Correction de Rush-02

Fonction / Élément	Version Originale (ce qui n'allait pas)	Nouvelle Version (Corrigée)
main	Coquille Vide. Déclarait des variables inutilement. Les appels à atoi ne servaient à rien. Ne se connectait à AUCUN autre fichier (conditioners_bridge, printer, etc.). Ne produisait aucun résultat.	Chef d'Orchestre. Gère les arguments (argc, argv). Appelle atoi_char_validator pour nettoyer et valider l'entrée. Connecte le pipeline : Appelle nbr_from_str puis print_tokens. Gère les cas d'erreur (Error, Dict Error).
atoi_char	Buguée et peu utile. Tentait de copier des chiffres mais utilisait un seul index, ce qui était un bug. Le but n'était pas clair.	Transformée en Validateur. Rôle clair : Valider et nettoyer la chaîne d'entrée. Corrigée : Utilise deux index (i et j) pour lire et écrire correctement. Retourne 1 (succès) ou 0 (échec), ce qui est utilisable dans un if.
is_space	Fonctionnelle. Faisait son travail correctement.	Inchangée et Conservée. Toujours utile pour la fonction de validation.
stock_name	Cassée et Inutile. Provoquait un crash garanti (erreur de segmentation) car elle utilisait un pointeur non initialisé. - Ne servait à rien dans la logique du projet.	Supprimée. Elle n'avait aucune raison d'exister.
malloc / free	Présents mais Inutiles. main allouait de la mémoire pour gnrltab, qui n'était jamais utilisé de manière productive.	Supprimés. Les données (clean_number, tokens) sont stockées dans des tableaux locaux sur la stack. C'est plus simple et plus sûr.
Prototypes	Absents. Il n'y avait aucun prototype pour les fonctions des autres fichiers.	Ajoutés. Des déclarations pour nbr_from_str et print_tokens sont ajoutées pour que main sache que ces fonctions existent.

Synthèse Globale

L'original rush-02.c était une collection de pièces détachées. La fonction main était une coquille vide qui n'appelait aucun des autres modules. Le programme ne faisait donc rien, même si les autres fichiers (.c) étaient corrects.

Le nouveau rush-02.c est le cerveau du programme. Sa fonction main a été remplie pour devenir le "chef d'orchestre" : elle valide l'entrée, appelle le module de conversion, puis le module d'impression, gérant ainsi tout le flux de données du début à la fin.

FLUX ORIGINAL :
main -> (rien)

FLUX NOUVEAU :
main -> atoi_char_validator -> nbr_from_str -> print_tokens

Le changement principal est donc la **création d'un flux de données logique** au sein de la fonction main, transformant le fichier rush-02.c du maillon le plus faible du projet en son composant le plus central.

Voyage d’un nombre de main à print

Nous allons suivre le voyage de ton nombre, 357951741258963, à travers tout le programme, du début à la fin.

Étape 1 : Le Terminal -> main

Nous envoyons la commande :

./name 357951741258963

Le programme se lance. La fonction main de rush-02.c démarre.

argc vaut 2.

argv[0] est "./name".

argv[1] est la chaîne de caractères "357951741258963".

Étape 2 : main -> atoi_char

main appelle la fonction de validation :

atoi_char("357951741258963", clean_number)

atoi_char se met au travail :

Il ne trouve aucun espace à ignorer.

Il ne trouve aucun signe + ou -.

Il voit que la chaîne commence bien par des chiffres.

Il copie un par un tous les chiffres de argv[1] dans le buffer clean_number.

Il termine le buffer avec un \0.

Résultat : atoi_char_validator retourne 1 (succès). La variable clean_number dans main contient maintenant la chaîne propre "357951741258963".

Étape 3 : main -> nbr_from_str (le début du voyage logique)

main appelle la fonction pont :

nbr_from_str("357951741258963", tokens, 1024)

nbr_from_str appelle pack_filler (de number_conditioner.c) :

pack_filler prend la chaîne "357951741258963".

Il la découpe de droite à gauche en paquets de 3 chiffres.

Il remplit un tableau packs avec les valeurs numériques de ces paquets.

Résultat : Le tableau packs contient {963, 258, 741, 951, 357}. La variable pmax (le nombre de paquets) est 5.

nbr_from_str appelle ensuite ft_transmitter avec ce tableau packs. C'est là que tout se joue. ft_transmitter va boucler sur les paquets, en commençant par la fin (les plus grandes valeurs).

Tour 1 (i=4) : Paquet 357 (Trillions)

Appelle packsofthree_emitter(357). La liste de jetons s'agrandit avec : {3, 100, 50, 7}.

Appelle scale_code_emitter(357, 4). La liste de jetons s'agrandit avec le jeton d'échelle : {-4}.

Tour 2 (i=3) : Paquet 951 (Milliards/Billions)

Appelle packsofthree_emitter(951) -> Ajoute les jetons : {9, 100, 50, 1}.

Appelle scale_code_emitter(951, 3) -> Ajoute le jeton d'échelle : {-3}.

Tour 3 (i=2) : Paquet 741 (Millions)

Appelle packsofthree_emitter(741) -> Ajoute les jetons : {7, 100, 40, 1}.

Appelle scale_code_emitter(741, 2) -> Ajoute le jeton d'échelle : {-2}.

Tour 4 (i=1) : Paquet 258 (Mille/Thousands)

Appelle packsofthree_emitter(258) -> Ajoute les jetons : {2, 100, 50, 8}.

Appelle scale_code_emitter(258, 1) -> Ajoute le jeton d'échelle : {-1}.

Tour 5 (i=0) : Paquet 963 (Unités)

Appelle packsofthree_emitter(963) -> Ajoute les jetons : {9, 100, 60, 3}.

Appelle scale_code_emitter(963, 0). L'index d'échelle est 0, donc la fonction ne fait rien.

Résultat : nbr_from_str retourne 24 (le nombre total de jetons). Le tableau tokens dans main est maintenant plein :

{3, 100, 50, 7, -4, 9, 100, 50, 1, -3, 7, 100, 40, 1, -2, 2, 100, 50, 8, -1, 9, 100, 60, 3}

Étape 4 : main -> print_tokens (la traduction finale)

main appelle la fonction d'impression :

print_tokens(tokens, 24)

print_tokens (de printer.c) se met au travail :

Il appelle read_dict et charge le fichier numbers.dict en mémoire.

Il entre dans sa boucle et commence à traduire chaque jeton du tableau tokens en appelant token_reader.

Jeton	token_reader trouve...	print_tokens écrit...
3	3: three	"three"
100	100: hundred	" hundred"
50	50: fifty	" fifty"
7	7: seven	" seven"
-4	1000... (13 chiffres): trillion	" trillion"
9	9: nine	" nine"
100	100: hundred	" hundred"
50	50: fifty	" fifty"
1	1: one	" one"
-3	1000... (10 chiffres): billion	" billion"
7	7: seven	" seven"
100	100: hundred	" hundred"
40	40: forty	" forty"
1	1: one	" one"
-2	1000000: million	" million"
2	2: two	" two"
100	100: hundred	" hundred"
50	50: fifty	" fifty"
8	8: eight	" eight"
-1	1000: thousand	" thousand"
9	9: nine	" nine"
100	100: hundred	" hundred"
60	60: sixty	" sixty"
3	3: three	" three"

Après la boucle, print_tokens ajoute un dernier saut de ligne. Ce qui est affiché à l'écran est la concaténation de tous les mots :

three hundred fifty seven trillion nine hundred fifty one billion seven hundred forty one million two hundred fifty eight thousand nine hundred sixty three