

| Fichier .h (ex: stocks.h) | Fichier .cpp (ex: stocks.cpp) | Fichier main.cpp |
|---|---|-----------------------------|
| #ifndef ... / #define ... / #endif | #include <iostream> | #include <iostream> |
| #include <vector> | #include "stocks.h" | #include "stocks.h" |
| using namespace std; | // (pas de using namespace std ici) | using namespace std; |
| const int NBI = 8; | | int main() { ... } |
| Prototypes des fonctions | Définitions (code) des fonctions | Appels aux fonctions |
| void afficherStocks(const vector<int>& stocks); | void afficherStocks(const vector<int>& stocks){ ... } | afficherStocks(monVecteur); |

.h (header): #ifndef, #include <vector>, using namespace std;, const int NBI=8;, Prototypes (void func(..));.cpp (code): #include <iostream>, #include "file.h", Définitions (void func(...){ ... }). main.cpp (principal): #include <iostream>, #include "file.h", using namespace std;, int main() { ... }.

std::vector

- #include <vector> vector<int> v = {1, 2, 3}; Accès: v[i] | Taille: v.size()
- Ajout fin: v.push_back(val); | Suppr fin: v.pop_back(); Boucle: for (int i=0; i < v.size(); i++) { ... }

Fonctions (Passage vector) Modifier: void func(vector<int>& v);

- **Lire (efficace):** void func(const vector<int>& v);

afficher stocks : for (int i=0; i < stocks.size(); i++) { if (stocks[i] == 0) { cout << "Pas d'ingr " << i << endl; }

else { cout << stocks[i] << " gr d'ingr " << i << endl; }}

ajuster stocks : for (int i=0; i < stocks.size(); i++) { // Négatifs -> 0 if (stocks[i] < 0) { stocks[i] = 0; }}

while (stocks.size() < NBI) { stocks.push_back(0); } // Trop court while (stocks.size() > NBI) { stocks.pop_back(); } // Trop long

main.cpp / Tests

- **main:** ajusterStocks(stocks); if (recetteFaisable(stocks, r1)) { faireRecette(stocks, r1); } else { ... } afficherStocks(stocks);
- **mainTest:** vector<int> t1 = {...}; vector<int> attendu = {...}; ajusterStocks(t1); if (t1 == attendu) { ... }

algo :Structure Algo

- Algorithme Nom / Constantes / Variables
- Fonction Nom(param) : TypeRetour ... Retourner val ... FinFonction
- Procédure Nom(E/S param) ... FinProcédure
- Début ... Fin

Patrons Code Algo

- caractereValide(c: car) : bool Retourner (c='H' OU c='B' OU c='G' OU c='D' OU c='S')
- saisirDirection() : car

Saisir dir TANT QUE (NON caractereValide(dir)) Afficher "Erreur" / Saisir dir
 FIN TANT QUE Retourner dir

deplacer le robot SI dir='H' ALORS y <- y+1 SINON SI dir='B' ALORS y <- y-1 SINON SI dir='D' ALORS x <- x+1
 SINON SI dir='G' ALORS x <- x-1 FIN SI
 boucle de jeu / : xR <- 1, yR <- 2, nbPas <- 0 X_ARR <- 4, Y_ARR <- 1 estArrive <- (xR=X_ARR ET
 yR=Y_ARR) SI (NON estArrive) ALORS direction <- saisirDirection() FIN SI

TANT QUE (direction != 'S' ET NON estArrive) deplacerRobot(xR, yR, direction) nbPas <- nbPas + 1
 estArrive <- (xR=X_ARR ET yR=Y_ARR) SI (NON estArrive) ALORS direction <- saisirDirection() FIN SI
 FIN TANT QUE
 SI (estArrive) ALORS Afficher "Bravo...", nbPas SINON Afficher "Arrêt...", nbPas FIN SI

obstacle , tableau : i <- 0, trouve <- faux TANT QUE (i < taille(tab) ET NON trouve)
 SI (tab[i]=x ET tab[i+1]=y) ALORS trouve <- vrai FIN SI i <- i + 2 // !! Important
 FIN TANT QUE Retourner trouve

Nouvelle Logique (Q1.6):

- **Avant boucle:** estSurObs <- surUnObstacle(tabObs, xR, yR)
- **Saisie avant boucle:** SI (NON estArrive ET NON estSurObs) ALORS dir <- saisirDirection() FIN SI
- **Condition boucle:** TANT QUE (dir != 'S' ET NON estArrive ET NON estSurObs)
- **Dans boucle:** deplacer..., nbPas++, estArrive <- ..., estSurObs <- ...
- **Saisie dans boucle:** SI (NON estArrive ET NON estSurObs) ALORS dir <- saisirDirection() FIN SI
- **Sortie boucle:** SI (estArrive) ... SINON SI (estSurObs) ... SINON ... FIN SI