DAUPHINE UNIVERSITÉ PARIS | TUNIS

Le langage C++
Master Actuariat

Séance 3

Fatma CHAKER KHARRAT
chakerfatma@yahoo.fr

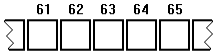
Plan

DAUPHINE UNIVERSITÉ PARIS | TUNIS

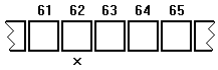
- **Les pointeurs**
- **L'allocation dynamique : New et Delete**

Les pointeurs

Une **variable** est destinée à contenir une **valeur** du type avec lequel elle est déclarée. Physiquement cette valeur se situe en mémoire.

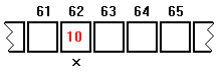


Quand on écris : **int x;**



Nous réservons une case pour la variable **x** dans la mémoire, case numéro **62** dans le cas du schéma.

Quand on écris : **x=10;**



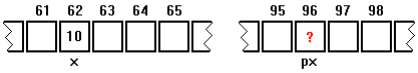
On dit que **x** a pour valeur **10**. Cette valeur est située physiquement à l'emplacement **&x** dans la mémoire (62 dans le contexte du schéma)

Les pointeurs (suite)

Un pointeur est aussi une **variable**, il est destiné à contenir une **adresse mémoire.**

```
int *px;   // Réserve un emplacement pour stocker une adresse mémoire.
px = &x;   // Ecrit l'adresse de x dans cet emplacement.
```

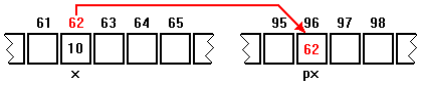
Quand on écris : **int *px;**



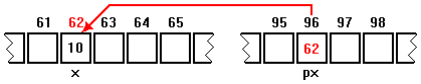
On réserve un emplacement en mémoire pour le pointeur **px** (case numéro 96 dans le cas du schéma).

Les pointeurs (suite)

Quand on écris : **px = &x;**



Quand on écris : **cout<<*px;** // Affiche la valeur de x par pointeur
// déréférencé (10 dans le cas du schéma).



*px = 20; // Maintenant x est égal à 20.

Exemple :

```
int main()
{
int A = 1;
int B = 2;
int C = 3;
int *P1, *P2;
P1=&A;
P2=&C;
*P1=(*P2)++;
P1=P2;
P2=&B;
*P1-=*P2;
++*P2;
*P1*=*P2;
A=++*P2**P1;
P1=&A;
*P2=*P1/=*P2;
return 0;
}
```

	A	B	C	P1	P2
Init	1	2	3	/	/
P1=&A	1	2	3	&A	/
P2=&C					
*P1=(*P2)++					
P1=P2					
P2=&B					
*P1-=*P2					
++*P2					
P1=*P2					
A=++*P2**P1					
P1=&A					
*P2=*P1/=*P2					

Tableaux et pointeurs

Le nom d'un tableau représente l'adresse de son premier élément :

Ainsi si on a : int T[100] alors **&T[0]** et **T** sont équivalents

Soit un tableau **T** d'un type quelconque et *i* un indice pour les composantes de **T** :

T *désigne* &T[0]
T+1 *désigne* &T[1]
T+i *désigne* &T[i]
***(T+i) *désigne* T[i]**

Manipulation des pointeurs

```
                int  tab [20]
// tab est de type int *
// tab correspond à l'adresse du début de la zone mémoire allouée pour
//contenir // 20 entiers
        (*tab)=3;      // équivalent à tab[0]=3

        tab++;         // décalage de tab d'une zone mémoire
                       // tab pointe sur la zone mémoire devant contenir le
                       //2ème élément du tableau cad &tab[1]

        (*tab)=4;      // équivalent tab[1]=4 car on a décalé tab à l'instruction
                       //précédente

        (*tab++)=5;    // équivalent à tab[2]=5;

        (*tab)++;      // équivalent à tab[2]=6;

        tab-=2;        // tab est décalée de 2 zones mémoires cad &tab[0]
```


Exemple :

Quels résultats fournira ce programme

```
#include<iostream>
using namespace std;
int main()
{
int T[3];
int i,j;
int * adt;
for (i=0,j=0; i<3; i++)  T[i]=j++ + i;                    /*1*/

for (i=0; i<3; i++)  cout<<T[i]<<" " ;                   /*2*/
cout<<" \n " ;

for (i=0; i<3; i++)  cout<<*(T+i)<<" " ;                 /*3*/
cout<<" \n " ;

for (adt=T,;adt<T+3; adt++)  cout<<*adt<<" " ;           /*4*/
cout<<" \n " ;

for (adt=T+2,;adt>=T; adt--)  cout<<*adt<<" " ;          /*5*/
cout<<" \n " ;
```

Redimensionner les tableaux ?

Repris du cours C++ de Michael Lampis – Univ. Paris Dauphine (2014/2015)

- Once an array has been declared its value is **fixed**
- What if I need more/less space?
- **Two solutions:**
  - **Vector class (to be seen later)**
  - Dynamic memory management (**new, delete**)
- Second solution is needed also to return arrays

Allocation dynamique de mémoire

Jusqu'à maintenant, on a vu que tous les objets (variables) ainsi que leur taille **devaient être déclarés au moment de la compilation**, càd explicitement dans le code C++.

En particulier, la dimension des tableaux doit être connue au moment de l'écriture du programme et ne peut être modifiée au moment de l'exécution.

Les fonctions de gestion "**dynamique**" de mémoire permettent de remédier à cette limitation.

2 fonctions de base :

New : allocation d'une zone de mémoire

delete : libération d'une zone mémoire précédemment allouée grâce à New

Les opérateurs New et delete

- 2 opérateurs : **new** et **delete**

```
int * Npt = new int;      // équivalent en C à N=(int *)malloc(sizeof(int));

int *Ntab = new int[20];  // équivalent en C à N=(int *)malloc(20 * sizeof(int));
```

new *type* : définition et allocation d'un pointeur de type ***type*.***
new *type*[*n*] : définition et allocation d'un pointeur de type ***type**** sur un tableau de *n* éléments de type *type.*

```
delete Npt;      // équivalent en C à free(Npt);

delete [ ] Ntab;    // équivalent en C à free(Ntab);
```


Allocation dynamique : New et Delete

Exemple

```
#include <iostream>
using namespace std;

int main()
{
   int i, taille;

   cout << "Tapez la valeur de taille : ";
   cin >> taille;
   int *t;
   t = new int[taille];

   for (i = 0; i < taille; i++)
      t[i] = i * i;

   for (i = 0; i < taille; i++)
     cout << t[i] << endl;
   delete[] t;

   return 0;
}
```

Exécution

```
Tapez la valeur de taille : 4
0
1
4
9
```

Exercice : Crible d'ératosthène

L'algorithme procède par élimination : il s'agit de supprimer d'une table des entiers de 2 à N tous les multiples d'un entier.
En supprimant tous les multiples, à la fin il ne restera que les entiers qui ne sont multiples d'aucun entier, et qui sont donc les nombres premiers.

On commence par rayer les multiples de 2, puis à chaque fois on raye les multiples du plus petit entier restant.
À la fin du processus, tous les entiers qui n'ont pas été rayés sont les nombres premiers inférieurs à N.

Exemple repris du cours C++de Tristan Cazenave – Univ. Paris Dauphine

Crible d'ératosthène : démo

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Prime numbers

```
#include<iostream>
using namespace std;

void erat(int Nb)
{ bool * tab;
tab=new bool[Nb];
 for (int i = 0; i < Nb; i++) {
 tab[i] = true;
 }
 tab[0] = false;
 tab[1] = false;
for (int i = 2; i < Nb; i++) {
if (tab[i]) { // n'a pas été rayé
for (int j = 2*i; j < Nb; j += i) { // on raye le reste
 tab[j] = false;
 }
 }
 }
for (int i = 0; i < Nb; i++) {
 if (tab[i])
cout<<i<<" ";
 }
 cout<<"\n";
  }

int main()
{
int N;
cout<<"Donner une valeur";
cin>>N;
erat(N);
system("pause");
}
```