Allocation de la mémoire

1. dans le segment `data` (allocation statique) ;
2. dans la pile (le segment `stack` – allocation dynamique) ;
3. sur le tas (le segment `heap` – allocation « à la demande »).

```
1  #include <iostream>
2
3  using namespace std ;
4
5  int var1 ;
6
7  int main()
8  {
9    int var2 ;
10   int & var3 = * new int ;
11   delete & var3 ;
12   return 0 ;
13 }
```

La vie des variables / le flux du programme – théorie

1. les variables statiques naissent avant le début du programme et meurent après la fin du programme ;
2. les variables dynamiques naissent quand le flux du programme passe sur leur déclaration et meurent quand le flux du programme quitte le bloc où elles ont été déclarées ;
3. les variables allouées sur le tas naissent quand l'opérateur `new` est exécuté et meurent quand l'opérateur `delete` est exécuté.

La vie des variables / le flux du programme – exemple

```
 1 #include <iostream>
 2
 3 using namespace std ;
 4
 5 int var1 = 3 ;
 6
 7 int main()
 8 {
 9   for ( size_t i = 0 ; i < 10 ; ++ i )
10   {
11     int var2 = 12 ;
12   }
13   // Fuite de mémoire !
14   for ( size_t i = 0 ; i < 10 ; ++ i )
15   {
16     int & var3 = * new int ;
17   }
18   return 0 ;
19 }
```

Par défaut, les arguments sont recopiés dans la pile

```
 1 #include <iostream>
 2
 3 using namespace std ;
 4
 5 void exclamation(size_t nombre)
 6 {
 7   while ( nombre -- > 0 )
 8     cout << '!' ;
 9   cout << endl ;
10 }
11 int main()
12 {
13   exclamation(10) ;
14   return 0 ;
15 }
```

L'adresse des arguments peut être copiée dans la pile

```
 1 #include <iostream>
 2
 3 using namespace std ;
 4
 5 void exclamation(const size_t & nombre)
 6 {
 7   for ( size_t i = 0 ; i < nombre ; ++ i )
 8     cout << '!' ;
 9   cout << endl ;
10 }
11 int main()
12 {
13   exclamation(10) ;
14   return 0 ;
15 }
```

L'adresse d'un argument en sortie doit être copiée dans la pile

```
 1 #include <iostream>
 2 #include <cmath>
 3
 4 using namespace std ;
 5
 6 void racine(const double & x, double & r)
 7 {
 8   r = sqrt(x) ;
 9 }
10 int main()
11 {
12   double y ;
13   racine(2, y) ;
14   return 0 ;
15 }
```

Les attributs d'un objet

Les attributs d'un objet sont des variables – comme les champs d'un enregistrement

```
 1 #include <iostream>
 2 #include <string>
 3 #include <vector>
 4
 5 using namespace std ;
 6
 7 class Etudiant {
 8   string nom ;
 9   vector<double> notes ;
10 } ;
11 int main()
12 {
13   Etudiant moi ;
14   vector<Etudiant> master_pro_maserati ;
15   return 0 ;
16 }
```

Les constructeurs d'un objet

Les constructeurs d'un objet définissent toutes les façons d'initialiser un objet

```
 1 #include <iostream>
 2 #include <string>
 3 #include <vector>
 4
 5 using namespace std ;
 6
 7 class Etudiant {
 8   string nom ;
 9   vector<double> notes ;
10 public :
11   Etudiant (const string & arg1) : nom(arg1), notes() {}
12 } ;
13 int main()
14 {
15   Etudiant moi("Legendre") ;
16   vector<Etudiant> master_pro_maserati ;
17   master_pro_maserati.push_back(Etudiant("Martin")) ;
18   master_pro_maserati.push_back(Etudiant("Dupond")) ;
19   return 0 ;
20 }
```

Les méthodes d'un objet

Les méthodes d'un objet définissent toutes les façons d'utiliser un objet

```
 1 #include <iostream>
 2 #include <string>
 3 #include <vector>
 4
 5 using namespace std ;
 6
 7 class Etudiant {
 8   string nom ; vector<double> notes ;
 9 public :
10   Etudiant (const string & arg1) : nom(arg1), notes() {}
11   void nouvelle_note(double arg)
12   {
13   notes.push_back(arg) ;
14   }
15   size_t nombre_notes() const
16   {
17   return notes.size() ;
18   }
19 } ;
```

```
1 int main()
2 {
3   Etudiant moi("Legendre") ;
4   moi.nouvelle_note(19.5) ;
5   moi.nouvelle_note(19) ;
6   moi.nouvelle_note(20) ;
7   return 0 ;
8 }
```