, de la pertinence de vos choix en matière de structures de données et de la pertinence du choix des types.
- Le barème est donné à titre indicatif.

1. Programmation d'un interpréteur NPI (20 points)

La firme Hewlett-Packard s'est imposée dans les années 1970 auprès de la communauté scientifique et étudiante avec l'introduction de sa calculatrice scientifique HP-35. Elle sortira d'ailleurs en 1988 la HP-42s encore considérée aujourd'hui par les passionnés comme la meilleure calculatrice HP jamais produite. La particularité de ces calculatrices est qu'elles utilisent la notation polonaise inversée (NPI). Celle-ci, appelée aussi notation post-fixée car l'opérateur est noté derrière les opérandes, permet d'écrire les formules arithmétiques sans utiliser les parenthèses et ce sans ambiguïté. Par exemple, l'expression « $20 \times (10 - 3)$ » s'écrit « $20\ 10\ 3 - \times$ ».

L'évaluation de ces expressions est réalisée sur les calculatrices à l'aide d'une pile. Le principe consiste à empiler les opérandes (les nombres) et à chaque fois que l'on rencontre une opération on dépile les opérandes nécessaires (2 pour un opérateur tel que l'addition) et on empile le résultat. Le schéma sur la droite illustre ce principe pour l'expression « $20\ 10\ 3 - \times$ ».



Passionné par la notation polonaise inversée et la HP-42s, vous souhaitez programmer votre propre interpréteur NPI en C++. Pour cela, et comme prérequis, vous devez implanter une pile dont le type abstrait est le suivant :

TA $Stack$
Utilise: $Element$, $Boolean$
Opération(s):
$newStack: \ \rightarrow Stack$
$push: Stack \times Element \rightarrow Stack$
$pop: Stack \rightharpoonup Stack$
$top: Stack \rightharpoonup Element$
$isEmpty: Stack \rightarrow Boolean$

Pré-condition(s): $\forall s \in Stack$
$defini(pop(s)) \Rightarrow \neg isEmpty(s)$
$defini(top(s)) \Rightarrow \neg isEmpty(s)$

Axiomes: $\forall e \in Element, \forall s \in Stack$
$isEmpty(newStack) \ = \ true$
$isEmpty(push(s,e)) = false$
$top(push(s,e))) = e$
$pop(push(s,e)) = s$

2.1. Gestion de la pile (15 points)

Vous avez la brillante idée d'utiliser comme structure de données une structure stockant un tableau de doubles `data` qui sera alloué dynamiquement, le nombre maximum `maxSize` d'éléments empilables, et le nombre `nbElements` d'éléments empilés.

a) Ecrivez en C++ la définition d'une structure de données `Stack` respectant la spécification précédente.

b) Ecrivez la définition du constructeur (nommé ici `newStack`) qui prend une taille en paramètre et retourne une pile valide. Si aucune taille n'est spécifiée alors initialement la pile pourra contenir au plus 10 éléments.

c) Dans la mesure où nous utilisons dans notre structure un tableau dynamique, il est nécessaire d'écrire une fonction qui permet de détruire proprement une pile. Ecrire la définition du sous-programme nommé `deleteStack`, qui réalise cette opération.

d) Pour implanter les opérations sur la pile vous avez besoin de savoir si la pile est vide ou non. Ecrivez la définition en C++ de l'observateur `isEmpty`.

e) Vous aurez également besoin de connaitre le nombre d'éléments dans la pile. Ecrivez la définition du sous-programme size dont le prototype est le suivant `size_t size(const Stack& s)` où le type `size_t` est défini dans la norme C++ comme équivalent à `unsigned long int`.

f) Une des opérations fondamentales sur la pile consiste à ajouter un élément au sommet. Ecrivez la définition des sous-programmes `resize` et `push` qui permettent respectivement de doubler la taille du conteneur lorsqu'elle est insuffisante et d'ajouter un élément sur une pile. C'est `push` qui aura à sa charge d'appeler `resize` en cas de besoin.

2.2. Utilisation de la pile (points 5)

Après une courte recherche sur internet vous avez trouvé le code source du programme principal d'un interpréteur que vous avez adapté pour qu'il utilise votre pile :

```
int main() {
  Stack myStack = newStack();
  string input;
  showUsage();
```

Initialisation d'une pile nommée myStack.

```
  do {
    showStack(myStack);

    cout << "command: ";
    cin >> input;
```

Boucle principale.
Affichage du contenu de la pile.
Saisie de l'entrée de l'utilisateur.

```
    try {
      size_t sz;
      double n = stod(input, &sz);

      if (sz > 0)
        push(myStack, n);
    }
```

On essaie d'interpréter l'entrée comme un nombre.
Si on y arrive on l'ajoute à la pile.

```
    catch (const invalid_argument& ia) {
      switch (input[0]) {
      case 'q':
        cout << "exit" << endl;
        deleteStack(myStack);
        break;
      case 'c':
        emptyStack(myStack);
        break;
      case 'u':
        showUsage();
        break;
      case '+':
      case '-':
      case '/':
      case '*':
        eval(myStack, input[0]);
        break;
      default: cout << "Bad command" << endl;
      }
    }
```

Si on n'arrive pas à lire un nombre, on regarde si le 1er caractère de l'entrée est une commande ou une opération.

*Si on trouve une commande (q, c ou u) ou un opérateur (+, -, /, *), on l'applique.*

```
  } while (input[0] != 'q');
```

On quitte si la commande vaut 'q'.

```

  return 0;
}
```

Remarque : ici on utilise le bloc d'instruction `try … catch…` que vous étudierez plus tard. La seule chose à comprendre est que si l'appel à `stod(input, &sz)` qui permet de convertir une chaîne en double échoue alors c'est le code contenu dans le bloc `catch` qui sera exécuté.

a) Ecrivez l'implantation du sous-programme `eval` dont le prototype est le suivant `void eval(Stack& s, char op)`. Ce sous-programme prend en paramètre une pile `s` et un opérateur `op` sous la forme d'un caractère. Il se charge de lire sur `s` les opérandes et d'appliquer l'opération correspondant à l'opérateur contenu dans `op` provenant du programme principal. Pour cette question vous ferez l'hypothèse que vous avez à votre disposition toutes les opérations décrites dans le TAD Stack.