

Corrigé des exercices du Cours 8

Le type caractère (`char`)

Les chaînes de caractères

Les structures

Vincent Lepetit

`vincent.lepetit@epfl.ch`

cherche_base

Écrivez une fonction `cherche_base` d'en-tête

```
int cherche_base(char * sequence, char base)
```

qui retourne le rang où apparaît pour la première fois la base représentée par le caractère `base` dans la séquence ADN `sequence`. Si la base n'apparaît pas, la fonction devra renvoyer -1 pour indiquer une erreur.

Exemples:

Après

```
int n = cherche_base("ATTGCC", 'C');  
n doit contenir 4;
```

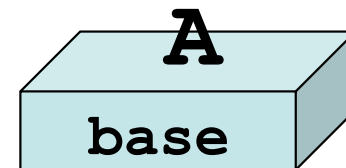
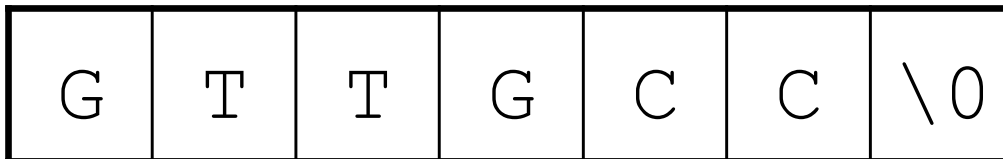
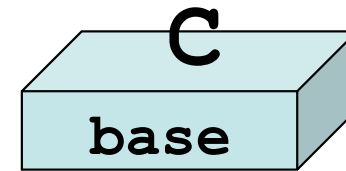
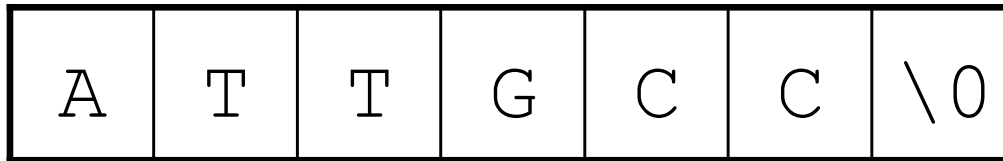
Après

```
int n = cherche_base("GTTGCC", 'A');  
n doit contenir -1.
```

cherche_base

```
int cherche_base(char * sequence, char base)
{
    for(int i = 0; i < strlen(sequence); i++)
        if (sequence[i] == base)
            return i;

    return -1;
}
```

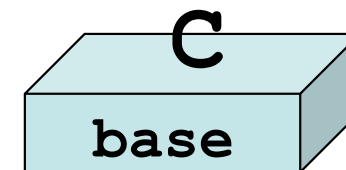
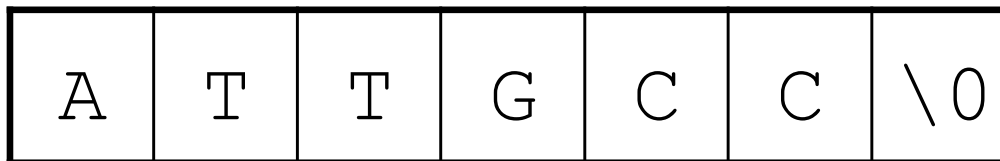


Attention

La fonction suivante ne marche pas:

```
int cherche_base(char * sequence, char base)
{
    for(int i = 0; i < strlen(sequence); i++)
        if (sequence[i] == base)
            return i;
    else
        return -1; // !!! FAUX
}
```

Dans le cas de l'exemple ci-dessous, on sort de la boucle quand *i* vaut 0 en renvoyant -1.



cherche_codon

Ecrivez une fonction `cherche_codon` d'en-tête

```
bool cherche_codon(char * sequence,  
                  char base1, char base2, char base3)
```

qui retourne `true` si le codon défini par les 3 caractères `base1 base2 base3` apparaît dans la séquence ADN `sequence`, et `false` sinon.

Exemples:

Après

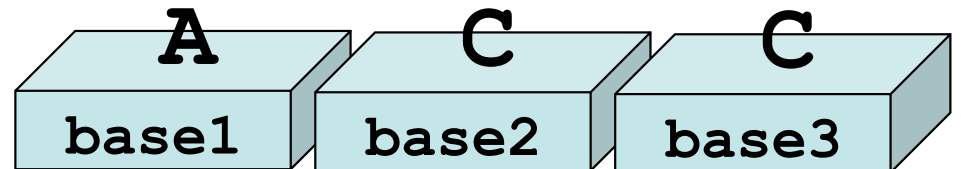
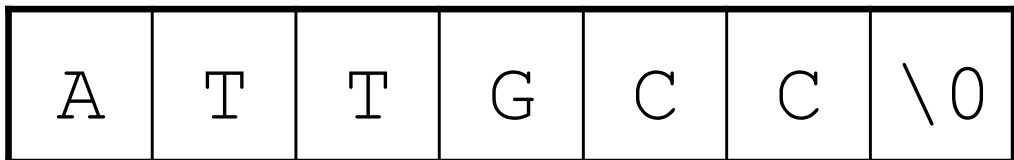
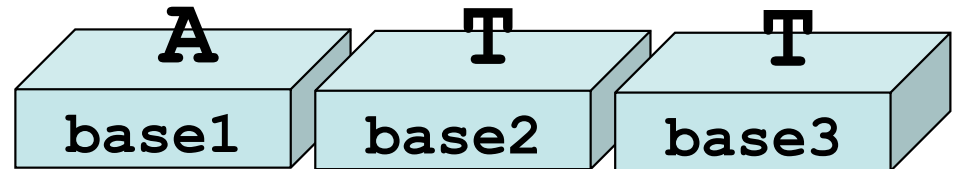
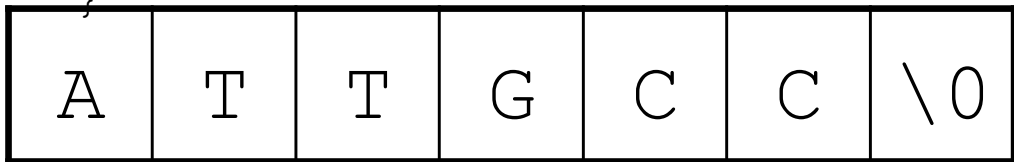
```
bool ok = cherche_codon("ATTGCC", 'A', 'T', 'T');  
ok doit contenir true;
```

Après

```
bool ok = cherche_codon("ATTGCC", 'A', 'C', 'C');  
ok doit contenir false.
```

cherche_codon

```
bool cherche_codon(char * sequence,  
                  char base1, char base2, char base3)  
{  
    for(int i = 0; i < strlen(sequence) - 2; i++)  
        if (sequence[i] == base1 &&  
            sequence[i + 1] == base2 &&  
            sequence[i + 2] == base3)  
            return true;  
  
    return false;  
}
```



complémentaire

Écrivez une fonction qui retourne le complémentaire d'une séquence ADN:
A devient T, T devient A, C devient G, G devient C.

Utilisez le prototype (= en-tête) suivant:

```
char * complementaire(char * seq_originale)
```

→ Il faut allouer une nouvelle chaîne de caractères dans laquelle vous stockerez la séquence complémentaire.

→ Après avoir calculé la séquence complémentaire, la fonction retournera un pointeur sur la nouvelle chaîne.

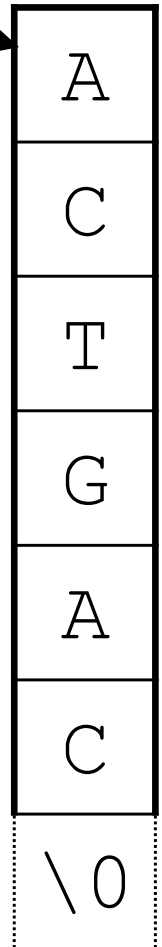
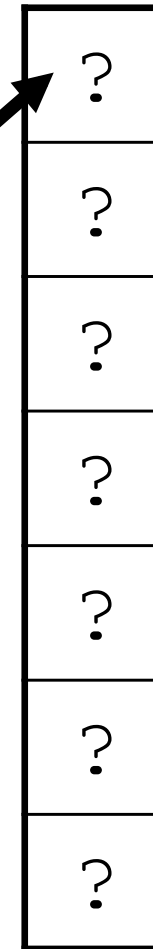
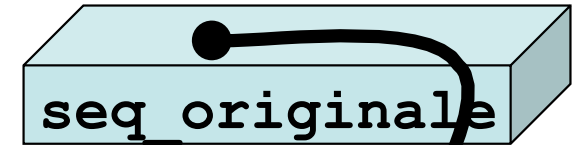
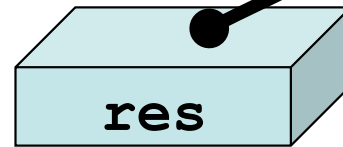
complémentaire

```
char * complémentaire(char * seq_originale)  
{
```

Commençons par allouer une chaîne de même taille que la chaîne `seq_originale` passée en paramètre.

Attention ! Il ne faut pas oublier de réserver une place de plus pour le caractère de fin de chaîne `'\0'` !

```
char * res = new char[strlen(seq_originale) + 1];
```

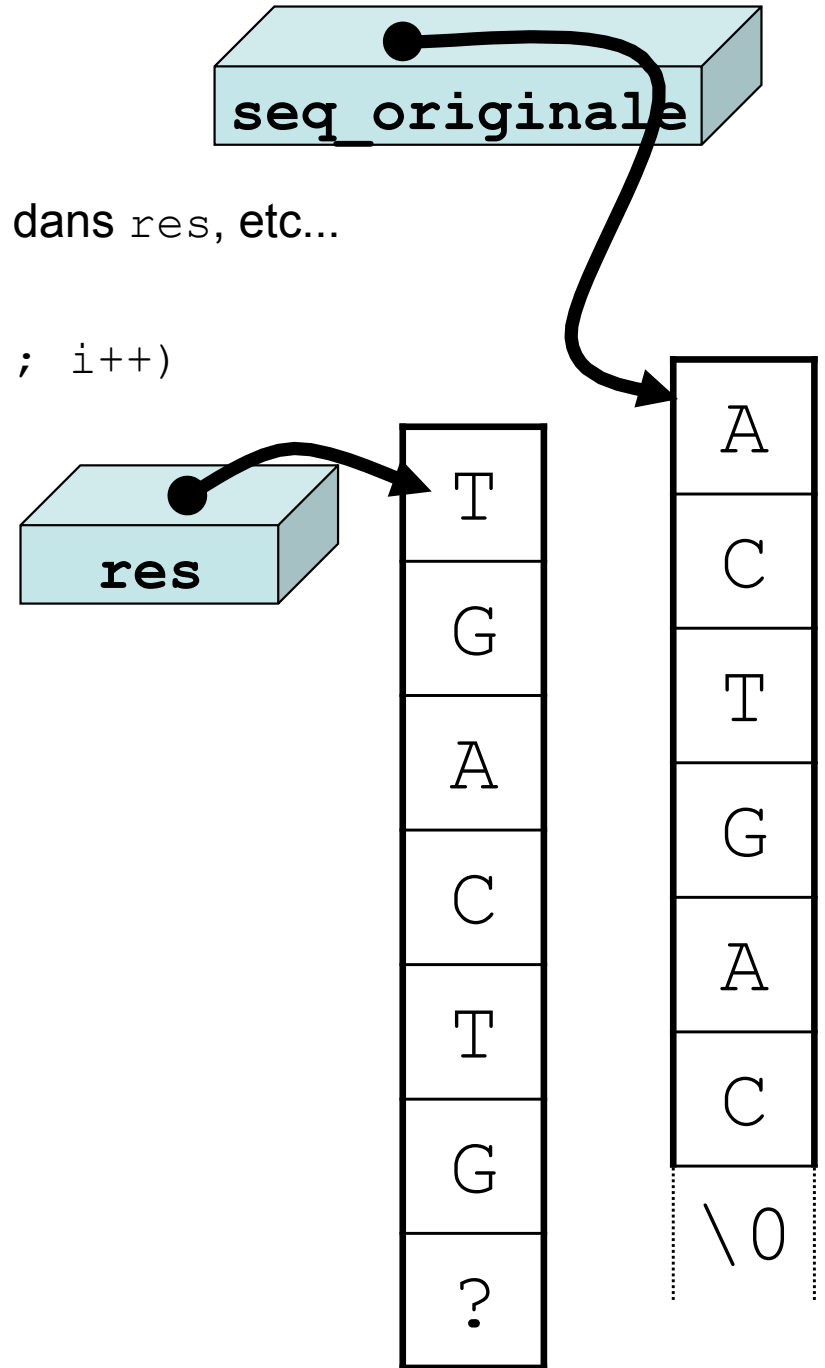


complémentaire

Parcourons les deux chaînes caractère par caractère:

Quand `seq_originale` contient un A, on place un T dans `res`, etc...

```
for(int i = 0; i < strlen(seq_originale); i++)
  switch(seq_originale[i])
  {
  case 'A':
    res[i] = 'T';
    break;
  case 'T':
    res[i] = 'A';
    break;
  (etc...)
  }
```



complémentaire

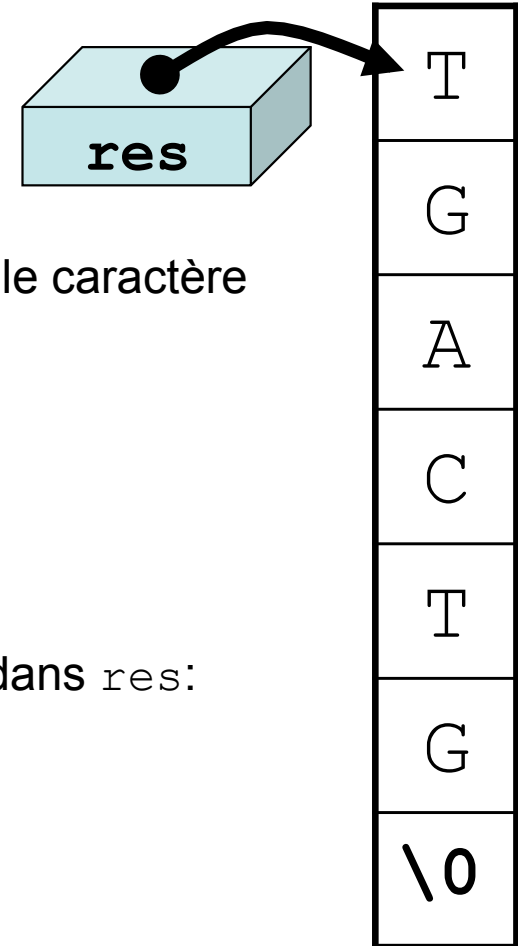
Attention, il y a un piège !

Il faut encore dire où s'arrête la chaîne `res`, c'est-à-dire placer le caractère `'\0'` au bon endroit:

```
res[strlen(seq_originale)] = '\0';
```

Il reste à retourner l'adresse de la mémoire allouée, contenue dans `res`:

```
return res;
```



complémentaire

```
char * complementaire(char * seq_originale)
{
    char * res = new char[strlen(seq_originale) + 1];

    for(int i = 0; i < strlen(seq_originale); i++)
        switch(seq_originale[i])
        {
            case 'A':
                res[i] = 'T';
                break;
            case 'T':
                res[i] = 'A';
                break;

            (etc...)
        }

    res[strlen(seq_originale)] = '\\0';

    return res;
}
```

Appel de complémentaire

```
char * complémentaire(char * seq_originale)
{
    char * res = new char[strlen(seq_originale) + 1];

    for(int i = 0; i < strlen(seq_originale); i++)
        switch(seq_originale[i])
        {
            case 'A':
                res[i] = 'T';
                break;
            case 'T':
                res[i] = 'A';
                break;

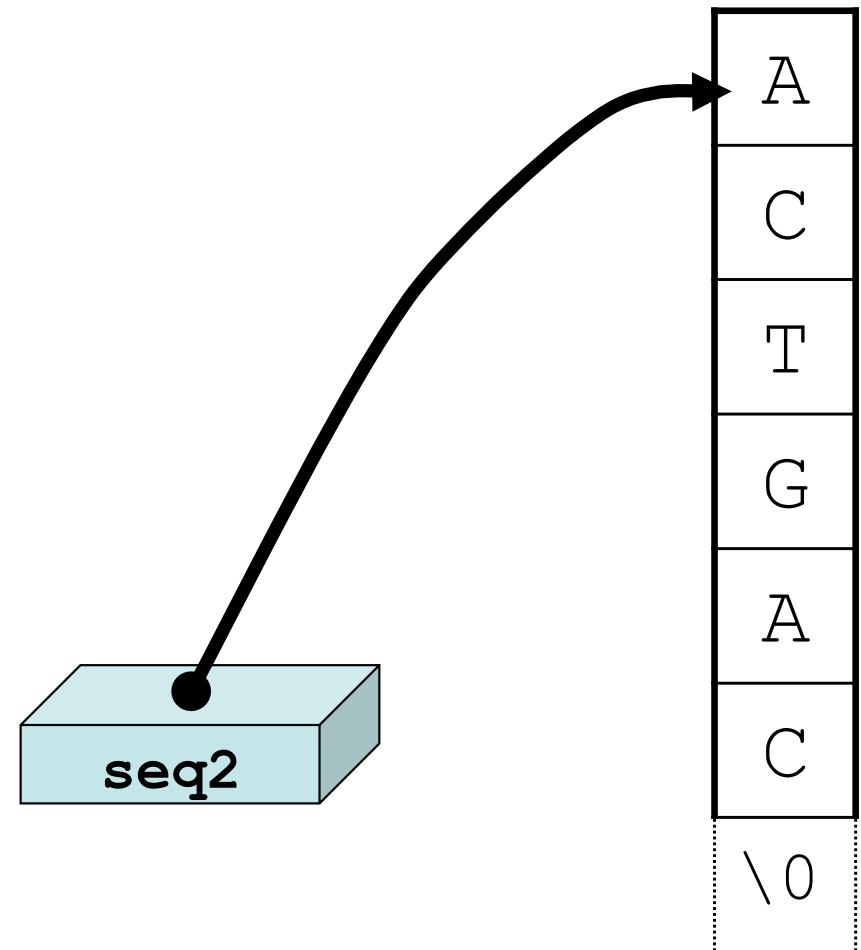
            (etc...)
        }

    res[strlen(seq_originale)] = '\0';

    return res;
}
(...)
```

APPEL:

➔ `char * seq3 = complémentaire(seq2);`



Appel de complémentaire

```
→ char * complémentaire(char * seq_originale)
{
    char * res = new char[strlen(seq_originale) + 1];

    for(int i = 0; i < strlen(seq_originale); i++)
        switch(seq_originale[i])
        {
            case 'A':
                res[i] = 'T';
                break;
            case 'T':
                res[i] = 'A';
                break;

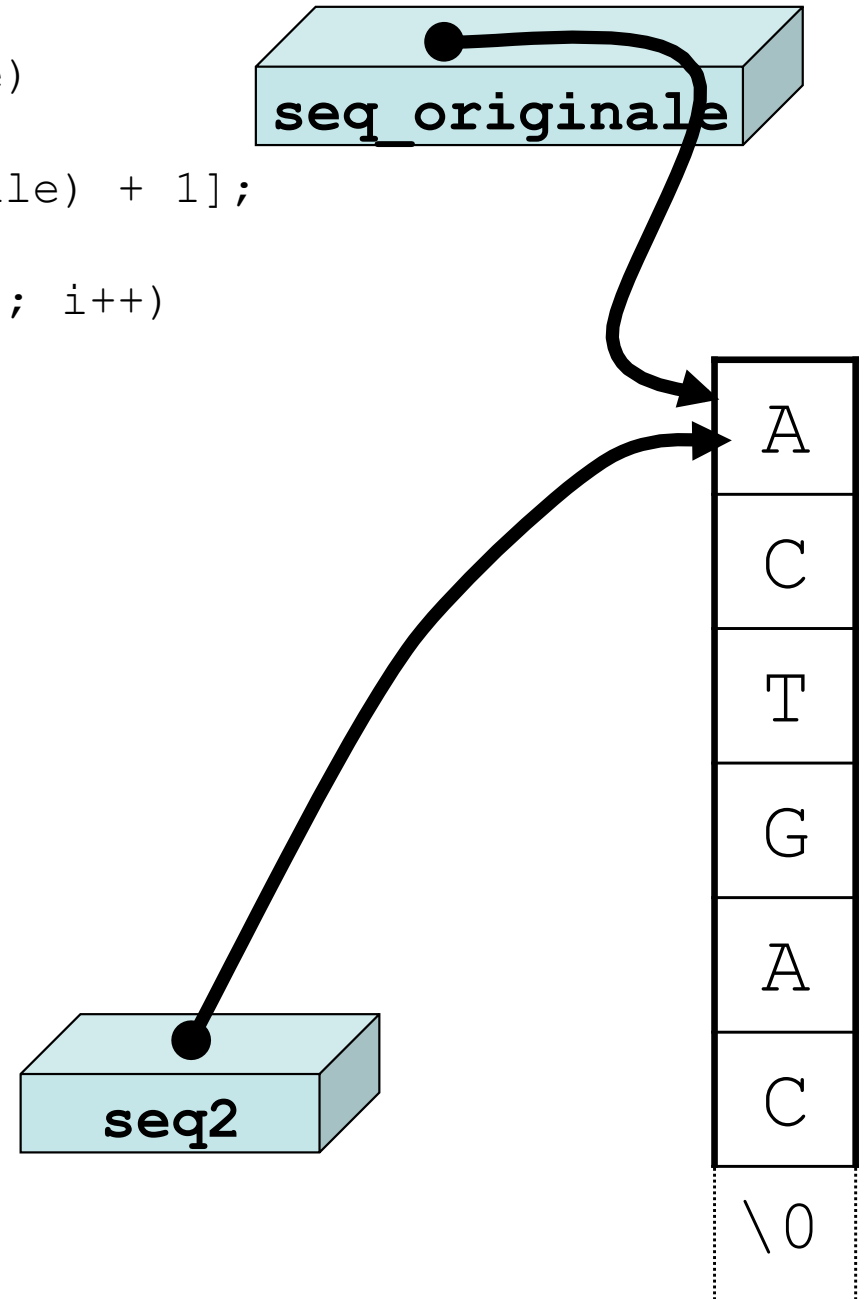
            (etc...)
        }

    res[strlen(seq_originale)] = '\0';

    return res;
}
(...)
```

APPEL:

```
→ char * seq3 = complémentaire(seq2);
```



Appel de complémentaire

```
char * complémentaire(char * seq_originale)
{
  → char * res = new char[strlen(seq_originale) + 1];

  for(int i = 0; i < strlen(seq_originale); i++)
    switch(seq_originale[i])
    {
    case 'A':
      res[i] = 'T';
      break;
    case 'T':
      res[i] = 'A';
      break;

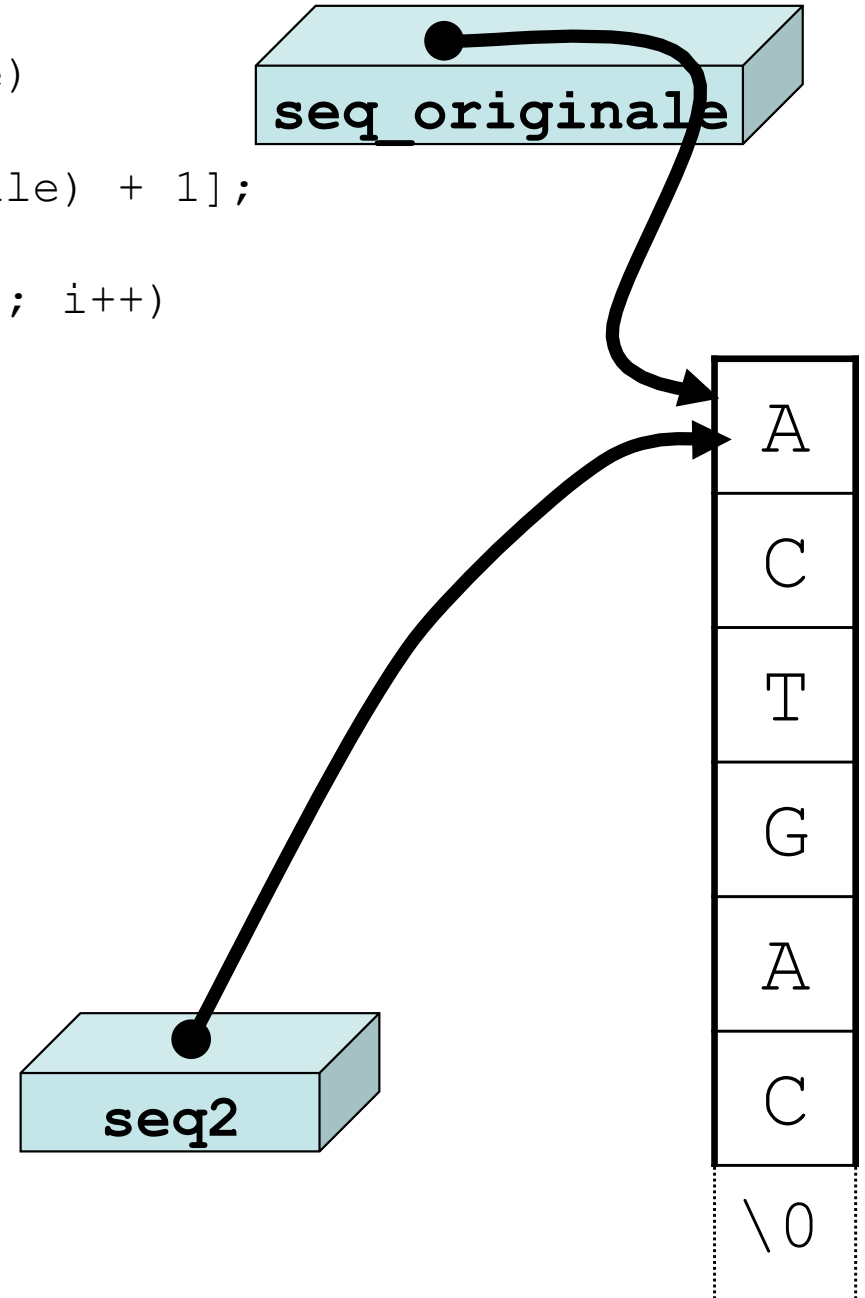
    (etc...)
    }

  res[strlen(seq_originale)] = '\0';

  return res;
}
(...)
```

APPEL:

```
→ char * seq3 = complémentaire(seq2);
```



Appel de complémentaire

```
char * complémentaire(char * seq_originale)
{
  → char * res = new char[strlen(seq_originale) + 1];

  for(int i = 0; i < strlen(seq_originale); i++)
    switch(seq_originale[i])
    {
    case 'A':
      res[i] = 'T';
      break;
    case 'T':
      res[i] = 'A';
      break;

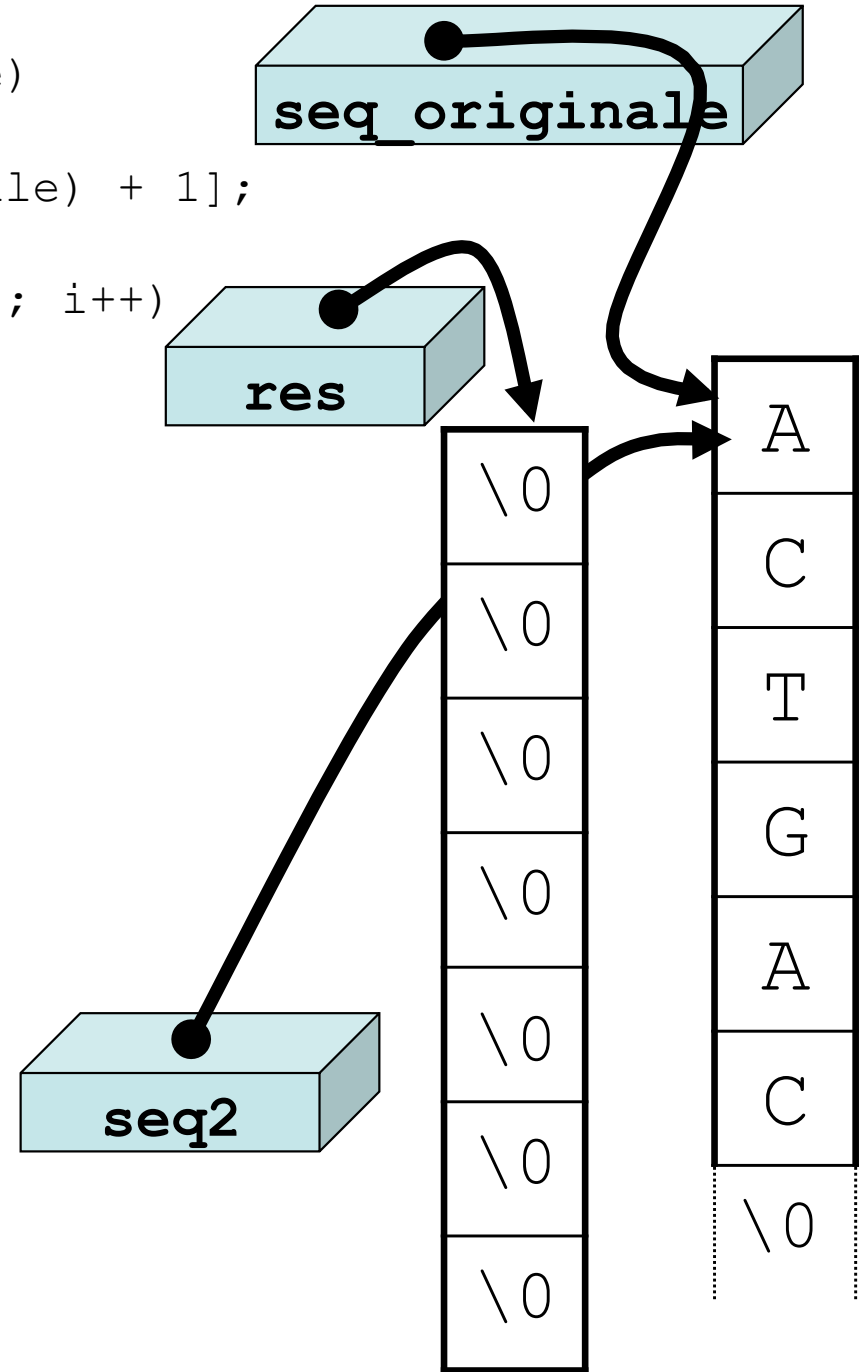
    (etc...)
    }

  res[strlen(seq_originale)] = '\\0';

  return res;
}
(...)
```

APPEL:

→ char * seq3 = complémentaire(seq2);



Appel de complémentaire

```
char * complémentaire(char * seq_originale)
{
    char * res = new char[strlen(seq_originale) + 1];
    for(int i = 0; i < strlen(seq_originale); i++)
        switch(seq_originale[i])
        {
            case 'A':
                res[i] = 'T';
                break;
            case 'T':
                res[i] = 'A';
                break;

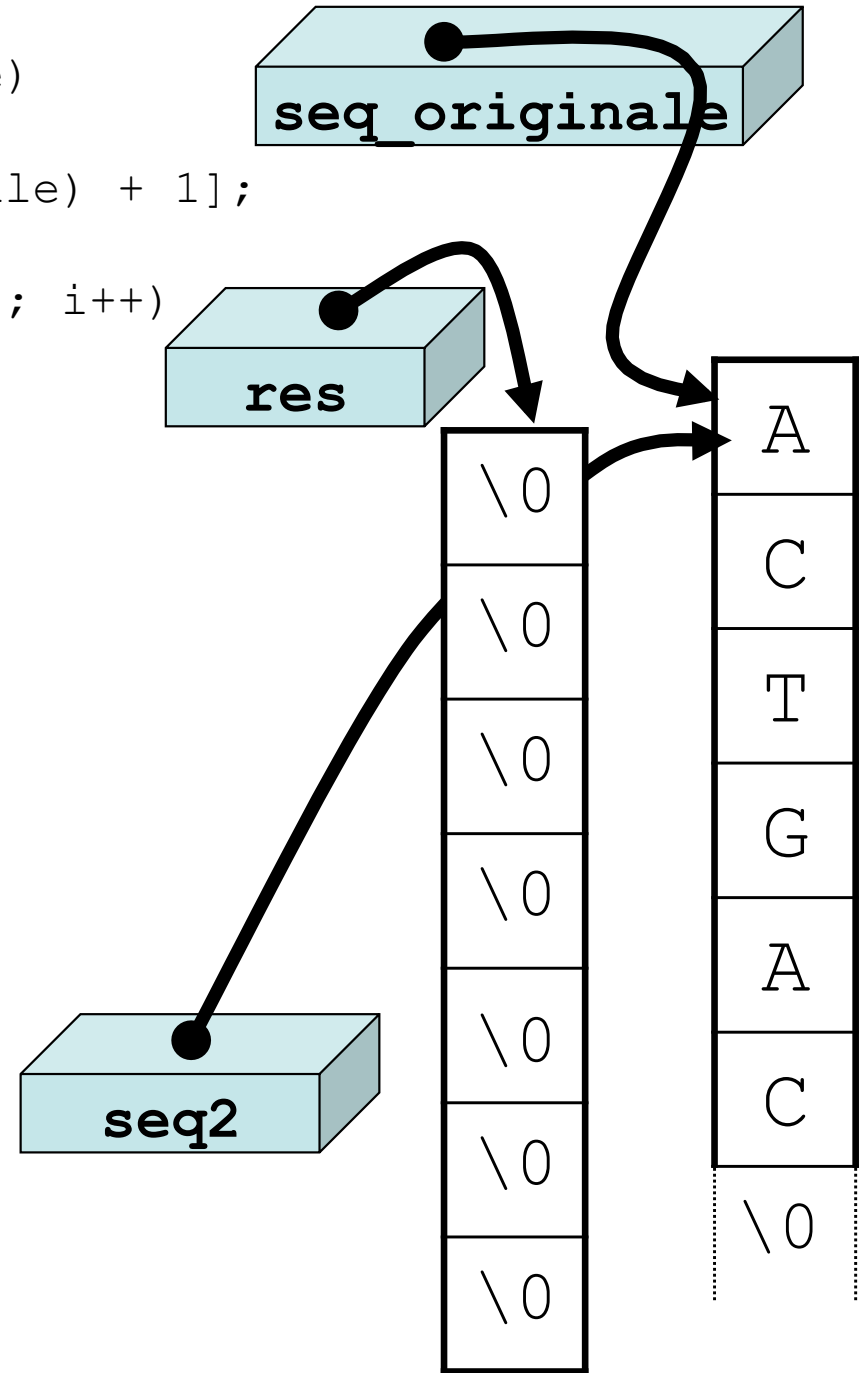
            (etc...)
        }

    res[strlen(seq_originale)] = '\0';

    return res;
}
(...)
```

APPEL:

→ char * seq3 = complémentaire(seq2);



Appel de complémentaire

```
char * complémentaire(char * seq_originale)
{
    char * res = new char[strlen(seq_originale) + 1];

    for(int i = 0; i < strlen(seq_originale); i++)
        switch(seq_originale[i])
        {
            case 'A':
                res[i] = 'T';
                break;
            case 'T':
                res[i] = 'A';
                break;

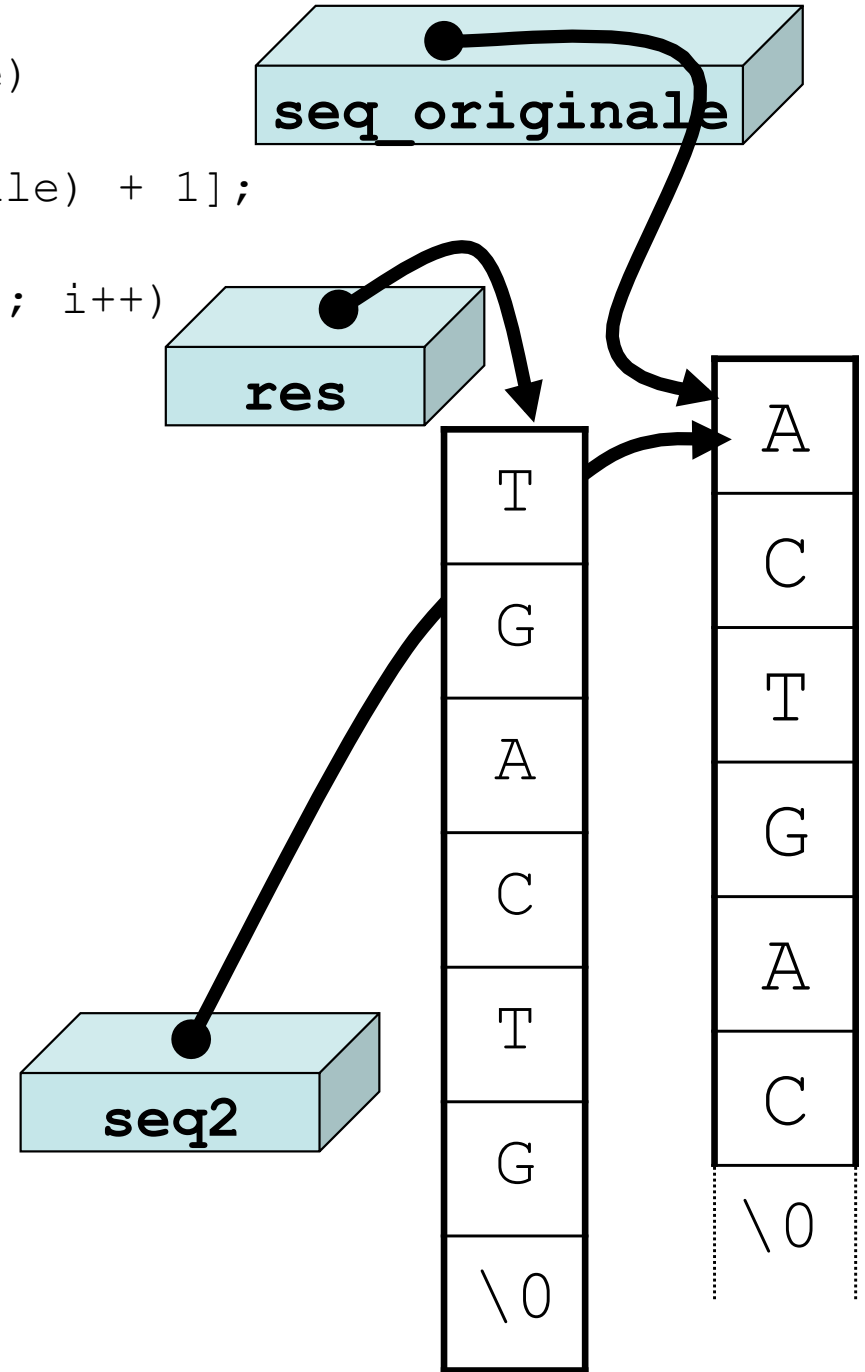
            (etc...)
        }

    res[strlen(seq_originale)] = '\0';

    return res;
}
(...)
```

APPEL:

→ char * seq3 = complémentaire(seq2);



Appel de complémentaire

```
char * complémentaire(char * seq_originale)
{
    char * res = new char[strlen(seq_originale) + 1];

    for(int i = 0; i < strlen(seq_originale); i++)
        switch(seq_originale[i])
        {
            case 'A':
                res[i] = 'T';
                break;
            case 'T':
                res[i] = 'A';
                break;

            (etc...)
        }
}
```

➔ `res[strlen(seq_originale)] = '\\0';`

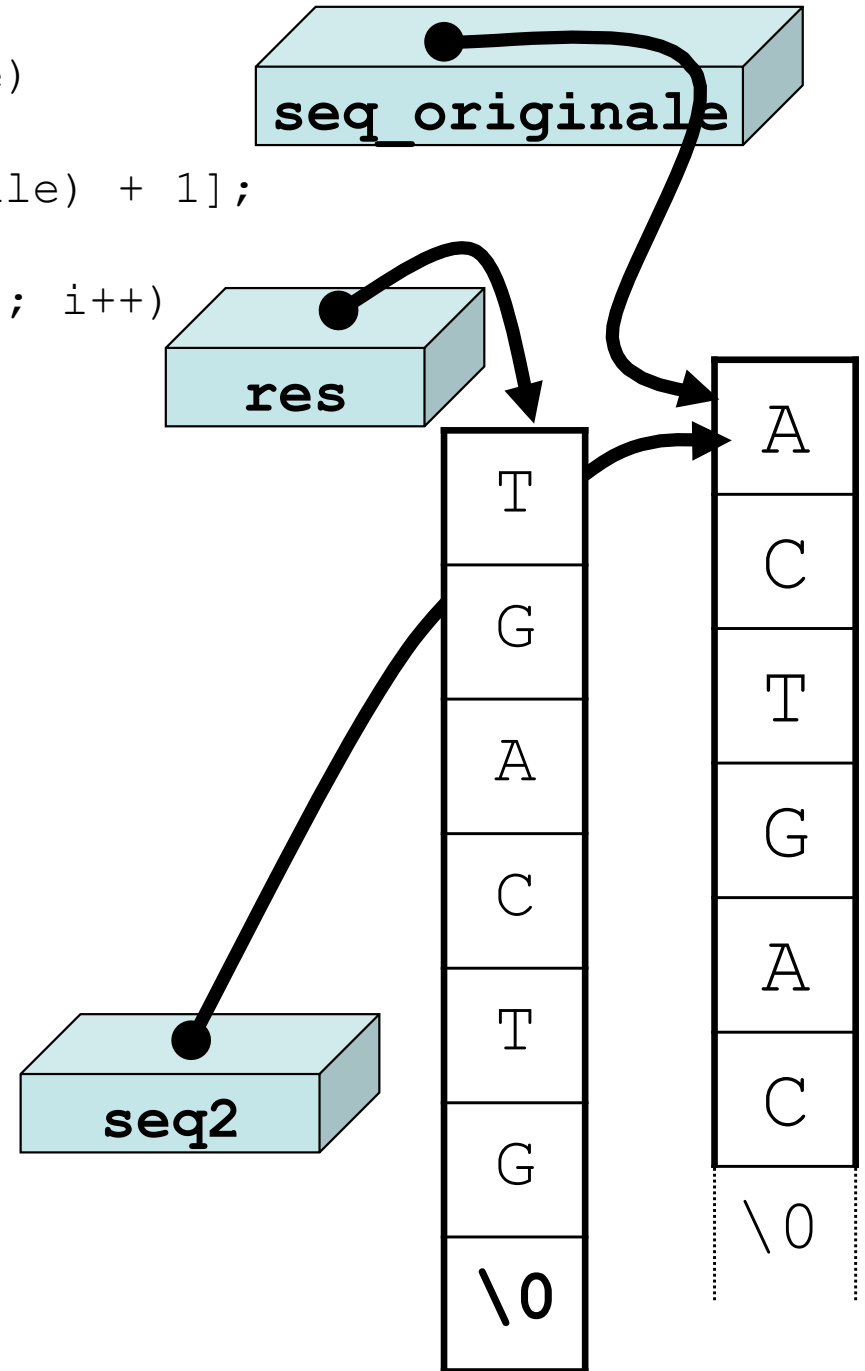
```
return res;
```

```
}
```

```
(...)
```

APPEL:

➔ `char * seq3 = complémentaire(seq2);`



Appel de complémentaire

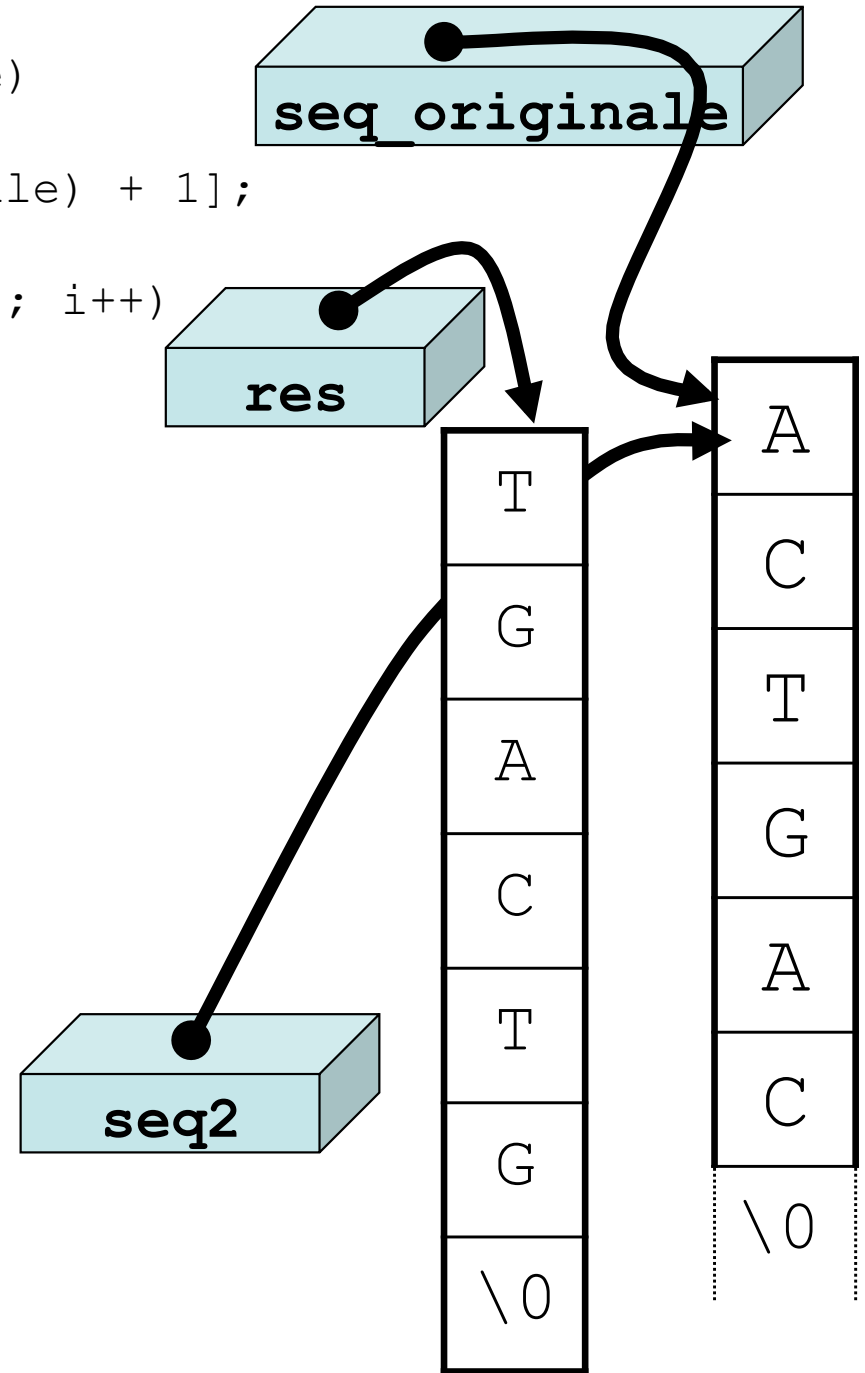
```
char * complémentaire(char * seq_originale)
{
    char * res = new char[strlen(seq_originale) + 1];

    for(int i = 0; i < strlen(seq_originale); i++)
        switch(seq_originale[i])
        {
            case 'A':
                res[i] = 'T';
                break;
            case 'T':
                res[i] = 'A';
                break;

            (etc...)
        }

    res[strlen(seq_originale)] = '\0';

    → return res;
}
(...)
APPEL:
→ char * seq3 = complémentaire(seq2);
```



Appel de complémentaire

```
char * complémentaire(char * seq_originale)
{
    char * res = new char[strlen(seq_originale) + 1];

    for(int i = 0; i < strlen(seq_originale); i++)
        switch(seq_originale[i])
        {
            case 'A':
                res[i] = 'T';
                break;
            case 'T':
                res[i] = 'A';
                break;

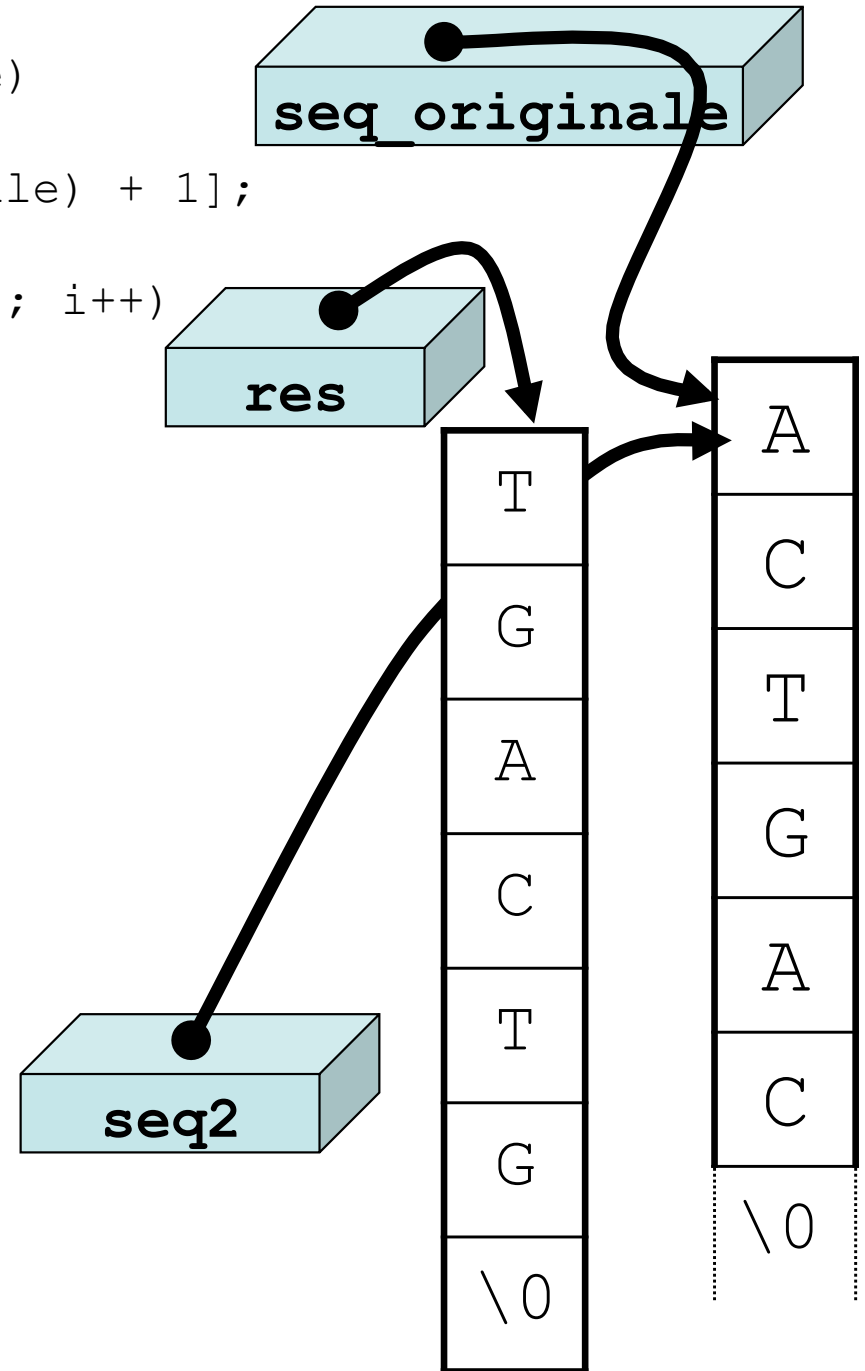
            (etc...)
        }

    res[strlen(seq_originale)] = '\0';

    return res;
}
(...)
```

APPEL:

➔ `char * seq3 = complémentaire(seq2);`



Appel de complémentaire

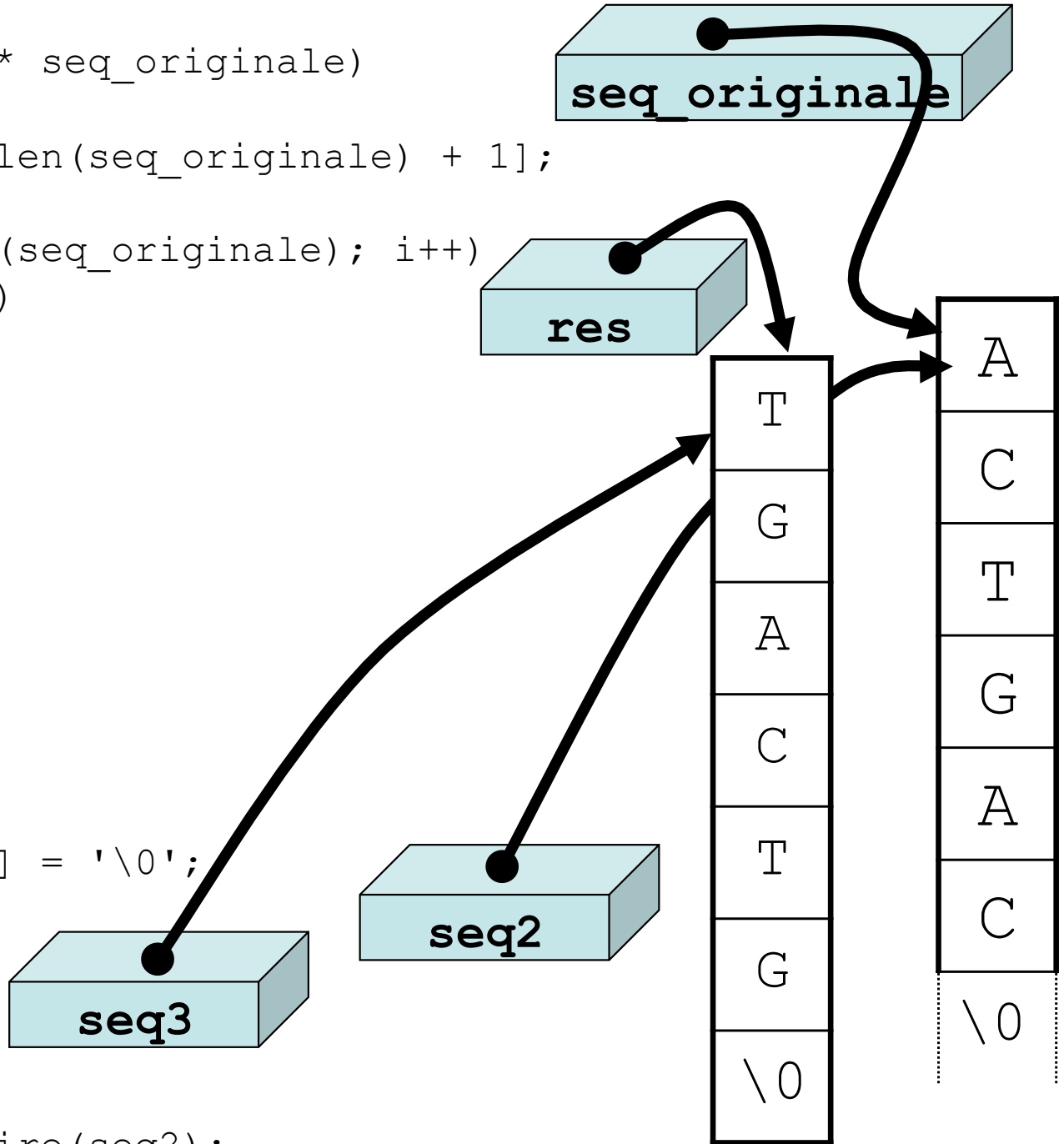
```
char * complémentaire(char * seq_originale)
{
    char * res = new char[strlen(seq_originale) + 1];

    for(int i = 0; i < strlen(seq_originale); i++)
        switch(seq_originale[i])
        {
            case 'A':
                res[i] = 'T';
                break;
            case 'T':
                res[i] = 'A';
                break;

            (etc...)
        }

    res[strlen(seq_originale)] = '\0';

    return res;
}
(...)
```



APPEL:

➔ char * seq3 = complémentaire(seq2);

Appel de complémentaire

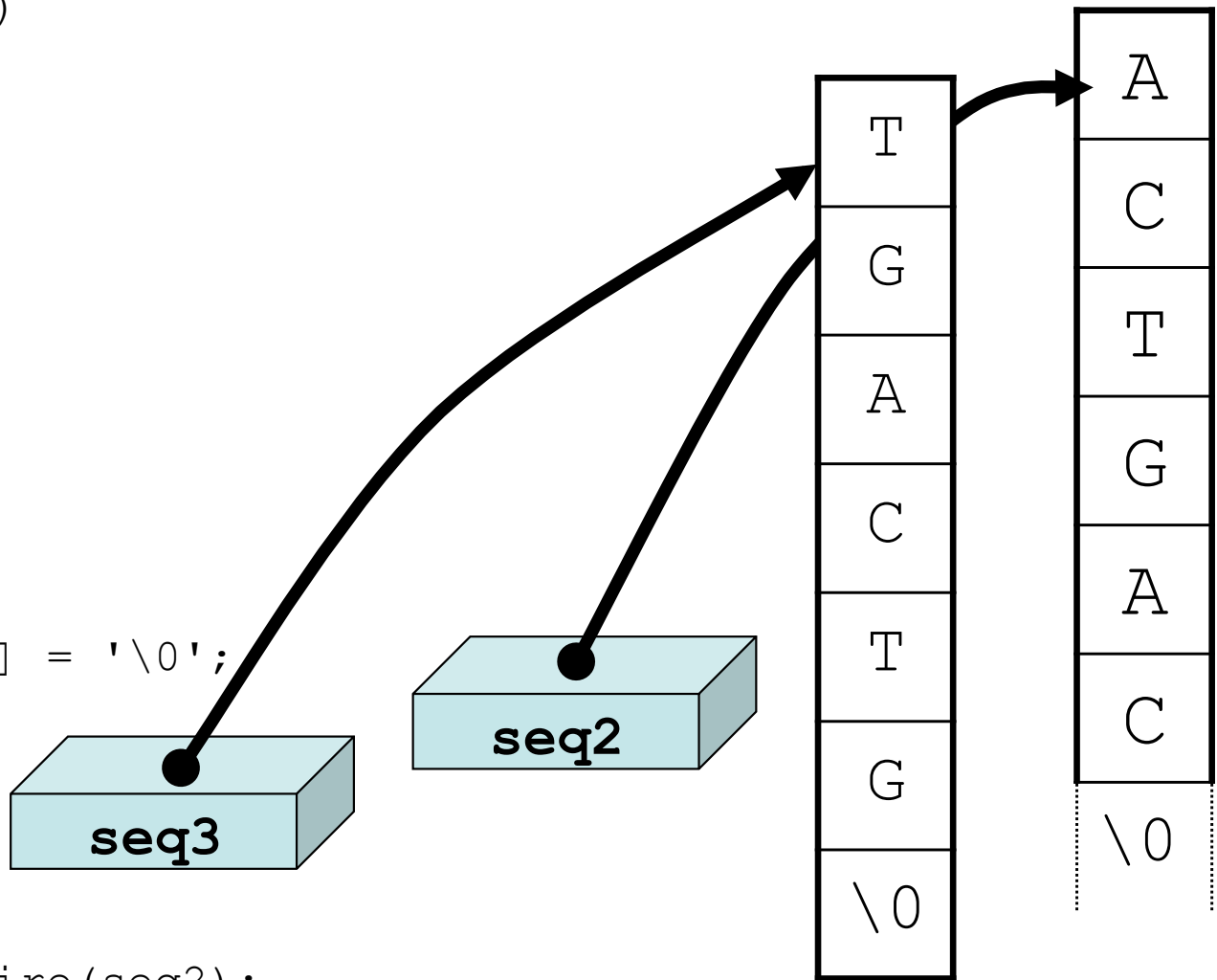
```
char * complémentaire(char * seq_originale)
{
    char * res = new char[strlen(seq_originale) + 1];

    for(int i = 0; i < strlen(seq_originale); i++)
        switch(seq_originale[i])
        {
            case 'A':
                res[i] = 'T';
                break;
            case 'T':
                res[i] = 'A';
                break;

            (etc...)
        }

    res[strlen(seq_originale)] = '\0';

    return res;
}
(...)
```



APPEL:

➔ char * seq3 = complémentaire(seq2);

Qu'affiche ce programme ?

```
struct A
{
    int n;
    float f;
};

struct B
{
    A x;
    float f;
};

A f(int m, float g)
{
    A z;
    z.n = m;
    z.f = g;
    return z;
}
```

```
...

int n = 1;
A x;
x.n = 2;
x.f = 3;
cout << n << " " << x.n << endl;

B y;
y.x = x;
x.f = 2;
cout << y.x.f << endl;

x = f(1, 2);
cout << x.n << " " << x.f << endl;
```

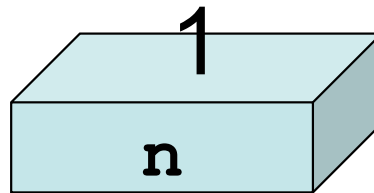
```
struct A
{
    int n;
    float f;
};
```

```
struct B
{
    A x;
    float f;
};
```

```
A f(int m, float g)
{
    A z;
    z.n = m;
    z.f = g;
    return z;
}
```

...

```
→ int n = 1;
   A x;
   x.n = 2;
   x.f = 3;
   cout << n << " " << x.n << endl;
```



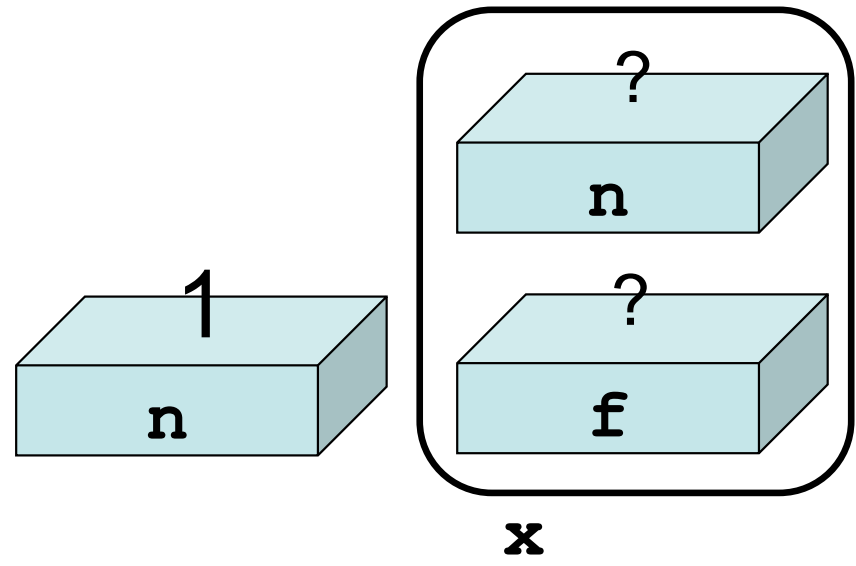
```
struct A
{
    int n;
    float f;
};
```

```
struct B
{
    A x;
    float f;
};
```

```
A f(int m, float g)
{
    A z;
    z.n = m;
    z.f = g;
    return z;
}
```

...

```
int n = 1;
A x;
x.n = 2;
x.f = 3;
cout << n << " " << x.n << endl;
```



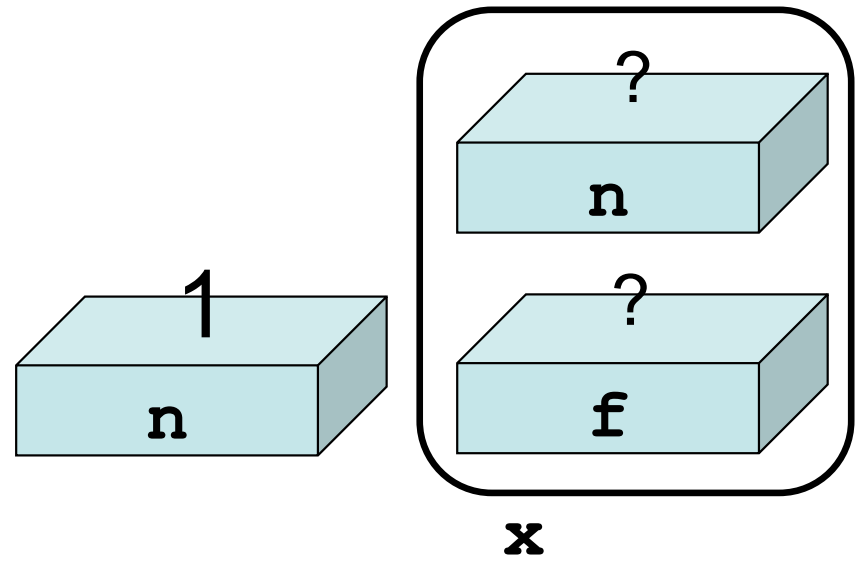
```
struct A
{
    int n;
    float f;
};
```

```
struct B
{
    A x;
    float f;
};
```

```
A f(int m, float g)
{
    A z;
    z.n = m;
    z.f = g;
    return z;
}
```

...

```
int n = 1;
A x;
→ x.n = 2;
x.f = 3;
cout << n << " " << x.n << endl;
```



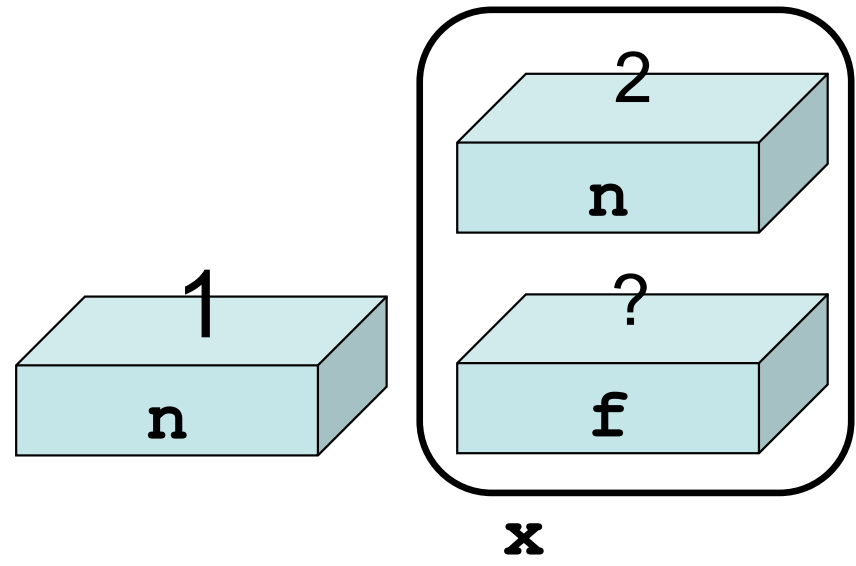
```
struct A
{
    int n;
    float f;
};
```

```
struct B
{
    A x;
    float f;
};
```

```
A f(int m, float g)
{
    A z;
    z.n = m;
    z.f = g;
    return z;
}
```

...

```
int n = 1;
A x;
→ x.n = 2;
x.f = 3;
cout << n << " " << x.n << endl;
```



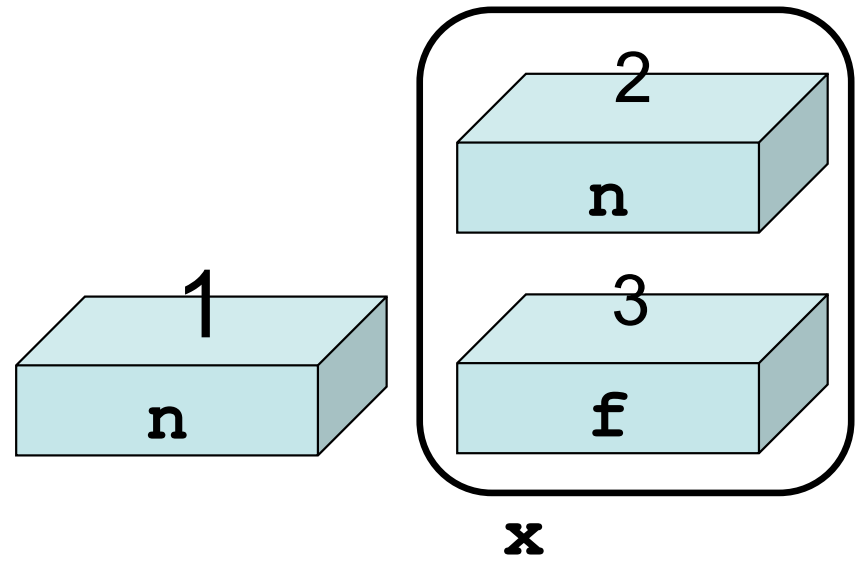
```
struct A
{
    int n;
    float f;
};
```

```
struct B
{
    A x;
    float f;
};
```

```
A f(int m, float g)
{
    A z;
    z.n = m;
    z.f = g;
    return z;
}
```

...

```
int n = 1;
A x;
x.n = 2;
x.f = 3;
cout << n << " " << x.n << endl;
```



```
struct A
{
    int n;
    float f;
};
```

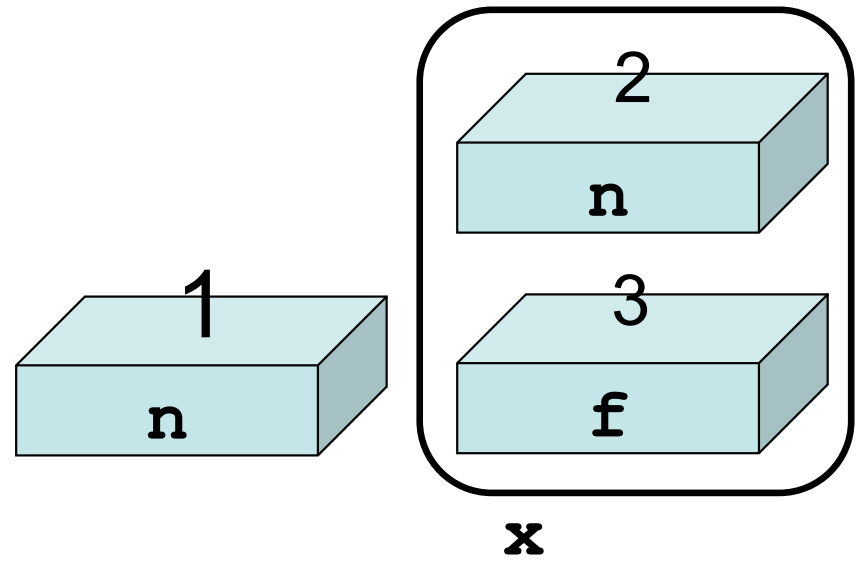
```
struct B
{
    A x;
    float f;
};
```

```
A f(int m, float g)
{
    A z;
    z.n = m;
    z.f = g;
    return z;
}
```

...

```
int n = 1;
A x;
x.n = 2;
x.f = 3;
```

```
→ cout << n << " " << x.n << endl;
```



1 2

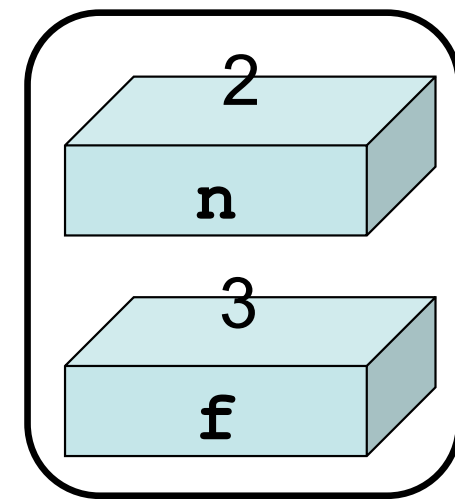
```
struct A
{
    int n;
    float f;
};
```

```
struct B
{
    A x;
    float f;
};
```

```
A f(int m, float g)
{
    A z;
    z.n = m;
    z.f = g;
    return z;
}
```

...

```
→ B y;
y.x = x;
x.f = 2;
cout << y.x.f << endl;
```



x



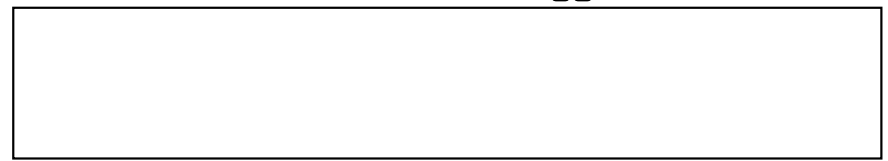
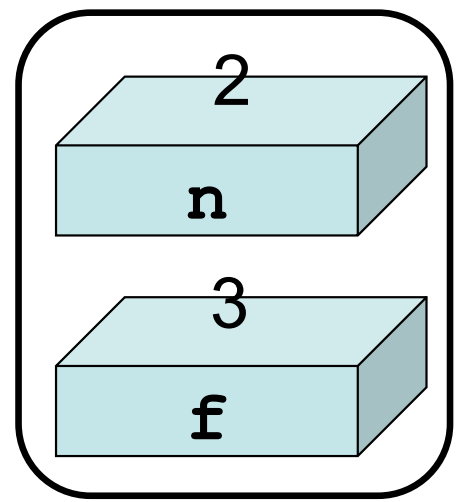
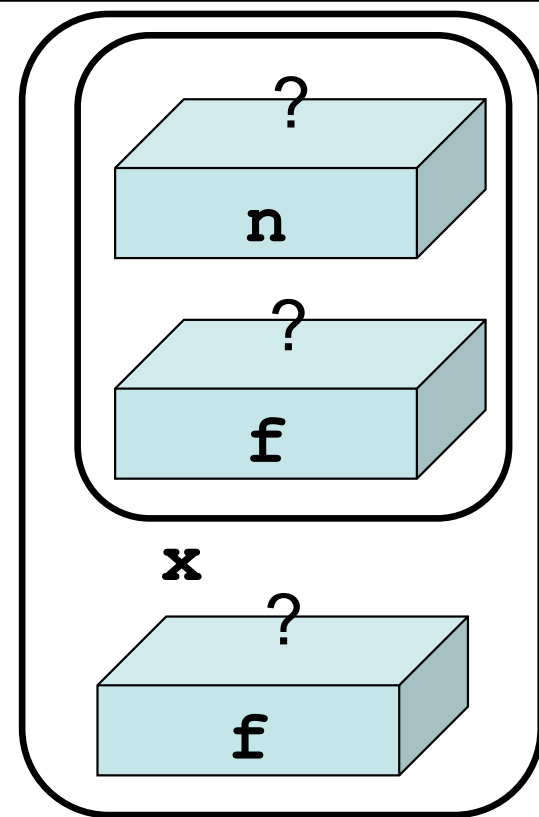
```
struct A
{
    int n;
    float f;
};
```

```
struct B
{
    A x;
    float f;
};
```

```
A f(int m, float g)
{
    A z;
    z.n = m;
    z.f = g;
    return z;
}
```

...

```
→ B y;
y.x = x;
x.f = 2;
cout << y.x.f << endl;
```

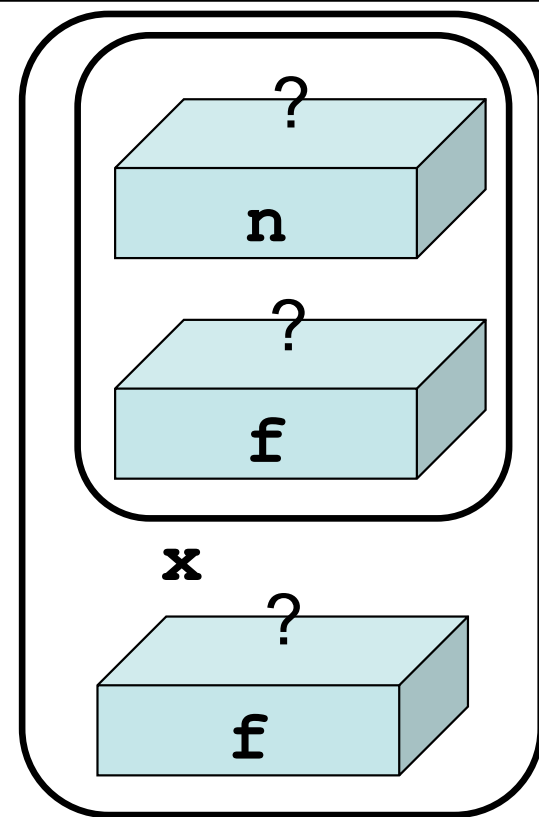


```
struct A
{
    int n;
    float f;
};

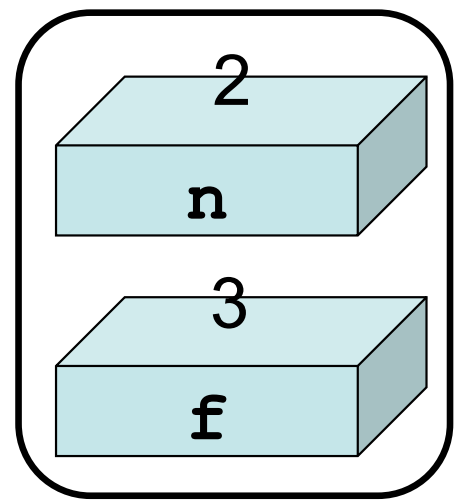
struct B
{
    A x;
    float f;
};

A f(int m, float g)
{
    A z;
    z.n = m;
    z.f = g;
    return z;
}
```

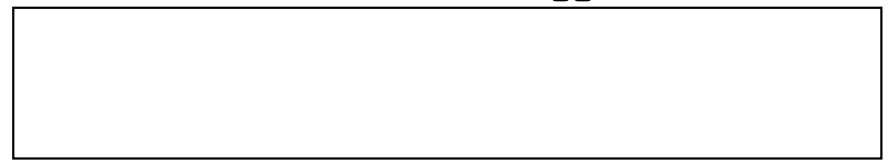
```
...
B y;
→ y.x = x;
x.f = 2;
cout << y.x.f << endl;
```



y



x

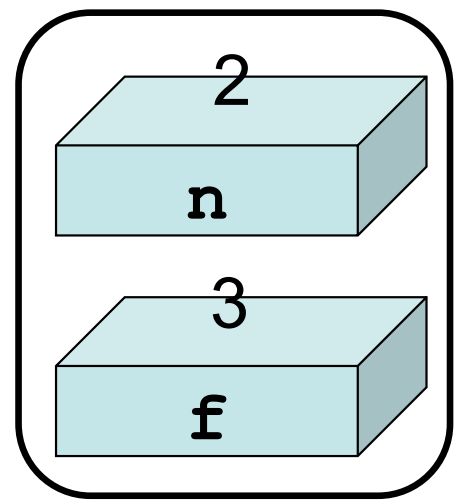
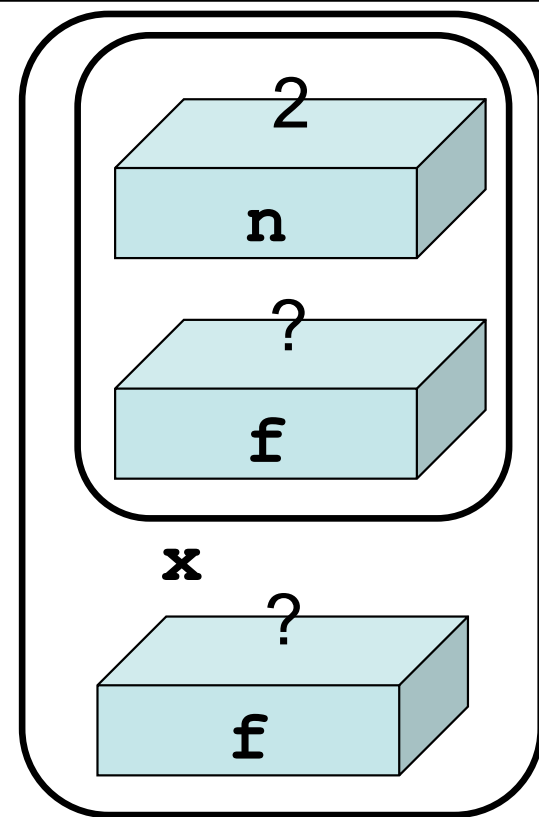


```
struct A
{
    int n;
    float f;
};

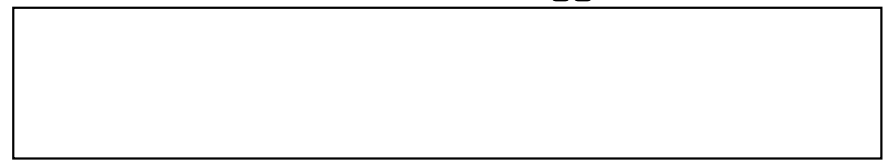
struct B
{
    A x;
    float f;
};

A f(int m, float g)
{
    A z;
    z.n = m;
    z.f = g;
    return z;
}
```

```
...
B y;
→ y.x = x;
  x.f = 2;
cout << y.x.f << endl;
```



x



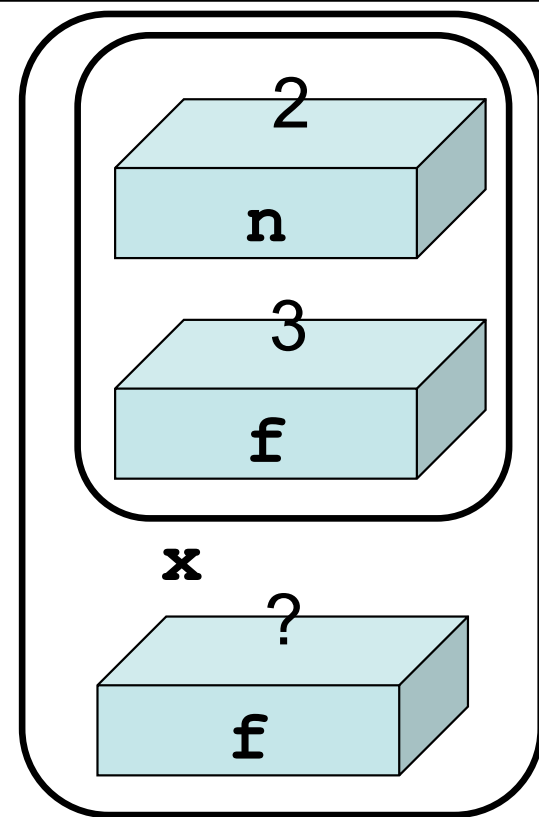
```
struct A
{
  int n;
  float f;
};
```

```
struct B
{
  A x;
  float f;
};
```

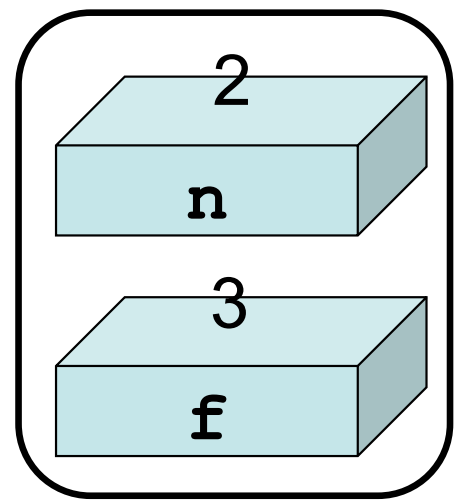
```
A f(int m, float g)
{
  A z;
  z.n = m;
  z.f = g;
  return z;
}
```

...

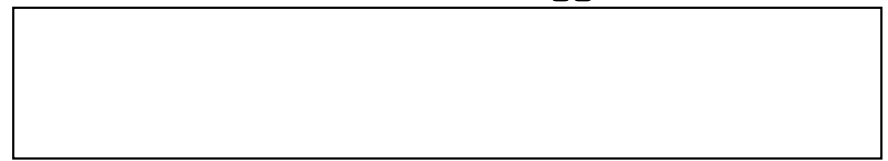
```
B y;
y.x = x;
x.f = 2;
cout << y.x.f << endl;
```



y



x



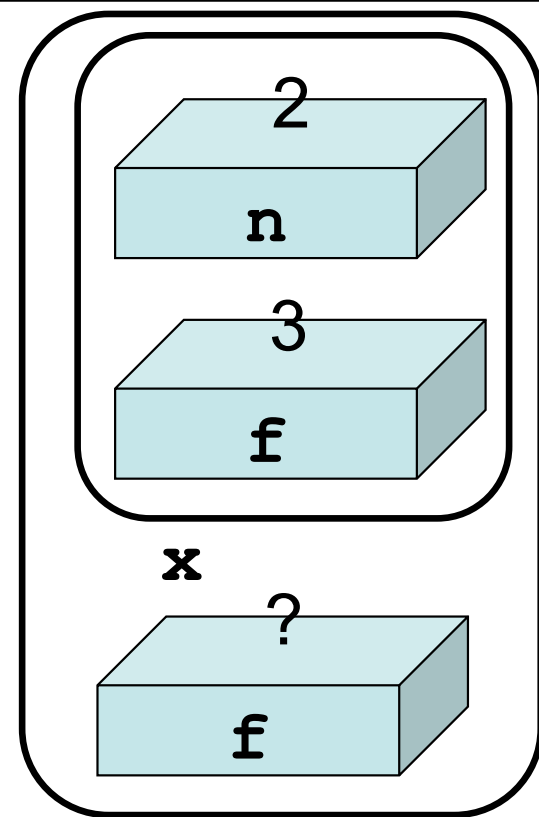
```
struct A
{
    int n;
    float f;
};
```

```
struct B
{
    A x;
    float f;
};
```

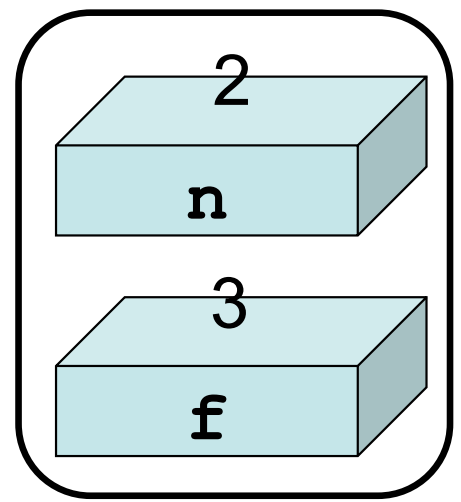
```
A f(int m, float g)
{
    A z;
    z.n = m;
    z.f = g;
    return z;
}
```

...

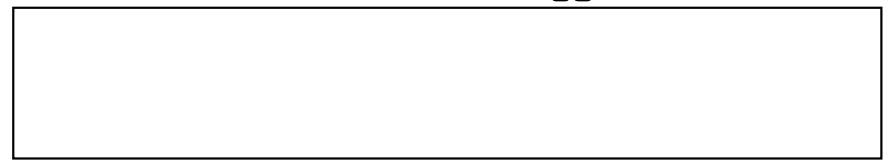
```
B y;
y.x = x;
x.f = 2;
cout << y.x.f << endl;
```



y



x



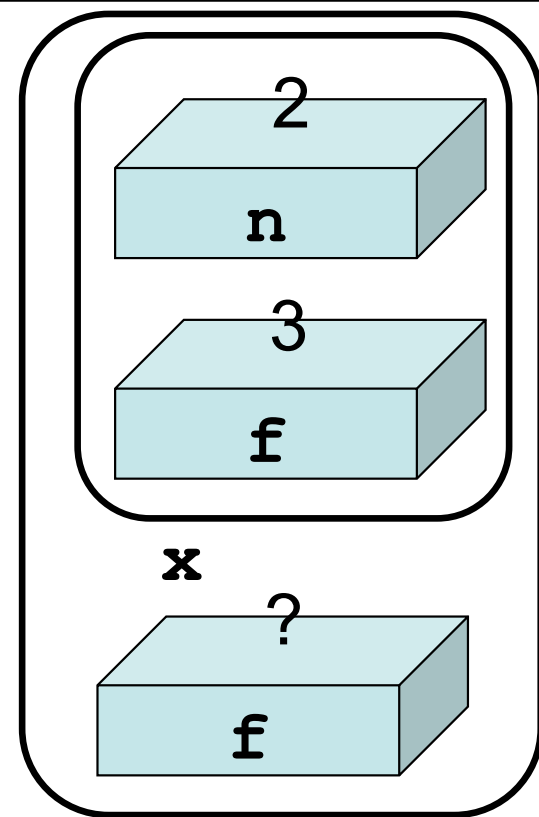
```
struct A
{
    int n;
    float f;
};
```

```
struct B
{
    A x;
    float f;
};
```

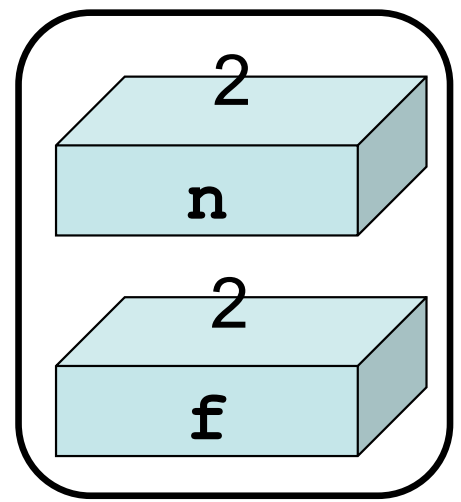
```
A f(int m, float g)
{
    A z;
    z.n = m;
    z.f = g;
    return z;
}
```

...

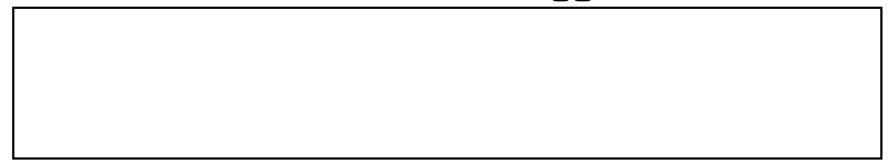
```
B y;
y.x = x;
x.f = 2;
cout << y.x.f << endl;
```



y



x



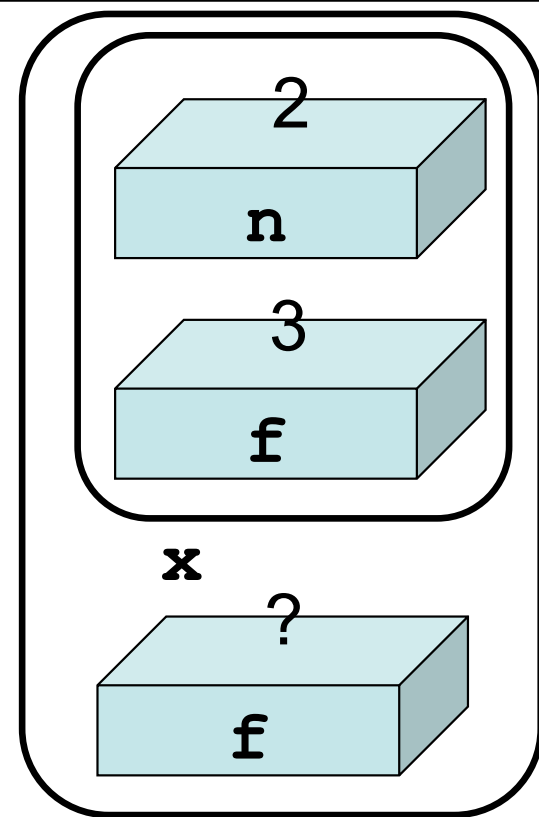
```
struct A
{
    int n;
    float f;
};

struct B
{
    A x;
    float f;
};

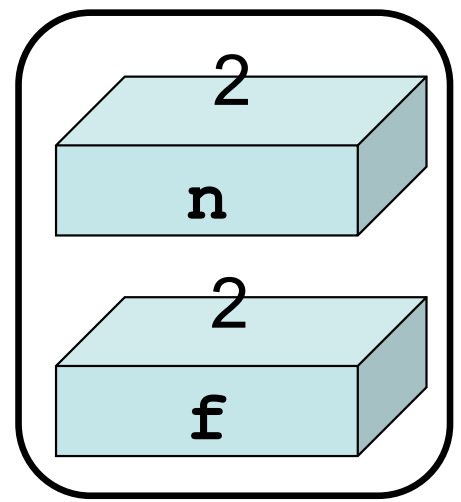
A f(int m, float g)
{
    A z;
    z.n = m;
    z.f = g;
    return z;
}
```

```
...

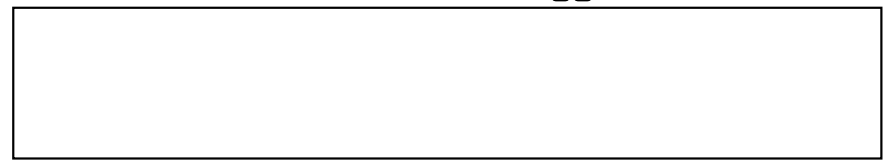
B y;
y.x = x;
x.f = 2;
cout << y.x.f << endl;
```



y



x



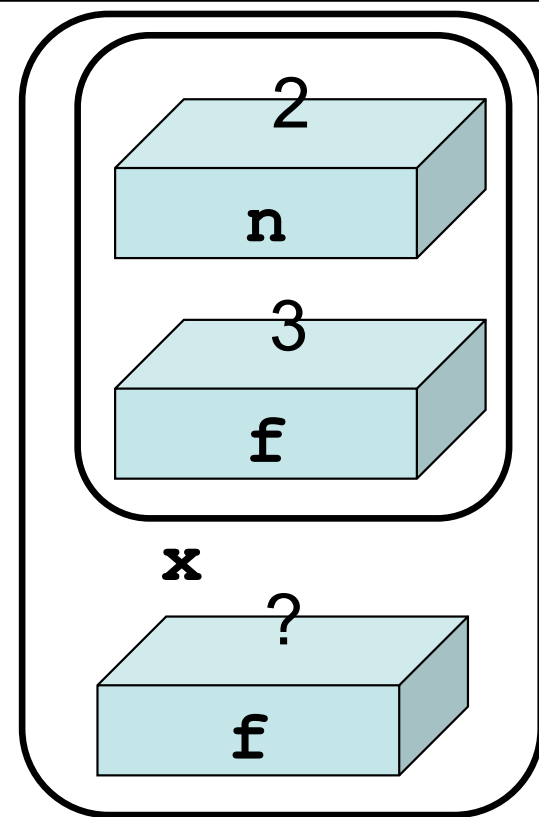
```
struct A
{
    int n;
    float f;
};

struct B
{
    A x;
    float f;
};

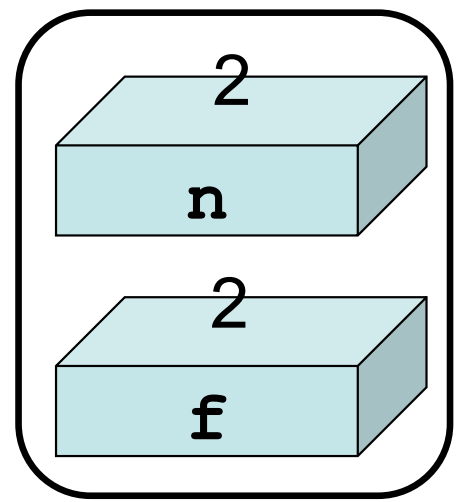
A f(int m, float g)
{
    A z;
    z.n = m;
    z.f = g;
    return z;
}
```

...

```
B y;
y.x = x;
x.f = 2;
cout << y.x.f << endl;
```



y



x

3

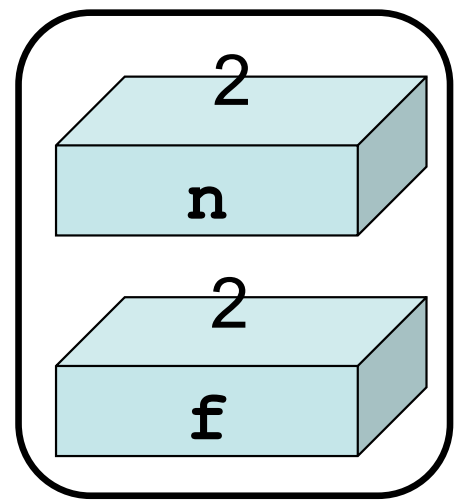
```
struct A
{
    int n;
    float f;
};
```

```
struct B
{
    A x;
    float f;
};
```

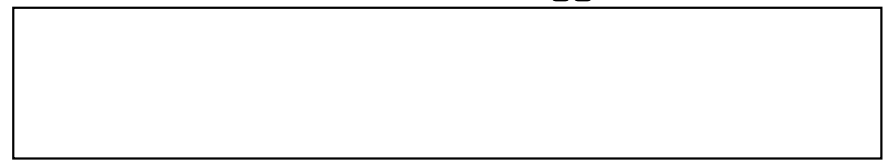
```
A f(int m, float g)
{
    A z;
    z.n = m;
    z.f = g;
    return z;
}
```

...

```
→ x = f(1, 2);
cout << x.n << " " << x.f << endl;
```



x



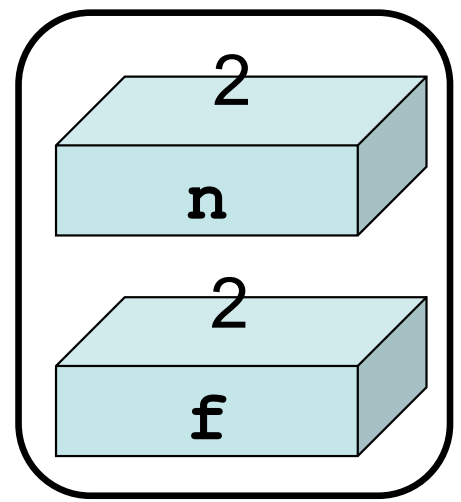
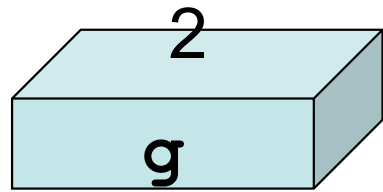
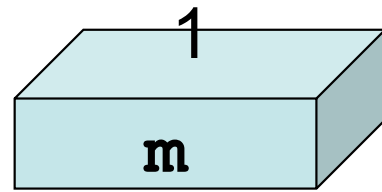
```
struct A
{
  int n;
  float f;
};
```

```
struct B
{
  A x;
  float f;
};
```

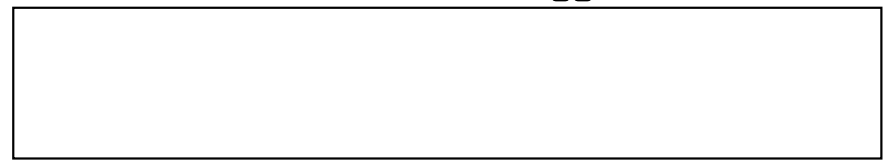
```
→ A f(int m, float g)
{
  A z;
  z.n = m;
  z.f = g;
  return z;
}
```

...

```
x = f(1, 2);
cout << x.n << " " << x.f << endl;
```



x



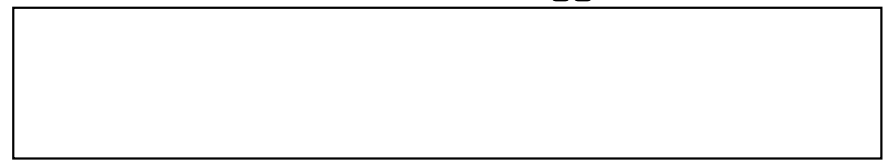
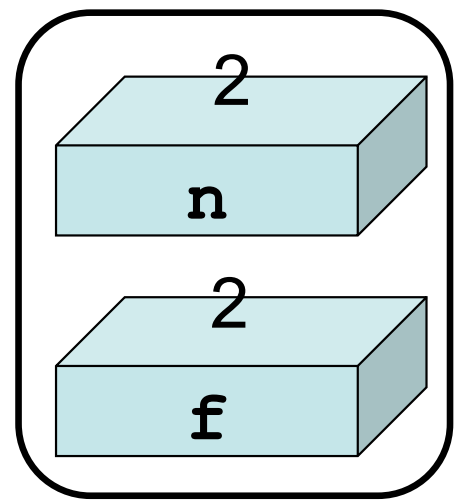
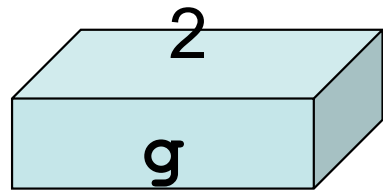
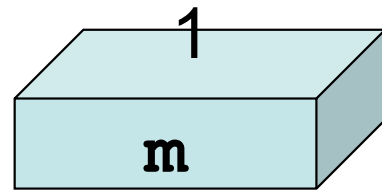
```
struct A
{
  int n;
  float f;
};
```

```
struct B
{
  A x;
  float f;
};
```

```
A f(int m, float g)
{
  A z;
  z.n = m;
  z.f = g;
  return z;
}
```

...

```
x = f(1, 2);
cout << x.n << " " << x.f << endl;
```



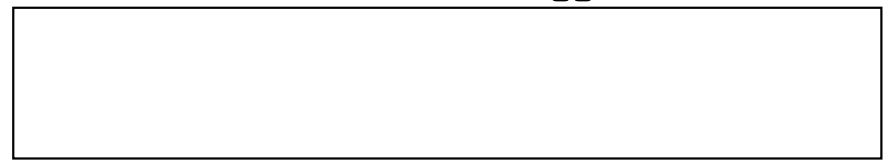
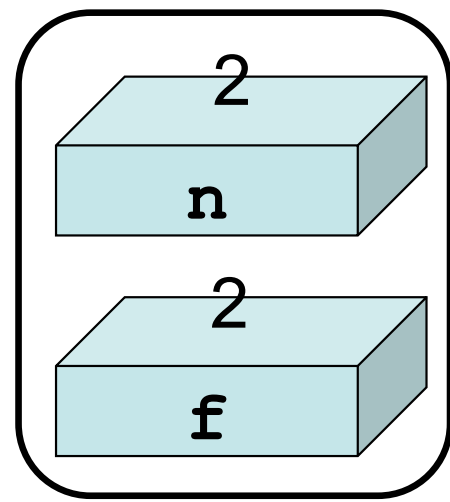
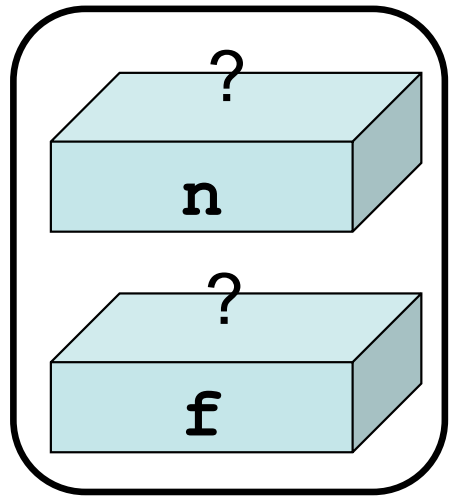
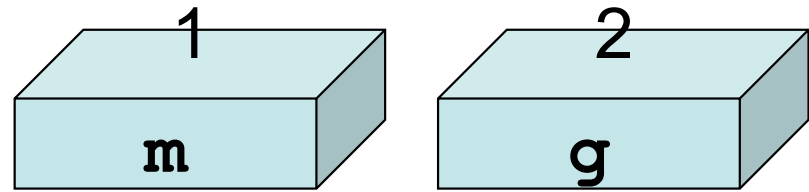
```
struct A
{
  int n;
  float f;
};
```

```
struct B
{
  A x;
  float f;
};
```

```
A f(int m, float g)
{
  A z;
  z.n = m;
  z.f = g;
  return z;
}
```

...

```
x = f(1, 2);
cout << x.n << " " << x.f << endl;
```



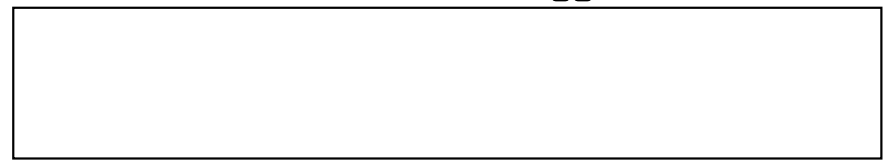
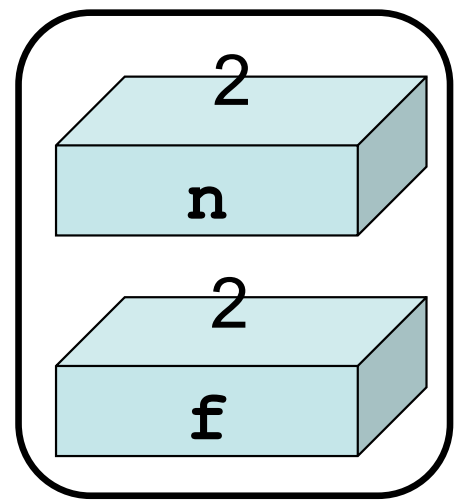
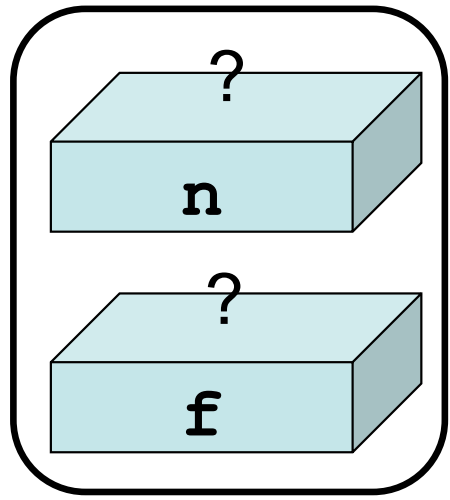
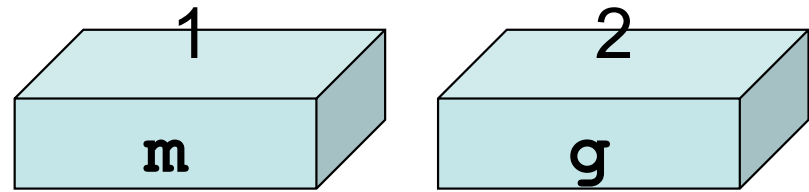
```
struct A
{
  int n;
  float f;
};
```

```
struct B
{
  A x;
  float f;
};
```

```
A f(int m, float g)
{
  A z;
  z.n = m;
  z.f = g;
  return z;
}
```

...

```
x = f(1, 2);
cout << x.n << " " << x.f << endl;
```



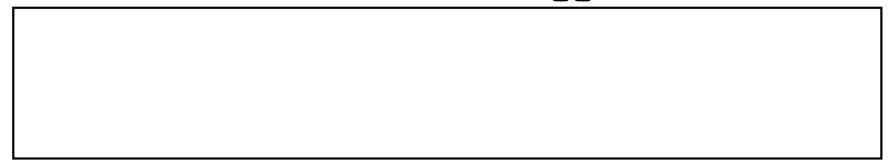
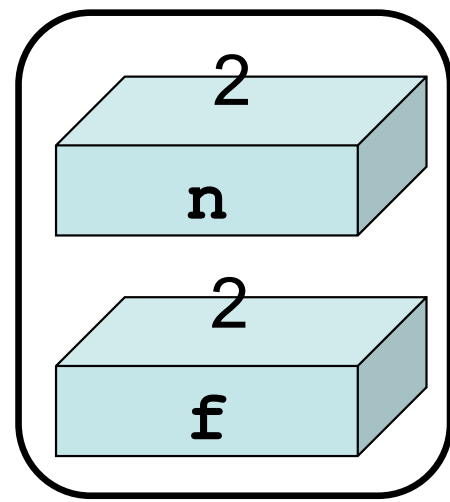
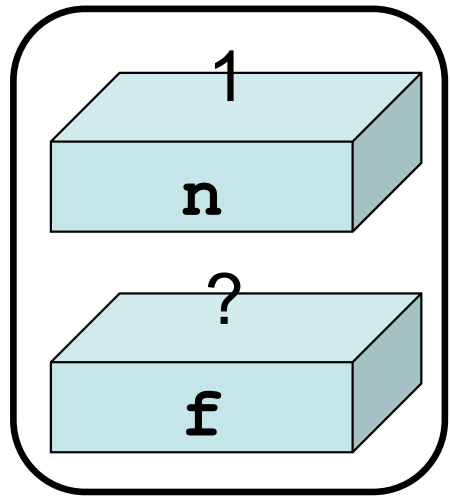
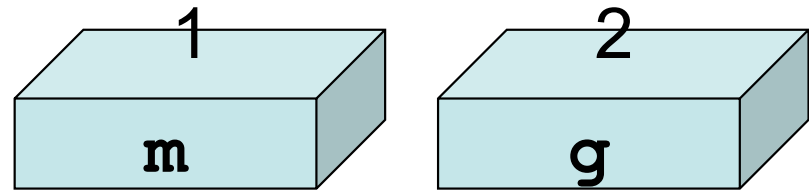
```
struct A
{
  int n;
  float f;
};
```

```
struct B
{
  A x;
  float f;
};
```

```
A f(int m, float g)
{
  A z;
  z.n = m;
  z.f = g;
  return z;
}
```

...

```
x = f(1, 2);
cout << x.n << " " << x.f << endl;
```



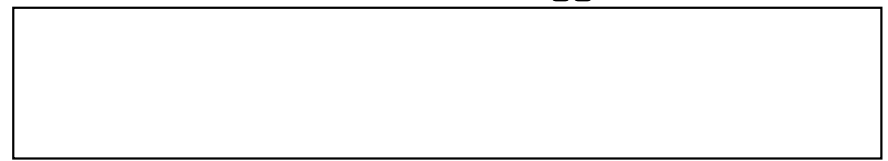
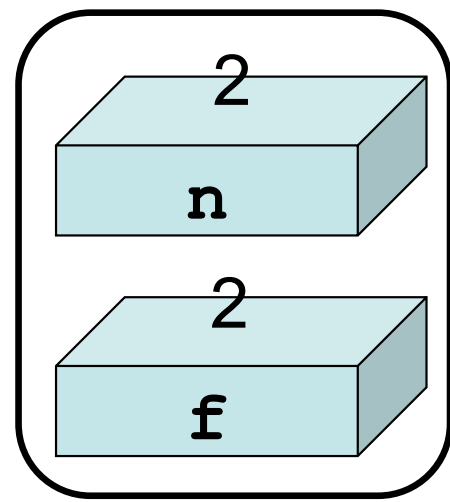
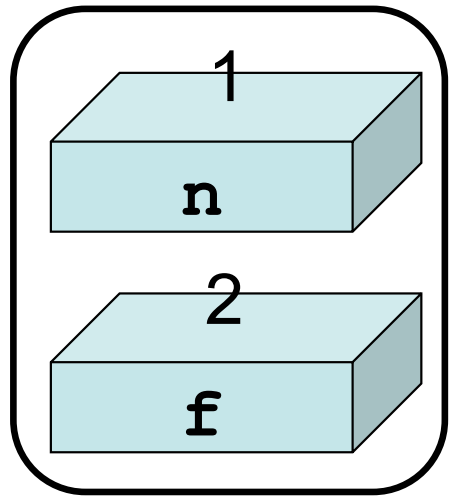
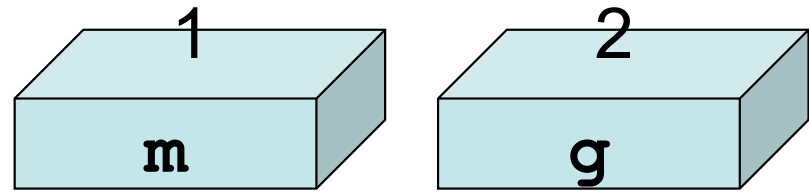
```
struct A
{
  int n;
  float f;
};
```

```
struct B
{
  A x;
  float f;
};
```

```
A f(int m, float g)
{
  A z;
  z.n = m;
  z.f = g;
  return z;
}
```

...

```
x = f(1, 2);
cout << x.n << " " << x.f << endl;
```



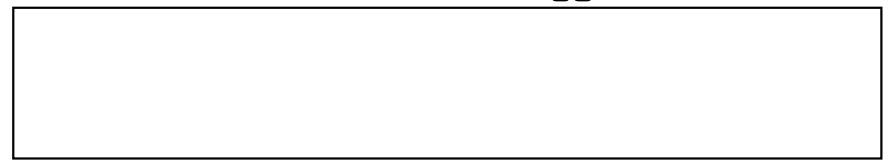
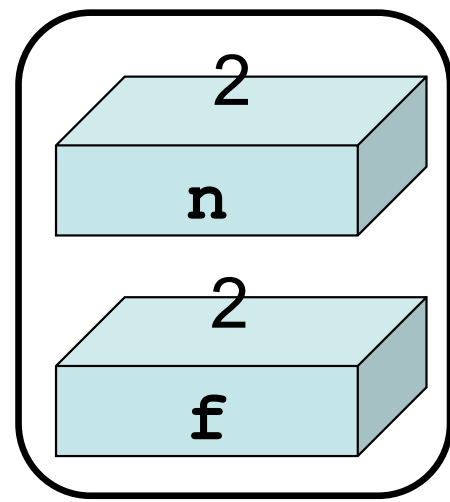
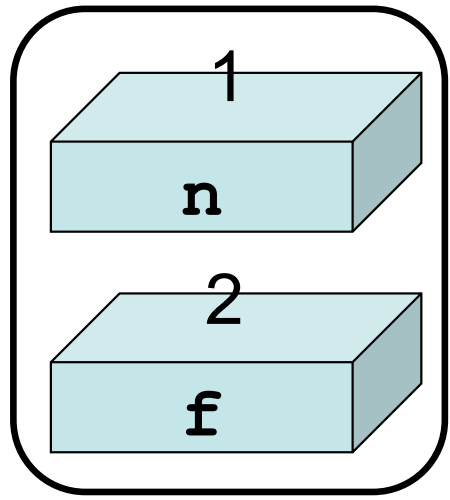
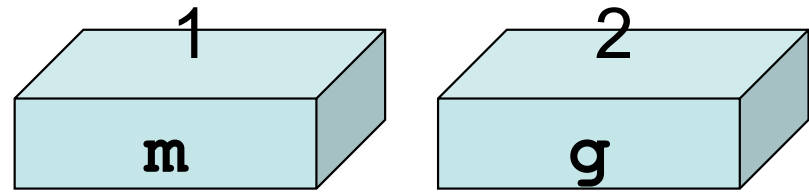
```
struct A
{
  int n;
  float f;
};
```

```
struct B
{
  A x;
  float f;
};
```

```
A f(int m, float g)
{
  A z;
  z.n = m;
  z.f = g;
  → return z;
}
```

...

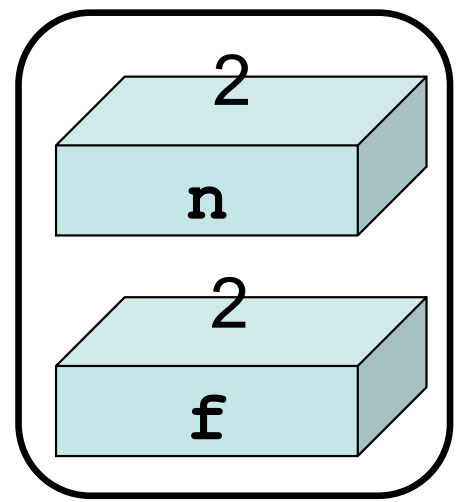
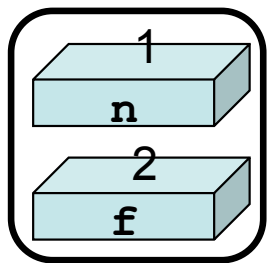
```
x = f(1, 2);
cout << x.n << " " << x.f << endl;
```



```
struct A
{
    int n;
    float f;
};
```

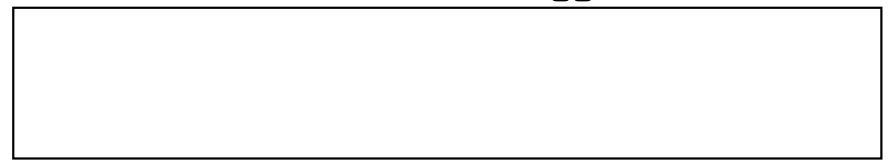
```
struct B
{
    A x;
    float f;
};
```

```
A f(int m, float g)
{
    A z;
    z.n = m;
    z.f = g;
    return z;
}
```



x

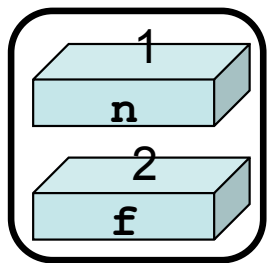
```
cout << x.n << " " << x.f << endl;
```



```
struct A
{
    int n;
    float f;
};
```

```
struct B
{
    A x;
    float f;
};
```

```
A f(int m, float g)
{
    A z;
    z.n = m;
    z.f = g;
    return z;
}
```

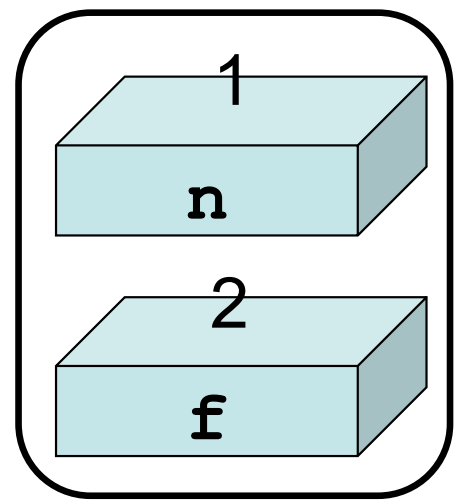


...

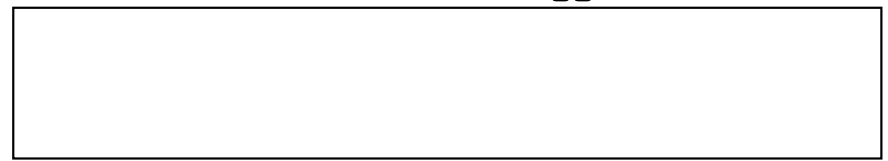


x =

```
cout << x.n << " " << x.f << endl;
```



x



```
struct A
{
    int n;
    float f;
};
```

```
struct B
{
    A x;
    float f;
};
```

```
A f(int m, float g)
{
    A z;
    z.n = m;
    z.f = g;
    return z;
}
```

...

```
x = f(1, 2);
```

```
→ cout << x.n << " " << x.f << endl;
```

