ＰＵＳＨ＿ＳＷＡＰ　メイン処理のやさしい解説

このドキュメントは、ｐｕｓｈ＿ｓｗａｐ　プログラムのエントリーポイント（ｍａｉｎ　関数まわ

－－－

全体のながれ

- 入力があるか確認　：　コマンドライン引数の個数を見て、並べ替える数字が１つも渡されてい
- スタック（リスト）の準備　：　ｓｔａｃｋ＿ａ　と　ｓｔａｃｋ＿ｂ　という　２　本のリン
- 文字列を数値リストに変換　：　ｐａｒｓｅ＿ａｒｇｓ　で文字列の配列を読み取り、ｓｔａｃ
- すでに並んでいないかチェック　：　ｉｓ＿ｓｏｒｔｅｄ　で　ｓｔａｃｋ＿ａ　が最初から昇
- リストの長さを数えて順位づけ　：　ｓｔａｃｋ＿ｓｉｚｅ　で要素数を数え、ｉｎｄｅｘ＿ｓ
- 要素数に合わせてソート方法を選択　：　ｃｈｏｏｓｅ＿ｓｏｒｔ
  がリストの大きさに応じた関数（ｓａ/ｓｏｒｔ＿ｔｈｒｅｅ/ｓｏｒｔ＿ｓｍａｌｌ/ｔｕｒ
- 後始末　：　使ったメモリを　ｆｒｅｅ＿ｓｔａｃｋ　で解放し、プログラムを終了します。

－－－

各処理の役わり

１．　ＭＡＩＮ　関数　（ＰＵＳＨ＿ＳＷＡＰ．Ｃ）

- 目的　：　プログラム全体の入り口で、処理の順番を管理します。
- 初心者ポイント　：　まず　ａｒｇｃ（引数の数）を調べ、仕事がなければすぐ終わる、という

２．　ＰＡＲＳＥ＿ＡＲＧＳ　（ＰＡＲＳＥ．Ｃ）

- 目的　：　受け取った文字列を整数に変換し、ｓｔａｃｋ＿ａ　というリンクトリストへ積み込
- 安全対策　：
- ｉｓ＿ｎｕｍｂｅｒ　で数字かどうかをチェック。
- ｃｈｅｃｋ＿ｏｖｅｒｆｌｏｗ　で　ｉｎｔ　の範囲を越えていないか確認。
- 重複チェック　（ｈａｓ＿ｄｕｐｌｉｃａｔｅｓ）　で同じ数字が複数ないか確認。
- 初心者ポイント　：　途中でエラーが見つかったら　ｅｒｒｏｒ＿ｅｘｉｔ　でメッセージを出

３．　ＩＳ＿ＳＯＲＴＥＤ　（ＵＴＩＬＳ．Ｃ）

- 目的　：　ｓｔａｃｋ＿ａ　の先頭から順番に値を見て、常に「次の値が現在の値以上」になっ
- 初心者ポイント　：　ｗｈｉｌｅ　ループでリストを１つずつ進めながら比較する典型的なパタ

４．　ＳＴＡＣＫ＿ＳＩＺＥ　と　ＩＮＤＥＸ＿ＳＴＡＣＫ

- ｓｔａｃｋ＿ｓｉｚｅ　（ｓｔａｃｋ．ｃ）　：　リストを最後までたどって要素数を数えるだ
- ｉｎｄｅｘ＿ｓｔａｃｋ　（ｉｎｄｅｘ＿ｓｅｔ．ｃ）　：　リスト全体を何度か走査しながら
  を振り分けます。
- 初心者ポイント　：　値そのものではなく「順位」を付けておくと、後でソート手順を決めやす

５．　ＣＨＯＯＳＥ＿ＳＯＲＴ（ＰＵＳＨ＿ＳＷＡＰ．Ｃ　内の静的関数）

- 目的　：　要素数に合わせて最適なソート戦略を選ぶ分岐です。
- ２個だけなら　ｓａ（ｓｗａｐ　操作）で済む。
- ３個なら　ｓｏｒｔ＿ｔｈｒｅｅ。
- ５個以下なら　ｓｏｒｔ＿ｓｍａｌｌ。

・　それ以上は本格的な　ｔｕｒｋ＿ｓｏｒｔ　に任せます。
・　初心者ポイント　：　ｉｆ　／　ｅｌｓｅ　ｉｆ　で条件を並べるときは、「小さいケースから

６．　後始末　（ＦＲＥＥ＿ＳＴＡＣＫ）
・　目的　：　ｓｔａｃｋ＿ａ　と　ｓｔａｃｋ＿ｂ　に残っているノードを順に　ｆｒｅｅ　して
・　初心者ポイント　：　ｆｒｅｅ＿ｓｔａｃｋ　の中で　ＮＵＬＬ　チェックをしているので、呼

- - -

まとめ
・　ｍａｉｎ　は「入力　→　変換　→　チェック　→　ソート　→　片付け」という一本道です。
・　各補助関数は、１つの役割に集中して作られており、読み解きやすくなっています。
・　エラー処理やメモリ解放が関数にまとめられているので、安全な作りになっているのがポイント

初心者の方は、ここで紹介した関数を追いながら実際のコードを開き、処理の流れを追ってみると理