Introduction à la programmation
Travaux pratiques: séance 6
INFO0201-1

G. Allemand, A.Wafflard & R. Chrétien, G. Vanhaele & B. Baert, X. Baumans
guillaume.allemand@uliege.be - adrien.wafflard@uliege.be

Programme de la séance

- Chaînes de caractères
  $\rightarrow$ Gestion de texte avec l'objet *string*
  - Code ASCII
  - Entrée-sortie de texte
  - Manipulations des chaînes de caractères et de leurs caractères individuels
  - Retour aux tableaux de **char**

Rappel : les caractères

- En C/C++, les caractères sont stockés dans des variables de type **char**. Ces variables contiennent en fait des nombres compris entre 0 et 127 qui représentent chacun un caractère différent.
- La correspondance entre un nombre et un caractère est établie dans une table appelée code **ASCII**.
- Le code correspondant à la lettre **A** est le nombre 65. On peut ainsi écrire indifféremment :

```
1 char une_lettre = 'A'; // --> le caractère A
2 une_lettre = 65; // --> aussi le caractère A
```

Pour indiquer qu'on considère un caractère comme sa valeur numérique correspondante dans le code ASCII, on entoure ledit caractère avec des **guillemets simples** : 'A', 'L' ou 'P' etc...

Rappel : les caractères - le code ASCII

Dec = Decimal; Hex = Hexadecimal; Char = Character

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Rappel : les caractères

- Affichage des caractères : lors de l'affichage d'une variable de type **char** avec cout, grâce au type, le programme sait qu'il s'agit d'un caractère et affiche le caractère correspondant au code ASCII stocké dans la variable, au lieu du code ASCII lui-même.

```
1 char une_lettre = 'A'; // place 65 dans la variable
2 cout << une_lettre << endl; // affiche A
3 une_lettre = 66; // la lettre B en code ASCII
4 cout << une_lettre << endl; // affiche B
5 cout << (int) une_lettre << endl; //affiche 66
```

Relations utiles avec les chaînes de caractères

Tous les caractères correspondant à un nombre précis, il existe des relations utiles entre certains caractères.

- Majuscule-Minuscule : les lettres A-Z et a-z se suivent dans l'ordre alphabétique dans le code ASCII. La lettre A possède le code 65, a $= 97$, B $= 66$, b $= 98$, etc...
  Il est donc possible de convertir une lettre en la majuscule ou la minuscule correspondante en soustrayant ou ajoutant $97 - 65 = 32$ à la valeur de la lettre en question.
- Valeur numérique d'un chiffre : de la même manière, les chiffres $0 - 9$ se suivent dans le code ASCII. Puisque le "0" a le code 48, il suffit de soustraire 48 à la valeur d'un chiffre codé en ASCII pour obtenir la valeur numérique correspondante.
- Caractères non imprimables : les caractères dont le code ASCII est compris entre 0 et 31 et le caractère de code 127 sont des caractères dits "non-imprimables" qu'il n'est donc pas possible d'afficher à l'écran.

Chaînes de caractères

Chaîne de caractères

Une **chaîne de caractères** est une suite de caractères contigus stockés dans un tableau de type **char** et dont le dernier élément est le caractère `null '\0'`.

Pour gérer du texte (un ensemble de caractères), on utilise des tableaux de caractères, qu'on appelle des chaînes de caractères (*string* en anglais).

Pour indiquer que le texte est terminé, on place un élément spécial (caractère `null`) dans le dernier élément de la chaîne. Cela peut-être le dernier élément du tableau, mais pas nécessairement. En effet, la chaîne de caractère peut être plus courte que le tableau dans lequel elle est stockée.
Le nombre maximal de caractères qu'on peut stocker dans une chaîne de caractères est donc égal à la taille du tableau de caractères - 1.

Chaînes de caractères standard : déclaration et initialisation

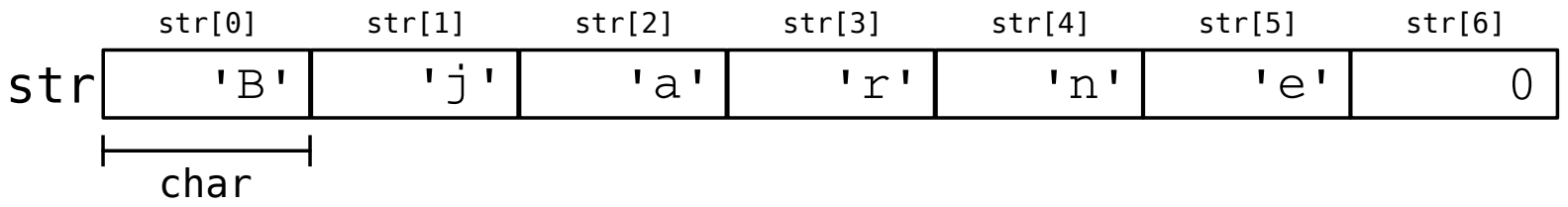
Pour déclarer et initialiser une chaîne de caractères, on déclare un tableau de type **char** et on l'initialise en écrivant la chaîne de caractères entre **guillemets doubles**.
Cela signale qu'il s'agit d'une chaîne de caractère et correspond à initialiser le tableau avec chacun des caractères de la chaîne en terminant par le caractère **null** (code ASCII = 0)

```
1 char str[] = "Bjarne";
```

est équivalent à

```
1 char str[7] = {'B', 'j', 'a', 'r', 'n', 'e', '\0'};
```

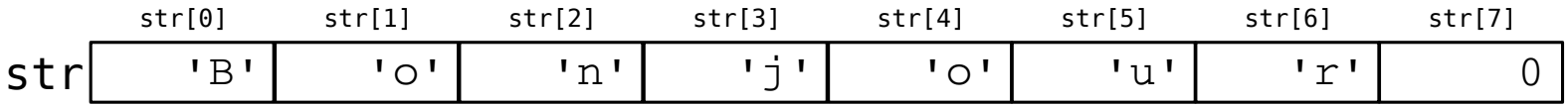
Représentation du tableau dans la mémoire :



Chaînes de caractères standard

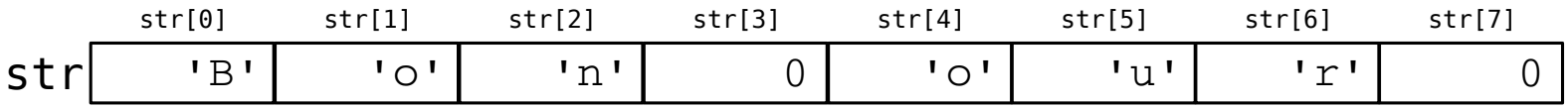
On ne peut normalement pas afficher un tableau entier de manière immédiate avec l'objet cout.
Les chaînes de caractères sont une exception : on peut afficher une chaîne de caractère en passant directement le nom du tableau qui la contient à l'objet cout :

```
1 char str[] = "Bonjour";
2 cout << str; // Affiche Bonjour
```



Le système affiche tous les caractères de la chaîne jusqu'à ce que le caractère **null** soit rencontré.

```
1 str[3] = 0;
2 cout << str; // Affiche Bon
```



L'objet *string*

- Pour gérer de manière pratique les tableaux contenants les chaînes de caractères, il existe en C++ un objet spécifique : l'objet **string**
- Les avantages principaux de cet objet sont les suivants :
  - Assistance dans la gestion du tableau contenant les caractères de la chaîne
  - Fonctions de manipulation des chaînes de caractères
  - Affectation et comparaison directe des chaînes de caractères (pas besoin de comparer tous les éléments du tableau un à un)
- Pour pouvoir l'utiliser, il faut inclure la bibliothèque "string" :

```
#include <string>
```

L'objet *string* : utilisation

- On déclare un objet *string* comme les types de base du C++ :

```
1 string message = "Ceci est du texte";
```

- On les affiche également comme les autres variables :

```
1 cout << message; // Affiche "Ceci est du texte"
```

- Chaque caractère continue à être l'élément d'un tableau :
  Si `message` est un objet de type **string**, alors `message[i]` est un caractère, de type **char**

```
1 cout << message[3] << message[10]; // Affiche "iu"
```

De plus, tous les opérateurs habituels peuvent être utilisés :

- On combine les chaînes de caractères en les additionnant
- On peut affecter une chaîne de caractères à une autre
- On peut vérifier si deux chaînes de caractères sont identiques

ATTENTION : on ne pourrait pas réaliser ces opérations directement sur des tableaux de `char` !

L'objet *string* : saisie au clavier

- L'objet cin permet lui aussi de lire directement des chaînes de caractères complètes vers des variables de type `string`.

```
1 string texte;
2 cin >> texte;
```

Le programme va copier tous les caractères saisis au clavier vers la chaîne de caractères jusqu'à ce qu'un espace soit rencontré. La taille du tableau de `char` contenant les caractères est automatiquement adaptée !

- On peut également vouloir saisir une ligne de texte entière, sans s'arrêter au premier espace rencontré. Pour cela, on utilise la fonction `getline(`*`source`*`, `*`destination`*`)` :

```
1 getline(cin, texte);
```

L'objet *string* : les fichiers

- Ecrire dans un fichier : l'écriture d'une chaîne de caractère dans un fichier fonctionne de la même manière qu'avec cout :

```
1 string texte = "Bonjour";
2 mon_fichier << texte;
```

- Lire dans un fichier : la lecture d'une chaîne de caractères fonctionne de la même manière qu'avec cin :

```
1 mon_fichier >> texte;
```

Comme avec cin, la lecture s'interrompt dès qu'un caractère 'espace' est rencontré.

- De même, on peut lire une ligne entière d'un coup avec `getline` :

```
1 getline(mon_fichier, texte);
```

Propriétés et fonctions membres de l'objet *string*

L'objet *string* contient également toute une série de fonctions permettant d'obtenir des informations ou d'effectuer des manipulations sur la chaîne de caractères qu'il contient.

Toute la documentation contenant la liste des fonctions liées à l'objet *string* ainsi que leurs paramètres et la manière de les utiliser est disponible à l'adresse
`http://www.cplusplus.com/reference/string/string/`
On trouve par exemple :

- **int** `string.size()` : Renvoie la longueur de la chaîne de caractères (indépendamment de la taille réellement allouée)
- **void** `string.clear()` : Efface le contenu de la chaîne de caractères

Propriétés et fonctions membres de l'objet *string*

- **bool** `string.empty()` : Teste si la chaîne de caractères est vide. Renvoie 0 (FAUX) si la chaîne n'est pas vide, et une valeur > 1 (VRAI) dans le cas contraire.
- **int** `string.find(`**string** `txt [, `**int** `pos = 0])` : Cherche si la chaîne de caractères contient la chaîne *txt*, à partir de la position de départ (facultative) *pos*. Si oui, la valeur retournée est la position du début de la chaîne trouvée *txt*, sinon il s'agit de la valeur -1.
- **string** `string.substr(` **int** `pos,` **int** `length)` : Renvoie une nouvelle chaîne de caractères constituée de *length* caractères depuis la position *pos* dans la chaîne initiale.
- **string** `string.insert(` **int** `pos,` **string** `txt)` : Renvoie une nouvelle chaîne dans laquelle la chaîne *txt* est insérée à la position *pos* de la chaîne initiale.

Comparaison de deux *string*

- On peut vérifier l'égalité de deux chaînes de caractères :

```
1 if(texte1 == texte2)
```

- On peut vérifier qu'elles sont différentes :

```
1 if(texte1 != texte2)
```

- On peut aussi vérifier laquelle précède l'autre suivant la table ASCII (par exemple, pour les trier) :

```
1 string t1, t2;
2 if(t1 < t2)
```

- On peut convertir un nombre (int, double, ...) en un string grâce à la fonction to_string()

```
1 string t1;
2 t1 = to_string(18);
```

Tableaux de *string*

L'objet *string* peut également être exprimé sous forme de tableau, dont chaque case contiendra une chaine de caractère. Pour cela, on le déclare avec des crochets []

```
1 string tableau [2];
2 tableau[0]="Hello ";
3 tableau[1]="world!";
4 cout<<tableau[0]<<tableau[1]<<endl; \\ Affiche "Hello
      world!"
```

On peut ainsi stocker les éléments d'un fichier dans un tableau de string, et appeler chaque élément de la même manière qu'un tableau :

```
1 mon_fichier >> tableau [i];
```

Exemples d'utilisation de *string*

```
 1  int main()
 2  {
 3    string txt = "To be or to be";
 4    string txt2 = txt + ", that is the question";
 5    if(txt2.find("be", 15))
 6      cout << "to be !" << endl;
 7    if(txt2.find("not") == -1)
 8    {
 9      cout << "not to be !" << endl;
10      txt2 = txt2.insert(9, "not ");
11      cout << txt2 << endl;
12    }
13    txt2 = txt2.substr(32, 8); // "question"
14    return 0;
15  }
```

Retour aux tableaux de caractères

A partir d'un objet *string*, on peut revenir à un simple tableau de caractères avec `.c_str()`. Ceci permet d'utiliser les chaînes de caractères avec certaines fonctions qui nécessitent des tableaux de `char`, et pas des `string`.

```
1 string texte = "Hello";
2 fonction_char( texte.c_str() );
```

ATTENTION : Avec **tous** les tableaux, donc les chaînes de caractères renvoyées par `c_str()` aussi, il est impossible de copier directement un tableau dans l'autre :

```
1 char buffer[256] = texte.c_str(); // NON !!!
```

Il n'est pas permis de copier un tableau dans un autre en utilisant l'opérateur d'affectation (explication à la prochaine séance).
Il faudrait donc réaliser les opérations "à la main", caractère par caractère, ou utiliser une fonction spécifique.

Exercices (1/4)

1. Manipulations simples d'une chaîne de caractères
   - Déclarer un objet de type `string` et l'initialiser en demandant à l'utilisateur de choisir un message au clavier ;
   - Afficher le message à l'écran ainsi que sa longueur ;
   - Modifier le message pour que la première lettre soit une majuscule si elle ne l'est pas ;
   - Ajouter un point "." à la fin du message s'il n'y en a pas ;
   - Afficher le message ainsi modifié à l'écran ;
   - Demander à l'utilisateur un mot à chercher ;
   - Chercher si le mot apparaît dans le message, et si c'est le cas, afficher à l'écran combien de fois il y apparaît et à quelle(s) position(s).
2. Manipulations de chaînes de caractères et fichiers
   - Ouvrir le fichier "users.txt" en lecture et charger les 10 noms et prénoms qui y figurent dans un tableau de `string` ;
   - Ecrire une fonction `string formattage(string nom, string prenom)` qui renvoie une chaine de caractères contenant la 1re lettre du prénom, un point "." suivi d'un espace et le nom ("P. Nom") ;
   - En utilisant cette fonction, Afficher la liste des noms et prénoms à l'écran sous la forme "P. Nom", où "P" est la 1re lettre du prénom.

Exercices (2/4)

3. Manipulations de chaînes de caractères et fichiers
   - Ouvrir le fichier “presence.txt” en lecture et charger les 200 noms et prénoms qui y figurent dans un tableau de `string` ;
   - Trier le tableau dans l'ordre alphabétiquement croissant des noms ;

     Pour ce faire
     - Parcourir le tableau, comparer les noms successifs deux à deux et les échanger s'ils ne sont pas dans le bon ordre ;
     - Répéter le parcours du tableau entier tant que des éléments ont dû être échangés quelque part lors de l'itération précédente ;
   - Enregistrer le tableau trié dans un fichier “presence_trie.txt” ;
   - Demander à l'utilisateur de saisir un nom à rechercher au clavier ;
   - Chercher dans le tableau si le nom y est présent et afficher le résultat à l'écran ;
   - Dans un fichier “verification_presence.txt”, ajouter le nom et prénom recherchés ainsi que le résultat sous la forme “Nom Prénom est présent/absent”, selon le cas ;
   - Demander un nouveau nom à vérifier à l'utilisateur tant que le mot “quitter” n'est pas saisi au clavier.

Exercices (3/4)

4. Code de César - cryptage de texte.
   - Ecrire une fonction de cryptage qui prend en paramètre une chaîne de caractère et une clé (=un nombre entier). La fonction va renvoyer le message crypté en décalant tous les caractères de la chaîne d'une valeur égale à la clé (exemple : la clé = 1, alors A $\rightarrow$ B, B $\rightarrow$ C, etc.)
     ATTENTION : veiller à ce que le décalage ait pour résultat un caractère dont la valeur est comprise entre 32 et 126 inclus (caractères imprimables).
   - Charger la ligne contenu dans le fichier message.txt disponible à l'adresse `http://www.pqs.ulg.ac.be/`, placer celle-ci dans une chaîne de caractères et afficher le texte à l'écran ;
   - Demander un mot de passe au clavier. La clé à utiliser pour crypter le texte sera obtenue en sommant la valeur de tous les caractères du mot de passe. Crypter ensuite le texte avec cette clé ;
   - Afficher le texte crypté à l'écran et l'enregistrer dans un fichier appelé message_code.txt.

Exercices (4/4)

5. Décryptage d'un code de César.
   - Écrire une fonction qui permet de décrypter une chaîne de caractères à partir de la clé du code de César correspondant (attention à l'intervalle $[32, 126]$ des caractères imprimables) ;
   - Charger le fichier crypté à l'exercice 4 et essayer de trouver la clé permettant de le décrypter. Pour ce faire, on supposera que la lettre la plus probable dans un texte écrit en français est la lettre 'e'.
   - Décrypter le message à l'aide de la clé trouvée, afficher le message décodé et l'enregistrer dans un fichier appelé message_decode.txt.