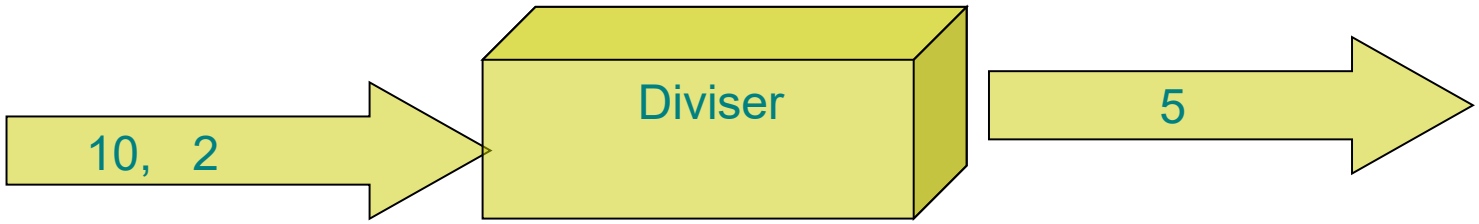
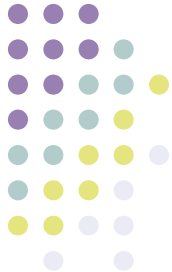
Créer et appeler une fonction

- Une fonction exécute des actions et renvoie un résultat.
- C'est un **morceau de code** qui sert à faire quelque chose de précis
- Une fonction possède des paramètres d'entrée et une sortie
- **L'entrée**: on fait "rentrer" des informations dans la fonction (en lui donnant des informations avec lesquelles travailler)
- **Les calculs** : grâce aux informations qu'elle a reçues en entrée, la fonction travaille et produit des résultats.
- **La sortie** : une fois qu'elle a fini ses calculs, la fonction renvoie des résultats. C'est ce qu'on appelle la (ou les) sortie(s), ou encore le (ou les) retour(s).



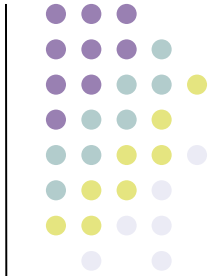
Créer et appeler une fonction



- Un programme est un ensemble de fonctions qui s'appellent les unes les autres
- Chaque fonction appelée peut appeler une ou plusieurs fonction (séquentiellement)
- Le but des fonctions est de simplifier le code source et le rendre lisible :
  - une tâche peut être réalisée en sous-tâches

```
type nomFonction(parametres)
{
    // Insérez vos instructions ici
}
```

Exemples simples



```
1 int somme(int a, int b)
2 {
3     return (a + b);
4 }
1 void affiche_somme (int a, int b)
2 {
3     cout << "la somme est : " << a + b << endl;
4 }
1 void disBonjour()
2 // cette fonction ne retourne rien ➔ void
3 {
4     cout << "Bonjour\n";
5 }
```

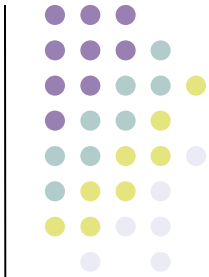
Créer et appeler des fonctions

```
 1 # include <iostream>
 2
 3 using namespace std;
 4
 5 int triple(int nombre)
 6 {
 7     return (3 * nombre);
 8 }
 9
10 int main()
11 {
12  int nombreEntre = 0, nombreTriple = 0;
13  cout << "Entrez un nombre... ";
14  cin >> nombreEntre;
15  Triple = triple(nombreEntre);
16  cout << "Le triple est :" << Triple << endl;
17  return 0;
18 }
```

Créer et appeler des fonctions

```
 1 # include <iostream>
 2 # include <math.h>
 3
 4 using namespace std;
 5
 6 float racine_carre(float nombre)
 7 {
 8     return sqrt(nombre);
 9 }
10
11 int main()
12 {
13     float nombreEntre = 0, nombre_RC = 0;
14     cout << "Entrez un nombre positif... ";
15     cin >> nombreEntre;
16     if (nombre_RC >= 0.0)
17     {
18         nombre_RC = racine_carre(nombreEntre);
19         cout << "La racine carre est " << nombre_RC << endl;
20     }
21     else
22     {
23         cout << "Le nombre doit etre positif\n";
24     }
25     return 0;
26 }
27
```

Un autre exemple intéressant



```
 1 # include <iostream>
 2 using namespace std;
 3
 4 int menu() {
 5     int choix = 0;
 6     while (choix < 1 || choix > 4) {
 7         cout << "Menu :\n";
 8         cout << "1 : Poulet de dinde aux escargots\n";
 9         cout << "2 : Concombres sucres a la sauce de myrtilles\n";
10         cout << "3 : Escalope de kangourou saignante \n";
11         cout << "4 : La surprise du Chef\n";
12         cout << "Votre choix ? ";
13         cin >> choix;
14     }
15     return choix;
16 }
18 int main()
{
19     switch (menu()) {
20         case 1:
21             cout << "Vous avez pris le poulet\n";
22             break;
23         case 2:
24             cout << "Vous avez pris les concombres\n";
25             break;
26         case 3:
27             cout << "Vous avez pris l'escalope\n";
28             break;
29         case 4:
30             cout << "Vous avez pris la surprise du Chef !\n";
31             break;
32     }
33     return 0;
34 }
```

Les prototypes

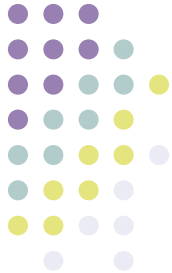
- Jusqu'ici, nous avons placé vos fonctions avant la fonction main. Pourquoi ?
- Parce que l'ordre a une réelle importance
- C’est une contrainte trop forte, surtout pour les programmes longs et complexes
- Nous allons "annoncer" nos fonctions à l'ordinateur en écrivant ce qu'on appelle des **prototypes**.

```
1 int somme(int, int);
```

```
1 int somme(int x, int y);
```

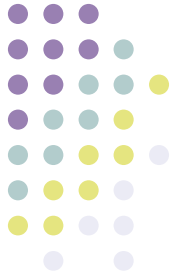
- Ca permet à l'ordinateur de s'organiser.

Les prototypes



```
1  #include <iostream>
2
3  using namespace std;
4
5  // La ligne suivante est le prototype de la fonction aireRectangle :
6  double aireRectangle(double largeur, double hauteur);
7  int main()
8  {
9      cout << "Rectangle de largeur 5 et hauteur 10. Aire "
10          << aireRectangle(5, 10);
11     cout << "Rectangle de largeur 2.5 et hauteur 3.5. Aire "
12          << aireRectangle(2.5, 3.5);
13     cout << "Rectangle de largeur 4.2 et hauteur 9.7. Aire "
14          << aireRectangle(4.2, 9.7);
15     return 0;
16 }
17
18 /* Notre fonction aireRectangle peut maintenant être mise n'importe
19    où dans le code source : */
20
21 double aireRectangle(double largeur, double hauteur)
22 {
23     return largeur * hauteur;
24 }
```

Les headers



- Jusqu'ici, nous n'avions qu'un seul fichier source dans notre projet.
- Dans la pratique, vos programmes ne seront pas tous écrits dans ce même fichier.
  - Imaginer un seul fichier avec 10000 lignes
  - Difficile à lire, difficile à modifier, ce n’est pas pro
- Dans la pratique, un programme est composé:
  - **Les .h** : appelés fichiers headers. Ces fichiers contiennent les prototypes des fonctions.
  - **Les .cpp (.c)** : les fichiers sources. Ces fichiers contiennent les fonctions elles-mêmes.

Les headers

- Imaginer un programme qui utilise:
  - jeu.h : les prototypes
  - Jeu.cpp : les définitions de fonctions
- Comment utiliser les fonctionalités de jeu dans votre programme

```
1 # include <stdlib.h>
2 # include <iostream>
3 # include "jeu.h"

5 void jouer(SDL_Surface* ecran) { // ...
```

- " … " / <…> : Le fichier jeu.h doit se trouver dans le répertoire courant

Les headers

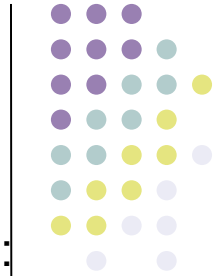
- Imaginer un programme qui utilise:
  - jeu.h : les prototypes
  - Jeu.cpp : les définitions de fonctions
- Comment utiliser les fonctionalités de jeu dans votre programme

```
1 # include <stdlib.h>
2 # include <iostream>
3 # include "jeu.h"

5 void jouer(SDL_Surface* ecran) { // ...
```

- " … " / <…> : Le fichier jeu.h doit se trouver dans le répertoire courant

La compilation séparée



+ Un programme est composé de plusieurs fichiers, souvent en couples :
  - toto.cpp , toto.h
  - tata.cpp , tata.h
  - titi.cpp , titi.h
  - main.cpp

    #include "toto.h"

    #include "tata.h"

    ……

g++ toto.cpp –c ➔ toto.o

g++ tata.cpp –c ➔ tata.o

g++ titi.cpp -c ➔titi.o

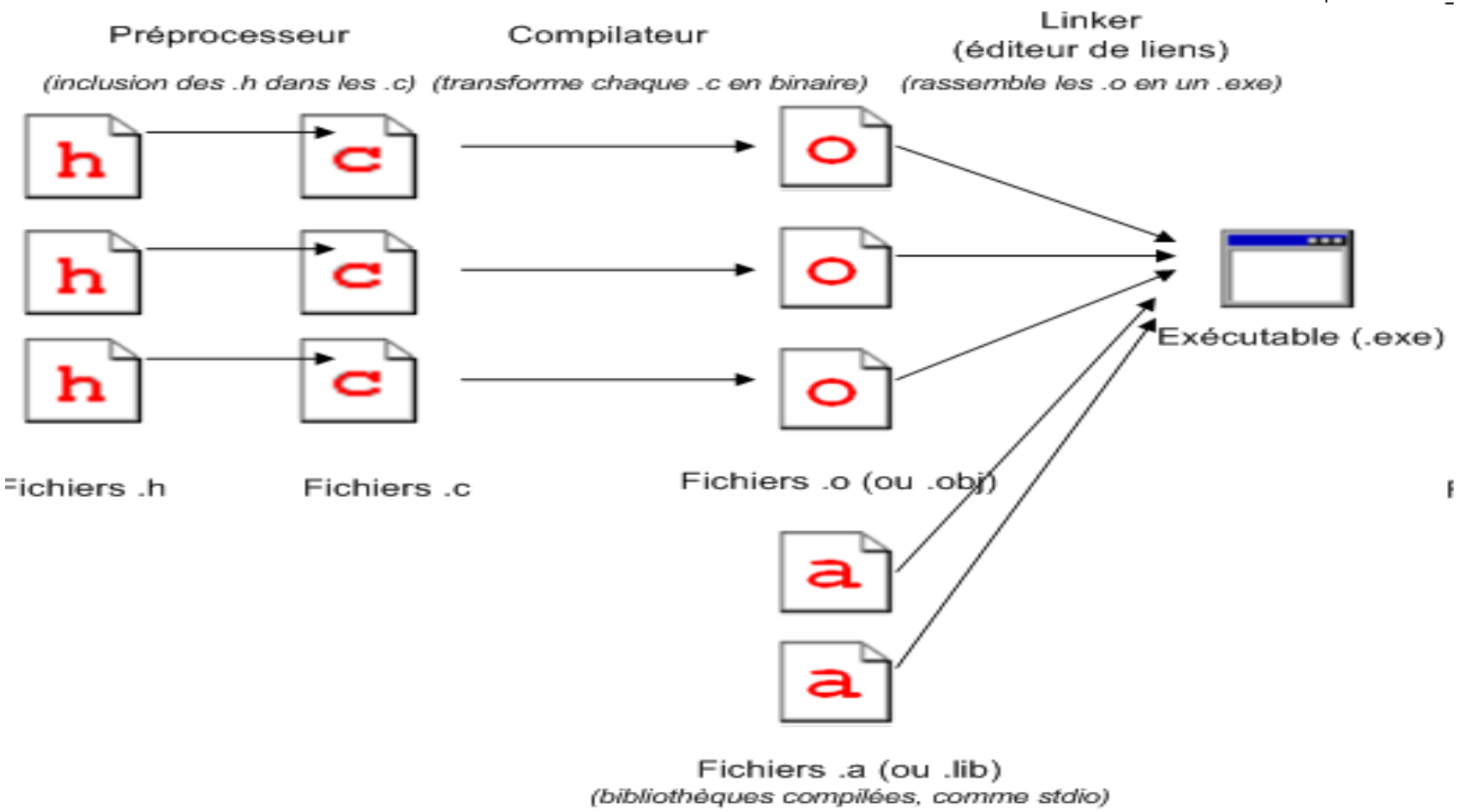
g++ main.cpp -c ➔main.o

g++ toto.o tata.o titi.o main.o –o prog.exe

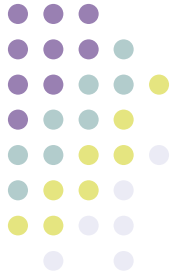
+ On peut aussi simplement faire :

  g++ toto.cpp tata.cpp titi.cpp main.cpp -o prog.exe

Lorsque vous utilisez des bibliothèques



La portée des fonctions et variables

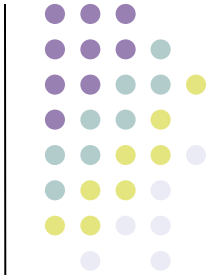


- Lorsque vous déclarez une variable dans une fonction, celle-ci est supprimée de la mémoire à la fin de la fonction :

```
1 int triple(int nombre)
2 {
3    int resultat = 0; // La variable resultat est créée
4       resultat = 3 * nombre;
5       return resultat;
6 } // la variable resultat est supprimée de la mémoire
```

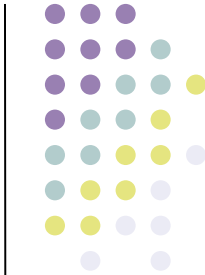
- Une variable déclarée dans une fonction n'existe donc que pendant que la fonction est en train d'être exécutée.

La portée des fonctions et variables



```
 1 int triple(int nombre);
 2
 3 int main()
 4 {
 5     cout << "Le triple de 15 est " << triple(15));
 6     cout << "Le triple de 15 est " << resultat; //  /!\
 7     // La ligne 6 plantera à la compilation
 8     return 0;
 9 }
10
11 int triple(int nombre)
12 {
13     int resultat = 0;
14     resultat = 3 * nombre;
15     return resultat;
16 }
```

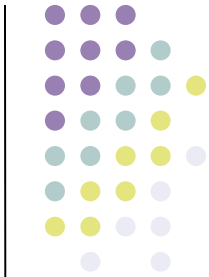
La portée des fonctions et variables



```
1  # include <iostream>
2  using namespace std;
3  // Une variable est connue seulement dans le bloc où elle est crée
4  int main()
5  {
6      int val = 7;
7      cout << "val vaut: " << val << endl;
8      {
9          int val = 5;
10         cout << "val vaut: "  << val << endl;
11     }
12     cout << "val vaut: "  << val << endl;
13     return 0;
14 }
```

```
Val vaut: 7
Val vaut: 5
Val vaut: 7
```

Variables globales à éviter



```
 1 #include <iostream>
 2 #include <stdlib.h>
 3 using namespace std;
 4
 5 int resultat = 0; // Déclaration de variable globale
 6
 7 void triple(int nombre); // Prototype de fonction
 8
 9 int main()
10 {
11     triple(15);  /* On appelle la fonction triple, qui modifie
12                     la variable globale resultat */
13     // On a accès à resultat
14     cout << "Le triple de 15 est: " << resultat << endl;
15     return 0;
16 }
17
18 void triple(int nombre)
19 {
20     resultat = 3 * nombre;
21 }
```

Variable globale

- **Variable globale accessible uniquement dans un fichier**

```
1 static int resultat = 0;
```

- **Variable statique à une fonction**

```
 1 # include <iostream>
 2 using namespace std;
 3
 4 int incremente();
 5
 6 int main()
 7 {
 8     cout << incremente() << endl;
 9     cout << incremente() << endl;
10     cout << incremente() << endl;
11     cout << incremente() << endl;
12     return 0;
13 }
14
15 int incremente()
16 {
17     static int nombre = 0;
18     nombre++;
19     return nombre;
20 }
```

```
1
2
3
4
```

Communication entre deux fonctions

+ Une fonction prends en entrée des arguments et donne en sortie des résultats

+ *int fonction(double P1, double P2, int P3, char P4)*
+ *En C++ une fonction peut prendre plusieurs valeur en entrée mais ne peut retourner qu'une seule valeur (avec return)*
+ *Comment on fait pour retourner 4 résultats comme dans le schémas :*
+ *Solution : donner en entrée des variables (utilisation de &) :*
+ *void fonction(double P1, double P2, int P3, char P4, int & R1, char & R2, double & R3, Point & R4)*

Exemple

+ Une fonction qui fait la division euclidienne :
  - Y = X x Q + R
+ La fonction prend deux en entrée X et Y (des entiers) et retourne Q et R :

```
void division(int Y, int X, int & Q, int & R)
{
   Q=Y/X ; R=Y%X ; // retours par variables
}
int main()
{
   int q, r ; // création des variables
   division(10,3,q,r) ;
   cout<<q<<" "<<r<<endl ;
}
```

des variables

Une fonction qui prend en argument un tableau d'entiers

+ Ecrire une fonction qui calcule les valeur min et max d'un tableau

```
void remplir_alea(int T[], int N, int a, int b) ;
void minmax(int T[], int N, int & mini, int & maxi)
{
   mini=maxi=T[0] ;
   for(int i=0;i<N;i++)
   {
       if(T[i]>maxi) maxi=T[i] ;
       else if(T[i]<mini) min=T[i] ;
   }
}
int main()
{
   int T[100] ;
   remplir_alea(T,20,1,10) ;
   int m, M ;
   minmax(T,20,m,M) ;
   cout<<m<<" "<<M<<endl ;
}
```