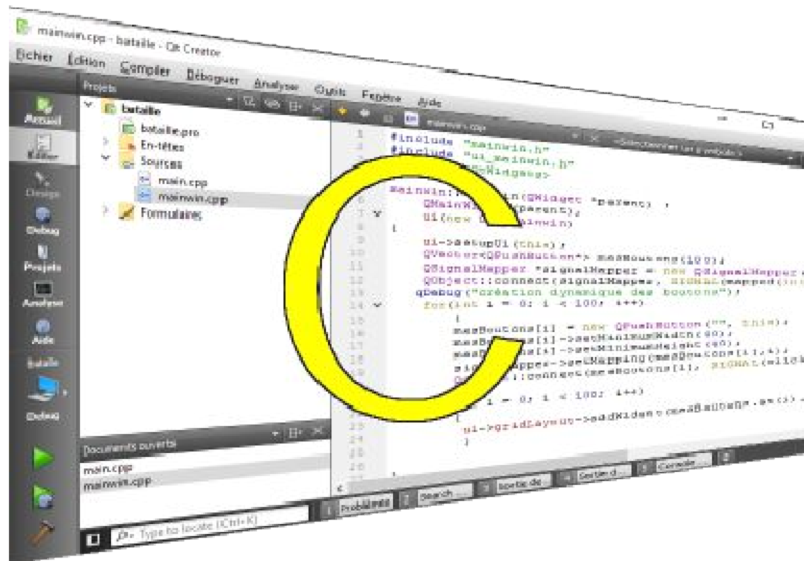
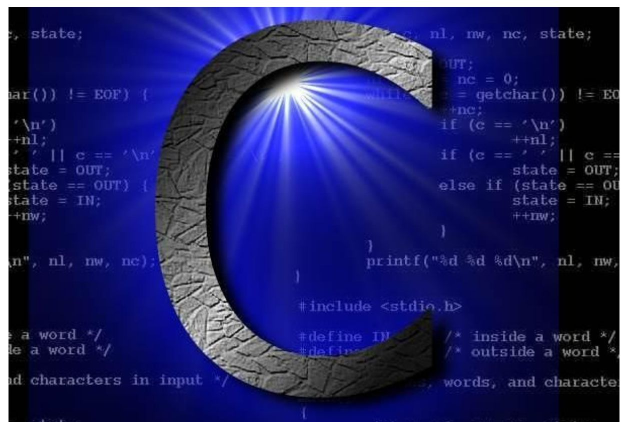
INTRODUCTION AU C ET C++



Baccalauréat STI2D - SIN

- SIN 2.1 : Traitement programmé

Généralités



Le langage C, né en 1972 présente de nombreux avantages :

- **Polyvalent** : il permet le développement d'applications dans tous les domaines d'application de l'OS au programme scientifique
- **Structuré** : il intègre toutes les possibilités des langages structurés
- **Proche de la couche matériel** : il permet de développer des programmes pour des systèmes minimums à microprocesseurs. On dit que c'est un langage de bas niveau.
- **Indépendant de l'OS et de la machine** : Le respect de la norme permet, par une simple recompilation, d'utiliser le code vers d'autres systèmes
- **Rapide** : le code compilé est compact du fait de la proximité avec la couche matérielle
- **Extensible** : de nombreuses bibliothèques existent et des millions d'utilisateurs à travers le monde s'échangent des informations
- A donné naissance à d'autres langages qui ont adapté certaines structures du langage : PHP, C++, java, etc

Un peu d'histoire

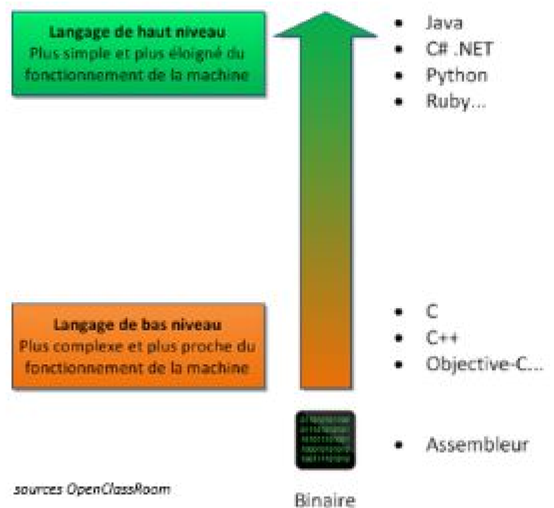
Le langage C a été mis au point en 1972 dans les laboratoires Bell par Dennis RITCHIE.

Il est un dérivé du langage B lui même inspiré du langage BCPL.
RICHIE avait besoin d'un langane de programmation solide pour développer la première version d'UNIX.
Depuis ses débuts, C a évolué et a été normalisé :

- en 1989 avec ANSI C
- en 1994, 1996, 1999 et 2011 : par le groupe de travail de l'ISO

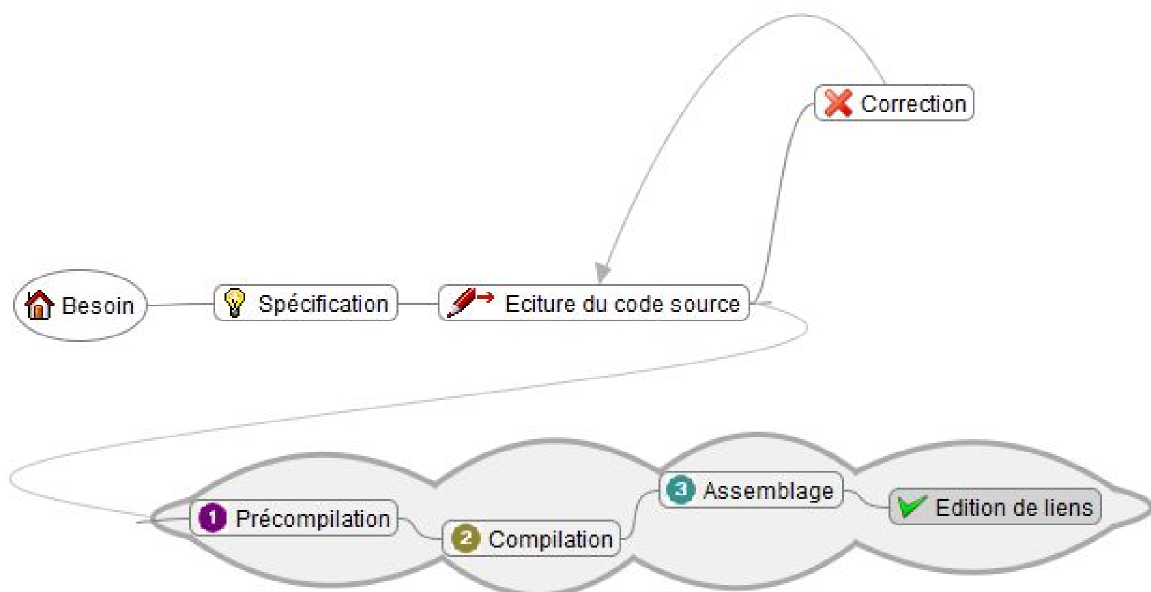
La nécessité de créer rapidement des programmes et adaptés au WEB a contribué à la naissance de langages orientés objets mais qui reprennent les mêmes bases:

- le plus connu étant le C++ : Très proche du C
- le C# : Langage de haut niveau
- Java : Langage de haut niveau

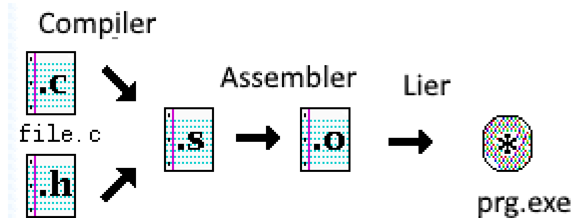


Génération d'un programme

La génération d'un programme passe par différentes étapes.



- **Spécification :** Réalisée par le donneur d'ordre (client ou client + assistant de maîtrise d'ouvrage) du projet de développement : Consiste a écrire de manière aussi précise que possible les fonctionnalités demandées à l'application. Voir ce cours si besoin.
- **Ecriture du code source** : C'est le travail de l'équipe de développement : Il s'agit d'écrire le programme en langage C
- **Correction :** C'est le travail de l'équipe de développement ou de l'équipe de recette du produit :Utilisation des outils de déverminage (débogage) et tests unitaires ou d'intégrations.
- **Précompilation :** réalisé par l'environnement de développement. Consiste à exécuter les directives des fichiers sources (#include, #if, #define, etc...)



- **Compilation :** Cette phase réalisée par l'environnement de développement consiste à générer le code assembleur à partir du code source. Les erreurs de syntaxes sont décelées ici.
- **Assemblage** : Réalisée par l'environnement de développement. Cette phase génère des fichiers objets utilisés par le lieur. Les fichier objets possèdent l'extension .obj pour windows et .o pour raspberry (Unix).
- **Edition de liens :** Réalisée par l'environnement de développement, elle a pour but de générer un fichier exécutable à partir des différents fichiers objets et des bibliothèques.

Mode Debug et Mode Release

Lorsque l'on développe une application, la phase de codage nécessite que l'informaticien puisse visualiser certaines données (état des variables, état de la mémoire, etc...). Ces données sont intégrées dans les fichiers objets lors de la phase de deverminage mais alourdissent leur taille.

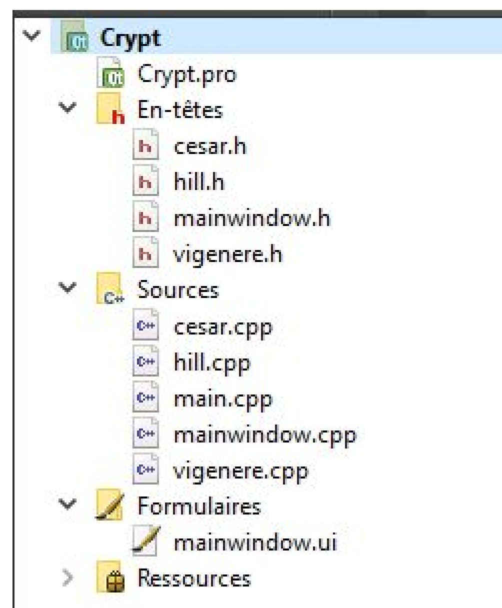
Pour la génération du fichier exécutable final, ces informations ne sont pas utiles.

Il est donc important de préciser à l'environnement de développement quel est l'objectif de la compilation.

Les fichiers d'un programme

Un programme C comporte généralement plusieurs fichiers :

- **Les fichiers sources (.c en C ou .cpp en C++)** : Ils contiennent le code source (en langage C) du programme. Par défaut, le nom du fichier source principal sera toujours ***main.c*** ou ***main.cpp***. Le contenu de ce fichier peut être très sommaire mais la fonction ***main*** doit y être.



main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

La compilation d'un programme source se fait de manière séquentielle ce qui veut dire qu'une fonction doit être connue avant d'être utilisée. Il faut donc écrire ce qu'on appelle les ***prototypes*** et qui correspondent à la déclaration des fonctions.

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QWidget>
#include <QMessageBox>

namespace Ui {
class MainWindow;
}

class MainWindow : public QWidget
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_chiffrer_clicked();

    void on_choix_chiffrage_currentIndexChanged(int index);

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

- **Les fichiers en-têtes (.h) :** ils contiennent les prototypes. En général, pour chaque fichier source (.c ou .cpp) il y a un fichier d'en-tête. Les fichiers d'en-tête doivent être déclarés dans le fichier source par la directive *#include*. Ils peuvent eux-même appeler d'autres en-têtes ou d'autres bibliothèques.

On remarque dans ce fichier d'en-tête la déclaration de la classe d'objet MainWindow

- **Le fichier de projet :** Il est unique dans un même projet car c'est lui qui décrit comment les fichiers doivent être utilisés et comment le compilateur doit s'exécuter. Avec QTCreator l'extension est .pro
- **Les fichiers de formulaires :** ils correspondent à la description des interfaces graphiques (les fenêtres du programme)
- **Les ressources :** il s'agit de tous les fichiers nécessaires pour l'application : images, icônes, vidéo, sons, etc....

Le langage

Les règles syntaxiques de base

1. Un bloc d'instruction est délimité par les caractères **{** et **}** qui sont les équivalents algorithmiques de ***DEBUT*** et ***FIN***

```
//*************** Affiche me message de déconnexion lorsque le serveur a déco
void MainWindow::deconnecte()
{
    ui->messageTE ->append(tr("<em>Déconnecté du serveur</em>"));
    ui->actionConnexion->setEnabled(true);
    ui->adresseEdit->setEnabled(true);
    ui->portSB->setEnabled(true);
}
//*********************************************************************
```

2. Il ne faut pas mettre deux instructions sur la même ligne.
3. Chaque ligne d'instruction se termine par le caractère **;**
4. Une ligne de commentaire (commentaire court) est précédée des caractères **//**
   Remarque1 : il est indispensable de commenter les programmes
   Remarque2 : une bonne pratique consiste à séparer les fonctions par une ligne de commentaire (des astérisques par exemple)
5. Un bloc de commentaire (commentaire long) commence par les caractères **/*** et termine par ***/**
6. Le nom des variables, des procédures ou des fonctions sont alphanumériques mais commencent obligatoirement par une lettre. Les caractères **_** ou **-** sont tolérés. Les lettres accentuées et l'espace sont interdits.
   Remarque : Une bonne pratique consiste à donner des noms explicites mais courts. Si la variable est formée par la concaténation de plusieurs mots, il est préférable de mettre le premier caractère de chaque mot en majuscule. Exemple : gainJoueur1
7. C distingue la casse des caractères : ***MaVariable*** est différent de ***maVariable***.

Les types standards

Un type caractérise les valeurs que peut prendre une donnée.
C utilise le typage statique ce qui veut dire que la vérification de type est réalisée à la compilation.
L'intérêt du type est de permettre à l'ordinateur de réserver des zones mémoire allouées aux variables du programme.

Un bon choix de type permet d'optimiser un programme mais aussi d'éviter des dysfonctionnements parfois tragiques :
Le vol d'Ariane 5 a échoué parce que le type de la variable définissant l'accélération était mauvais, ce qui a provoqué un erreur de dépassement d'entier. (Sources Wikipedia).

Les types standards en C et C++

Type	Explication	Taille (en octets)	Plage de valeurs
bool	booléen (existe en C++ mais pas en C)		true ou false
char	caractère	1	-128 à +127
unsigned char	caractère non signé	1	0 à 255
int	Entier	2 (processeur 16bits)	-32768 à 32767
unsigned int	Entier non signé	2 (processeur 16 bits)	0 à 65535
long int	Entier long	4	-2147483648 à 2147483647
unsigned long int	Entier long non signé	4	0 à 4294967295
float	flottant (à virgule)	4	$3.4*10^{-38}$ à $3.4*10^{38}$
double	double flottant	8	$1.7*10^{-308}$ à $1.7*10^{308}$
long double	flottant double long	10	$3.4*10^{-4932}$ à $3.4*10^{4932}$

Remarque 1 : C ne gère pas de manière native les chaînes de caractère, mais en déclarant la bibliothèque string.
Remarque 2: Il est possible de créer d'autres types en utilisant les structures de données
Remarque 3 : Un type char permet de stocker la valeur ASCII d'un caractère

char caractere;
caractere=66;
est identique à
caractere='B';

```
#include <iostream>
#include <string>
using namespace std;
int main() {
  string nom;
  cout << "Donnez votre nom: ";
  cin >> nom;
  cout << "Bonjour " << nom << endl;
}
```

Remarque 4 : un tableau de caractère est identique à une chaine de caractère :

char maChaine[];
maChaine="Bonjour";

Utilisation des types / Déclaration d'une variable

Une déclaration de variable dans un programme correspond à une réservation d'octets dans la mémoires.

Pour déclarer une variable, on commence par déclarer le type de variable.

```
//-----------------------------------------------------------------------------
int MainWindow::enhancedanalize(int x,int y)
//recherche les lignes avec 2 pions placés
//rend 2 si l'ordi a deja 2 pions, rend 4 si l'humain à deux pions
//on favorise la défense
{
int human2=0;
int machine2=0;
int resultat=0;
//on vérifie si l'occupation de la colonne
   for (int i=0;i<3;i++)
   {
       if (game[x][i]==1)
       human2=human2+1;
       if (game[x][i]==2)
       machine2=machine2+1;
    }

       return resultat;
}
//-----------------------------------------------------------------------------
```

Déclaration de variables locales de type INT (nombre entier) initialisées à 0

Déclaration d'une variable dans une boucle d'itération

- Il est toujours possible, voire indispensable dans certains cas, d'initialiser une variable : *type nomVariable = valeur;*
- Lorsqu'une variable (et même une constante) est déclarée dans une procédure ou une fonction, cette variable n'a d'existence que dans la fonction.
- Lorsqu'une variable est déclarée dans l'en-tête du programme, elle aura une existence dans tout le programme
- Les fonctions restituent une donnée. Elles sont donc également typées (voir ci-dessus : *enhancedanalize* restituera un entier)

Les opérateurs

Opérateurs logiques de test

Ces opérateurs sont nécessaires pour combiner des conditions logiques

- **&&** : fonction ET : a && b (a est VRAI ET b est VRAI)
- **||** Fonction OU : a || b
- **!** fonction NON : a!=0

Opérations logiques

- Opérateur **AND** : a **&** b : si a=0x85 et b = 0x0F le résultat donne 0x05
- Opérateur OR : a | b : réalise le OU logique entre a et b
- Opérateur NOT : ~a : complément de a

Comparaisons

- égalité : if (a==b)....
- supériorité : if (a>b)...
- supériorité et égalité : if (a>=b)....
- infériorité if (a<b)
- infériorité ou égalité : if (a<=b)....
- différence if !(a==b)... ou if (a!=b)...

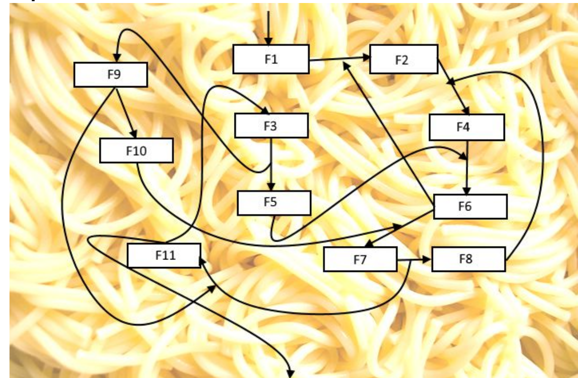
Opérations arithmétiques

- affectation : =
- addition : b + 3;
- soustraction a - 4
- multiplication a * 2
- division : a/3; //si a est un entier, le résultat est un entier. Si a est un flottant, le résultat est un flottant
- modulo (reste de la division) 7 %2;//donne 1

Langage procédural

Un programme informatique deviendrait très vite illisible si l'ensemble du code devait être écrit sans avoir la possibilité de réutiliser (d'appeler) certains blocs. Le langage GWBasic de la fin des années 1970 présentait cet inconvénient.

Un langage procédural permet d'écrire des blocs de codes appelés procédures, routines ou fonctions. Ces blocs possèdent un nom unique et ils peuvent être appelés dans le code par ce nom.



La programmation procédurale présente les avantages

- de fournir un code plus lisible (des noms décrivent un ensemble d'instructions)
- de fournir un code plus compact (moins de ligne)
- d'éviter des sauts vers des lignes qui rendent la lecture illisible ("*programmation spaghetti*")

Le C est un langage procédural qui gère à la fois

- les procédures : *void*
- les fonctions : *function*

Les procédures

Une procédure est un bloc d'instruction qui réalise des opérations sans restitution de résultat.

<u>Par exemple :</u>

- affichage d'un message à l'écran
- sauvegarde d'un fichier
- lancement d'une séquence vidéo
- réponse à un événement (clic souris, touche de clavier...)

```
//---------------------------------------------------------------------------
void dialogue() //demande à l'utilisateur quel URL il faut encoder
{
    bool ok = false;
    url = QInputDialog::getText(NULL, "QR Code", "Quel est l'URL que vous sou
}
//---------------------------------------------------------------------------
void save() //sauvegarde le QR Code au format BMP
{
        QPixmap qrimg();
        qrimg = qrimg.scaledToWidth(250);
        qrimg = qrimg.scaledToHeight(250);
        qrimg.save("qr_code.bmp");
}
//---------------------------------------------------------------------------
```

Quelques règles :

- le mot void précède le nom de la procédure
- les noms de procédures respectent le même formalisme que les [noms de variable](#).
- La procédure doit bien sûr être déclarée dans l'entête.
- Une variable déclarée dans une procédure (entre le caractère { de début de

```
//------------------------------------------------------------------------
void MainWindow::boutonclique(int numero)
{
    bool next=false;
    int choixx=0;
    int choixy=0;
    switch (numero)
    {
         case 1: {
            if (game[0][0]==0)
             {
              bouton1->setIconSize(QSize(95,95));
              bouton1->setIcon(QIcon(":/icones/joueur.png"));
              game[0][0]=1;
              next=true;
              choixx=0;choixy=0;
             }
              break;
        }
        case 2: {
        if (game[1][0]==0)
         {
        bouton2->setIconSize(QSize(95,95));
        bouton2->setIcon(QIcon(":/icones/joueur.png"));
        game[1][0]=1;
        choixx=1;choixy=0;
        next=true;

         }
        break;
   }

 }
}
```

  procédure et le caractère } de fin) n'a d'existence que dans la procédure. Il s'agit d'une variable locale.
- il est possible de spécifier des arguments lors de la déclaration d'une procédure. Ils sont des variables utilisables dans le code de la procédure.
- En C++, si la procédure est liée à un objet, le nom de la procédure est précédée par les caractère **::** (**->** en C++ Builder), puis par le nom de la classe d'objet.

Les fonctions

Une fonction est un procédure qui restitue un résultat.
Les règles sont donc sensiblement les mêmes que pour les procédures.
Tous les programmes ont au moins une fonction qui est ***main()***. Elle est située dans le fichier ***main.c*** ou ***main.cpp***.

Cette fonction est indispensable car elle démarre l'exécution du programme.

main.cpp

```
//----------- Tic Tac Toe -------------------------
//------------------------------------------------
#include <QApplication>
#include <mainwindow.h>
#include <QtWidgets>
#include <vector>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    MainWindow fenetreprincipale;
    fenetreprincipale.show(); //on affiche la fenêtre
    return app.exec();
}
```

Pour déclarer une fonction la syntaxe est : ***type nom(arguments)***

- **type :** spécifie le type de la valeur restituée
- **nom** : il faut choisir un nom qui respecte les spécifications déjà vues (variables, procédures) et qui décrit bien ce que fait la fonction.
- **arguments :** comme pour *void*, ce sont les données avec lesquelles la fonction va travailler.

Se rajoutent à la déclaration :

- **les accolades ({ }):** elles encadrent le code de la fonction
- **l'instruction *return*** qui affecte le résultat des opérations de la fonction au nom de la fonction.

main.cpp*

```
#include <iostream>
#include <string>
using namespace std; //précise que l'on utilise des bibliothèques standards

int surface(int cote)
{
    return cote*cote;
}

int main() {
    int c;
   cout<<"Donner la valeur du cote de ce carre :";
   cin>>c ;
   cout<<"La surface est : "<<surface(c)<<endl;
}
```

Les structures de contrôle

Tout programme doit en fonctions des résultats ou donnés précédents prendre une décision sur des opérations à mener. Les structures de contrôle permettent cela.

Elles sont classées en deux catégories :

- Les structures conditionnelles
- Les structures itératives

Les structures conditionnelles

IF THEN ELSE

```
void tester()
{
    if (casecochee==true)
    {
        cout<<"La case est cochée"<<endl;
    }
    else
    {
        cout<<"La case n'est pas cochée"<<endl;
    }
    cout<<"Fin !"<<endl;
}
```

SI la condition est vrai alors une action est exécutée SINON une autre action est réalisée.

Remarque 1 : la condition ELSE n'est pas obligatoire

Remarque 2 : il est possible d'imbriquer plusieurs conditions IF.. THEN .. ELSE

SWITCH

```
void tester()
{
    switch (bouton)
    {
      case 0 : cout<<"Bouton 1"<<endl;break;
      case 1 : cout<<"Bouton 2"<<endl;break;
      case 2 : cout<<"Bouton 3"<<endl;break;
      case 3 : cout<<"Bouton 4"<<endl;break;
      default :    cout<<"Aucun bouton"<<endl;break;
    }
}
```

Cette structure permet d'éviter la succession de IF..THEN..ELSE dasn une structure de contrôle.
La variable testée n'est pas nécessairement binaire. Il peut s'agit d'un nombre, d'un caractère.
Remarque : l'instruction break termine chaque possibilité et il n'est pas nécessaire de mettre des accolades lorsqu'une seule instruction et le break sont utilisés.
Remarque : *Default* permet de définir l'action par défaut si aucune autre condition ne convient.

Les structures itératives

Appelées aussi boucles, elles permettent de répéter des blocs d'instructions.

Boucle FOR

```
void tester(int N)
{
    for (int i=0;i<N;i++)
        cout<<"i="<<N<<endl;
    cout<<"Fin!";
}
```

La syntaxe de cette structure est un peu particulière. *i* est déclaré comme une variable entière locale. L'itération s'arrête lorsque la condition *i<N* n'est plus satisfaite. L'incrémentation de *i* est spécifiée par *i++*.
La boucle FOR ne peut être utilisée que si l'on connait au départ le nombre d'itérations à réaliser.

Boucle WHILE

```
void tester(int N)
{
    int M=1;
    while (M<N)
    {
    cout<<"N="<<N<<" et M="<<M<<endl;
    M+=2;
    }
    cout<<"Fin!"<<endl;
}
```

Cette boucle est utilisée quand on ne connaît pas au départ le nombre d'itérations à réaliser.

Il faut l'utiliser avec beaucoup de précaution car elle peut provoquer des blocages de programme si la condition d'arrêt n'est jamais satisfaite.

DO... WHILE

La boucle *DO WHILE* est une boucle *WHILE* dans laquelle le test d'arrêt est fait à la fin du bloc d'instructions.

```
void tester(int N)
{
    int M=1;
    do
    {
    M+=2;
    cout<<"N="<<N<<" et M="<<M<<endl;
    }while (M<N);
    cout<<"Fin!"<<endl;
}
```

Comme pour la boucle *WHILE*, il est nécessaire que la condition d'arrêt soit satisfaite.

Introduction sur les pointeurs

Un pointeur est une variable contenant *l'adresse* (la position dans la mémoire centrale de l'ordinateur) d'une autre variable d'un type donné. Les pointeurs permettent de définir des structures dynamiques, c'est-à-dire qui évolue au cours du temps comme des tableaux dynamiques par exemple.

Principe

En préambule, il est bon de rappeler que chaque variable correspond à des cases (octets) dans la mémoire centrale de l'ordinateur.

```
00882A70  FB FC FD FE DF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA  ûüýþßÀÁÂÃÄÅÆÇÈÉÊ
00882A80  CB CC CD CE CF D0 D1 D2 D3 D4 D5 D6 00 D8 D9 DA  ËÌÍÎÏÐÑÒÓÔÕÖ.ØÙÚ
00882A90  DB DC DD DE 9F 00 00 00 08 00 45 00 24 01 0F 00  ÛÜÝÞ▌.....E.$...
00882AA0  A8 2A 88 00 00 00 00 00 44 3A 5C 50 75 72 65 42  ¨*▌.....D:\PureB
00882AB0  61 73 69 63 34 5F 30 5C 43 6F 6D 70 69 6C 65 72  asic4_0\Compiler
00882AC0  73 5C 50 75 72 65 42 61 73 69 63 30 2E 65 78 65  s\PureBasic0.exe
00882AD0  00 00 00 00 00 00 00 00 12 00 08 00 2C 01 08 00  ............,...
00882AE0  70 2B 88 00 B0 2B 88 00 F8 2B 88 00 18 2C 88 00  p+▌.°+▌.ø+▌..,▌.
00882AF0  58 2C 88 00 78 2C 88 00 A8 2C 88 00 C8 2C 88 00  X,▌.x,▌.¨,▌.È,▌.
00882B00  E0 2C 88 00 10 2D 88 00 30 2D 88 00 78 2D 88 00  à,▌..-▌.0-▌.x-▌.
00882B10  98 2D 88 00 B0 2D 88 00 B0 2E 88 00 F8 2E 88 00  ▌-▌.°-▌.°.▌.ø.▌.
```

Selon le type de la variable, le nombre de cases est plus ou moins important. Une déclaration de variable dans un programme correspond à une réservation d'octets dans la mémoires.

Chaque case mémoire est caractérisée par son adresse.
Il est donc possible, dans un programme, d'accéder au contenu d'une variable

- par le nom de la variable :

  unsigned char maVar; //réserve l'emplacement d'un octet en mémoire pour *maVar*
  maVar =4;//on accède à la donnée *maVar* par son nom et on peut la modifier

```
00882A70  FB FC FD FE DF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA
00882A80  CB CC CD CE CF D0 D1 D2 D3 D4 D5 D6 00 D8 D9 DA
00882A90  DB DC  adresse de maVar  08 00 45 00 24 01 0F 00
00882AA0  A8 2A  maVar       00 00 44 3A 5C 50 75 72 65 42
00882AB0  61 73 69 63 34 5F 30 5C 43 6F 6D 70 69 6C 65 72
00882AC0  73 5C 50 75 72 65 42 61 73 69 63 30 2E 65 78 65
00882AD0  04 00 00 00 00 00 00 00 12 00 08 00 2C 01 08 00
00882AE0  70 2B 88 00 B0 2B 88 00 F8 2B 88 00 18 2C 88 00
00882AF0  58 2C 88 00 78 2C 88 00 A8 2C 88 00 C8 2C 88 00
00882B00  E0 2C 88 00 10 2D 88 00 30 2D 88 00 78 2D 88 00
```

- par l'adresse de la variable.
  Le plus souvent il n'est pas nécessaire de connaître l'adresse de la variable. Elle est transparente pour le programmeur mais on peut l'obtenir en utilisant le caractère *espaluette* : &

Exemple :

dans l'exemple précédent l'instruction :

```
cout<<&maVar;
```

donnerait 00882AD0

Qu'est-ce qu'un pointeur

Un pointeur est donc une variable qui contient l'adresse d'une variable.
Un pointeur est toujours typé et doit être déclaré comme toute autre variable.

Déclaration d'un pointeur

forme: type * nom_du_pointeur(valeur initiale);
exemple :int * mon_pointeur(0);

En reprenant l'exemple précédent :

unsigned char *pVar(0); //création d'un pointeur dont le type est identique à celui maVar.On l'initialise à 0
unsigned char temp;
pVar = &maVar; //pVar prend la valeur de l'adresse de maVar
temp = *pVar; //après cette opération temp sera égal à maVar

```
00882A70   FB FC FD FE DF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA
00882A80   CB CC CD CE CF D0 D1 D2 D3 D4 D5 D6 00 D8 D9 DA
00882A90   DB DC  adresse de maVar     08 00 45 00 24 01 0F 00
00882AA0   A8 2A  maVar             00 00 44 3A 5C 50 75 72 65 42
00882AB0   61 73 69 63 34 5F 30 5C 43 6F 6D 70 69 6C 65 72
00882AC0   73 5C 50 75 72 65 42 61 73 69 63 30 2E 65 78 65
00882AD0   04 00 00 00 00 00 00 00 12 00 08 00 2C 01 08 00
00882AE0   70 2B 88 00 B0 2B 88 00 F8 2B 88 00 18 2C 88 00
00882AF0     pVar          2C 88 00 A8 2C 88 00 C8 2C 88 00
00882B00   E0 2C 88 00 10 2D 88 00 30 2D 88 00 78 2D 88 00
00882B10   00 88 2A D0 B0 2D 88 00 B0 2E 88 00 F8 2E 88 00
```

Manipulation d'un pointeur

Création

Si le pointeur doit être créé de manière manuelle, ce qui signifie qu'il n'a pas été déclaré dans l'en-tête du programme (comme dans les exemples ci-dessus), il est nécessaire de réserver de la place dans la mémoire avec l'instruction ***NEW***.

pVar = new (unsigned char); //on demande au système de trouver de la place dans la mémoire

Destruction

Pour libérer la mémoire, il faudra ensuite utiliser l'instruction ***DELETE***

delete pVar; // on libère la mémoire

Remarque : lors de la fermeture normale d'un programme, les réservation sont automatiquement libérées par le système d'exploitation.

Intérêt des pointeurs

- Lorsque des objets ou des variables doivent être créés de manière dynamique dans un programme alors qu'on ne sait pas quelle taille ils vont occuper dans la mémoire, il est indispensable de passer par les pointeurs (ou des structures dérivées comme des vecteurs)
  - Exemple de création de 100 boutons dans une fenêtre (this) :

```
for(int i = 0; i < 100; i++)
  {
  mesBoutons[i] = new QPushButton("", this);
  mesBoutons[i]->setMinimumWidth(40);
  mesBoutons[i]->setMinimumHeight(40);
  }
```

- Les pointeurs sont utilisés aussi pour partager des données (en particulier des tableaux) entre les fonctions ou dans plusieurs morceaux de code.

Généralités sur les tableaux

Rares sont les applications informatiques qui ne nécessitent pas un tableau.
Un tableau est un structure de données complexes regroupant plusieurs (voire de nombreuses) données de même type. Par exemple, le nom des élèves d'une classe pourraient être enregistrés dans un tableau.

Dimension d'un tableau

Un tableau peut être à une dimension ou à plusieurs. **Mais la dimension est fixe**.
Si la taille du tableau est connue et fixe, on utilisera un tableau statique dans le cas contraire on fera appel aux tableaux dynamiques.

Accès aux valeurs

On accède à une valeur stockée dans un tableau en combinant le nom du tableau et l'indice (numéro de case) de la valeur dans le tableau.
Exemple d'un tableau de 1 dimension de taille égale à 10 :

Nom	2	8	10	50	1	25	18	0	8
Indice	0	1	2	3	4	5	6	7	8

La valeur de la case d'*indice* 5 du tableau *Nom* est 25

Exemple de tableau :

- Un chaîne de caractère est un tableau de caractères sur une dimension
- Un plateau de jeu (un damier) est un tableau sur deux dimensions
- Un fichier informatique est un tableau de valeur.

Les tableaux statiques

Déclaration d'un tableau statique à une dimension :

Un tableau statique possède une taille fixe initialisée et qui ne peut pas changer.

```
int montableau[10]: //déclaration d'un tableau d'entiers de 10 cases
```

Remarque : la première case porte toujours l'indice 0

Affectation de valeurs :

```
for (int i=0;i<10;i++)
{ //affecte un nombre croissant de 1 à 10 dans les 10 cellules du tableau
 montableau[i]=i;
}
```

Lecture de valeurs :

```
if (montableau[1]!=0)
{
cout<<"Valeur différente de 0";
}
```

Tableau statique à deux dimensions

Le principe est le même que précédemment, mais il faut rajouter la dimension supplémentaire :

```
int montableaudouble[10][5]: //déclaration d'un tableau d'entiers de 10 x5 cases
```

L'affectation et la lecture se fait de la même manière que précédemment..

0, 0	0, 1	0, 2	...
1, 0	1, 1	1, 2	...
2, 0	2, 1	2, 2	...
...	...	...	...

Affectation de valeurs à un tableau à deux dimensions:

montableaudouble[1][1]=0;

Lecture de valeurs :

```
if (montableaudouble[2][1]==8)
{
//code
}
```

Tableau de constante

Il est fréquent qu'il faille initialiser un tableau avec des constantes.
La méthode passe par la définition du tableau et par la déclinaison des valeurs.

Exemple d'un tableau à une dimension :

```
char chaine[] = {'B', 'o', 'n', 'j', 'o', 'u', 'r'};
```

Exemple d'un tableau à deux dimensions :

```
int tabl[3][3] =
{
{4, 3, 6},
{10, 0, 0},
{-1, 5, 3}
};
```

Autre exemple : Tableau de 25x25

```
//-------------------------------------------------------
bool tableau_qr[25][25]=
{
    {1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1},
    {1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1},
    {1,0,1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,1,1,1,0,1},
    {1,0,1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,1,1,1,0,1},
    {1,0,1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,1,1,1,0,1},
    {1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1},
    {1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,1,1,1,1,1,1},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0},
    {1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,0},
    {1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0},
    {1,0,1,1,1,0,1,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0},
    {1,0,1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {1,0,1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
};
//-------------------------------------------------------
```

Tableau dynamique

Comme il est nécessaire de déclarer une variable avant de l'utiliser et comme la taille d'un tableau n'est pas forcément connue, on peut contourner cette contrainte en déclarant un pointeur vers un tableau dont on définira la taille ultérieurement.

Cette déclaration est possible avec la commande NEW :

nom_du_pointeur= new type[taille]

Exemple

Dans l'initialisation (header par exemple) :

```
int *tableau;
int taille;
```

Dans le code :

```
cin >> taille;
tableau = new int[taille];
```

Autre exemple : Placement des données d'un fichier (dont on ne connait pas la taille initialement) dans un tableau

Dans l'entête :

```
unsigned char *filecontent;
```

Dans le code :

```
QFile orgBMPfile("image.bmp"); //on lie un type file à un fichier image
taillefichier=10000; //la taille totale du fichier fait 10000 octets
filecontent = new unsigned char[taillefichier]; //on créé le tableau
orgBMPfile.read((char *)filecontent, taillefichier);// on copie toutes les données dans le tableau
```

Remarque :(char *) permet ici d'adapter les types à la fonction read

Remarque : Une autre méthode consiste à utiliser les vecteurs

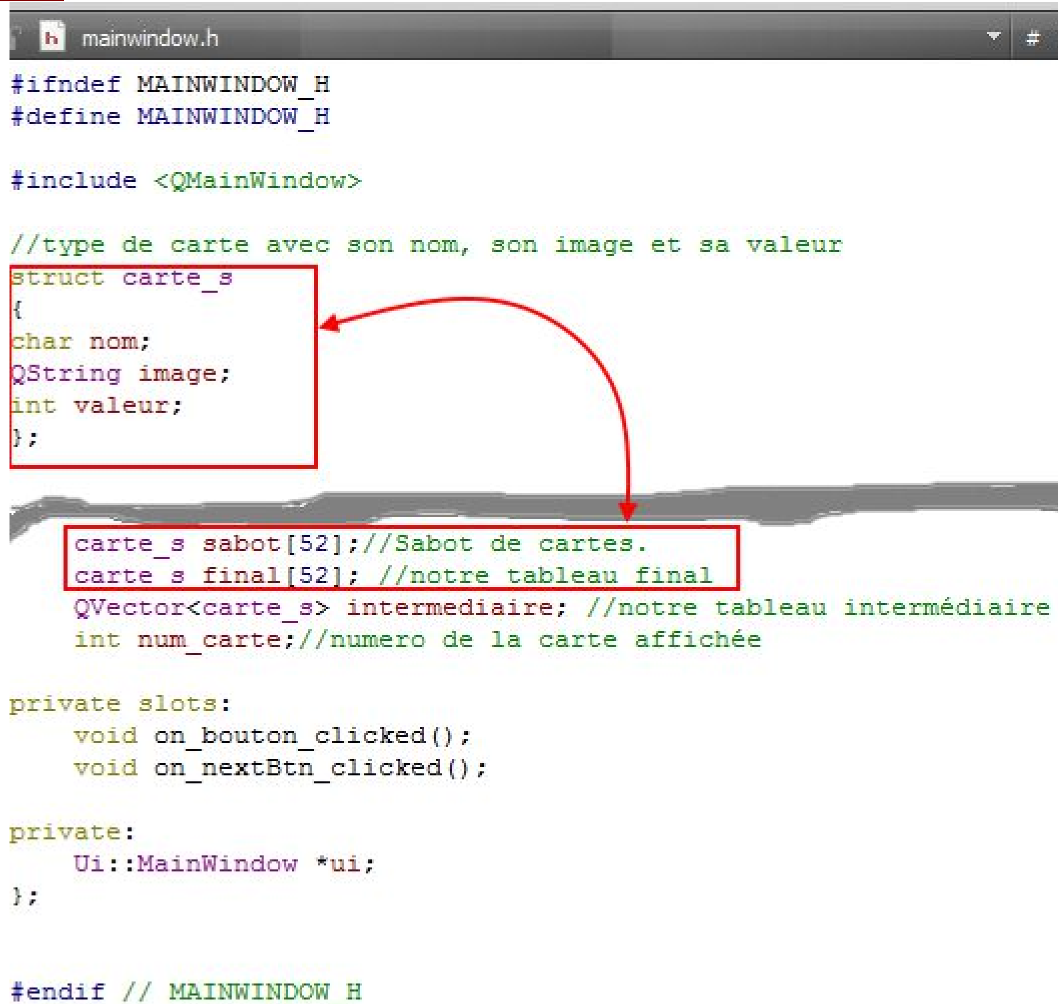
Les structures de données

Les types standards définie dans C normalisé ne permettent pas toujours de satisfaire à tous les besoins.
Les structures de données permettent de définir de nouveaux types plus complexes.
Une structure de donnée permet de regrouper des données de type différent dans un même type de variable.
Par exemple, un utilisateur de bibliothèque possède un nom et un prénom de type chaîne et un numéro de type entier.
Il est possible de créer un type particulier puis des variables ou des constantes définies par ce type.
Le mot clé en C++ est ***struct***.

Déclaration :

```
struct Tuser
        {
           QString nom;
           QString prenom;
           int ID;
        } ;
Tuser monUtilisateur;
```

Exemple



```
mainwindow.h

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

//type de carte avec son nom, son image et sa valeur
struct carte_s
{
char nom;
QString image;
int valeur;
};

    carte_s sabot[52];//Sabot de cartes.
    carte_s final[52]; //notre tableau final
    QVector<carte_s> intermediaire; //notre tableau intermédiaire
    int num_carte;//numero de la carte affichée

private slots:
    void on_bouton_clicked();
    void on_nextBtn_clicked();

private:
    Ui::MainWindow *ui;
};


#endif // MAINWINDOW_H
```

On peut utiliser la structure pour créer un tableau de structure. Dans l'exemple ci-dessus sabot est un tableau de cartes à jouer définies par la structure de données carte_s.