Chapter 3

Variables

3.1. Introduction

In programming, variables play an important role because they allow the user to store, modify, or retrieve values throughout the program when it is running.

Variables are then used to store information that can change, depending on conditions or on information passed to the program. They are also provide a way of labeling data with a descriptive name, so our programs can be understood more clearly by the reader and ourselves. Their sole purpose is to label and store data in memory.

Rules on naming variables

The name of a variable is called an **identifier**. A C++ identifier:

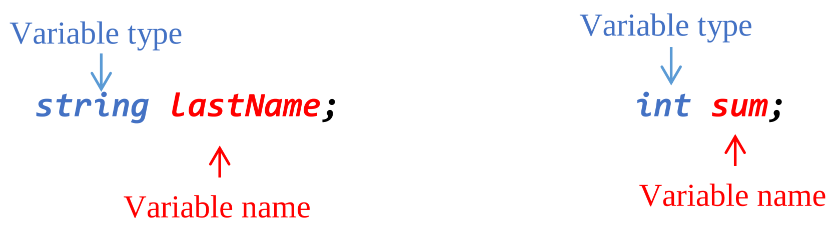
- Must start with either a letter or the underscore symbol, and all the rest of the characters must be letters, digits, or the underscore symbol.
- Cannot start with a number
- Cannot have space
- Only uses one character, the underscore, others symbols are forbidden.

When naming a variable, it is recommendated to name it according to the stored value or the use of the variable. For example, if you need to create a variable to store the sum of two numbers, you might want to call the variable ***sum***. Or if you need a variable to store a last name, you might the variable ***lastName***.

Variables declaration

Every variable in a C++ program must be declared before it is used. Declaring variable is basically telling the program the type of value that a variable will store.

The syntax for variable declarations is as follow:



```
string lastName;
int sum;
```

The following table shows the variable types:

Type Name	Memory Used	Precision	Variable Value type
`string`	4 bytes	Not applicable	"string " → double quotation mark
`char`	1 byte	Not applicable	'k' → single quotation mark
`int`	4 bytes	Not applicable	Negative and positive whole number. 10 digits

`short`
`short int`
	2 bytes	Not applicable	Negative and positive whole number. 5 digits

`long`
`long int`
	4 bytes	Not applicable	Negative and positive whole number. 10 digits

`float`	4 bytes	7 digits	Negative and positive whole and decimal number. 6 digits
`double`	8 bytes	15 digits	Negative and positive whole and decimal number. 6 digits
`long double`	10 bytes	19 digits	Negative and positive whole and decimal number. 6 digits
`bool`	1 byte	Not applicable	true or false

Variables of the same type can be written in the same line separated by a comma:

```
int number;
int sum;
```

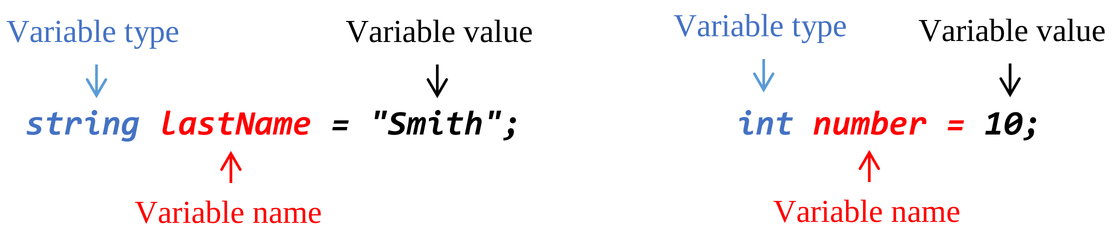
Variable **`number`** and **`sum`** can be written in one line as:

```
int number, sum;
```

<u>Initializing variables in declaration</u>

You can initialize a variable, which is giving a value, at the time that you declare the variable

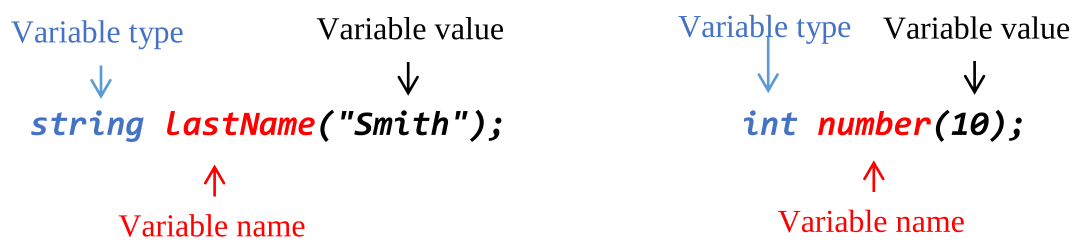
Syntax:



```
string LastName = "Smith";          int number = 10;
```

An alternative syntax for initializing in declarations is my enclosing the initial value in parenthesis next to the variable name.

Syntax:



```
string lastName("Smith");
```

```
int number(10);
```

3.2. `char` variable

`char` variable type is an integral type that stores characters as an integer value from the ASCII characters.

ASCII stands for American Standard Code for Information Interchange, and it defines a particular way to represent English characters (plus a few other symbols) as numbers between 0 and 127 (called an **ASCII code** or **code point**). For example, ASCII code 97 is interpreted as the character ‘a’.

Character literals are always placed between single quotes.

Here’s a full table of ASCII characters:

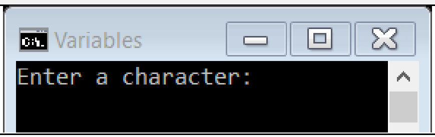
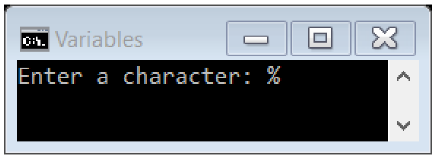
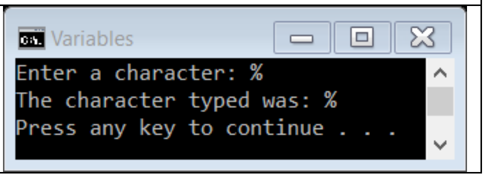
Code	Symbol (character)	Code	Symbol (character)	Code	Symbol (character)	Code	Symbol (character)
0	NUL (null)	32	(space)	64	@	96	`
1	SOH (start of header)	33	!	65	A	97	a
2	STX (start of text)	34	”	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	’	71	G	103	g
8	BS (backspace)	40	(	72	H	104	h

9	HT (horizontal tab)	41	)	73	I	105	i
10	LF (line feed/new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (form feed / new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (data control 1)	49	1	81	Q	113	q
18	DC2 (data control 2)	50	2	82	R	114	r
19	DC3 (data control 3)	51	3	83	S	115	s
20	DC4 (data control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of transmission block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[	123	{
28	FS (file separator)	60	<	92	\	124	|
29	GS (group separator)	61	=	93	]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL (delete)

Codes 0-31 are called the unprintable chars, and they're mostly used to do formatting and control printers. Most of these are obsolete now.

Codes 32-127 are called the printable characters, and they represent the letters, number characters, and punctuation that most computers use to display basic English text.

For example, write a C++ program to request a user to type a character and then display the typed character:

Algorithm	C++ code	Display - Output
Print a message requesting user to type a character	`cout<<"Enter a character: ";`	 Enter a character:
Create a **`char`** variable to save the character typed by the user.	`char typeChar;`	
Collect information from keyboard and save the information in variable **`typeChar`**	`cin>>typeChar;`	User types **%**  Enter a character: %
Print a message that will display the typed character	`cout<<"The character typed was: "<<enterChar1;`	 Enter a character: % The character typed was: % Press any key to continue . . .

3.3. `string` variable

String variable is used to store a set of one or more character known as sequence of text.

String can be assigned using string variable as:

Syntax:

Variable type — Variable value

↓ ↓

`string lastName = "Smith";`

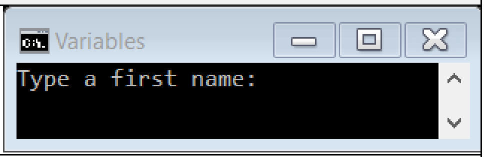
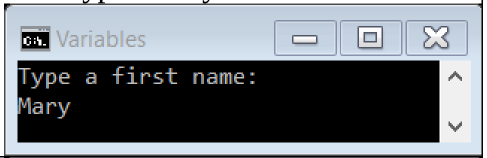
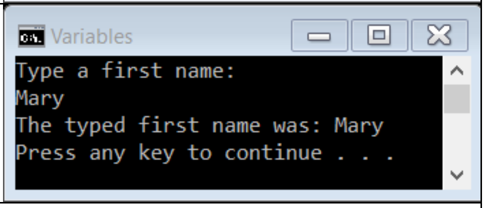
↑

Variable name

We can demonstrate that the variable ***`LastName`*** contains the value ***`Smith`*** by outputting the contents of the variable by typing the variable name ***`LastName`*** as:

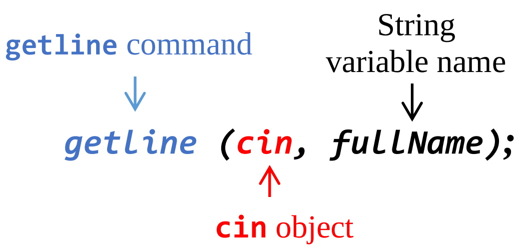
```
cout<<lastName<<endl;
```

You can also request a user to type a name and then display the name using the following algorithm:

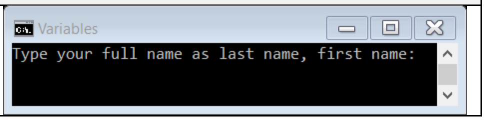
Algorithm	C++ code	Display - Output
Print a message requesting the user to type a first name	`cout<<"Type a first name: "<<endl;`	 `Type a first name:`
Create a string variable to save the first name typed by the user.	`string firstName;`	
Collect information from keyboard and save the information in variable **`firstName`**	`cin>>firstName;`	User types Mary  `Type a first name:` `Mary`
Print a message that will display the first name entered by the user	`cout<<"The typed first name was: " <<firstName;`	 `Type a first name:` `Mary` `The typed first name was: Mary` `Press any key to continue . . .`
End program		If the program ran sucessfully, the end program line should display: **`Press any key to continue…`**

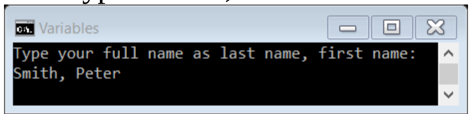
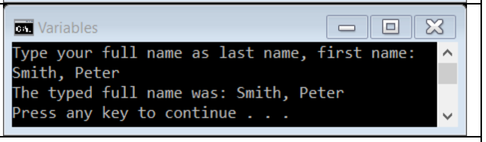
***`getline`* command**

The **`getline`** command is used to input a phrase containing space into a string variable. The syntax for `getline` is:



For example, to write a program that will ask the user to enter a full name:

Algorithm	C++ code	Display - Output
Print a message requesting user to type a full name	`cout<<"Type your full name as last name, first name: "<<endl;`	 `Type your full name as last name, first name:`

Create a string variable to save the full name typed by the user.	`string fullName;`	
Collect information from keyboard and store the information in variable **`fullName`**	`getline(cin, fullName);`	User types Smith, Peter  `Type your full name as last name, first name:` `Smith, Peter`
Print a message that will display the full name entered by the user	`cout<<"The typed full name was: " <<fullName;`	 `Type your full name as last name, first name:` `Smith, Peter` `The typed full name was: Smith, Peter` `Press any key to continue . . .`
End program		If the program ran sucessfully, the end program line should display: **`Press any key to continue…`**

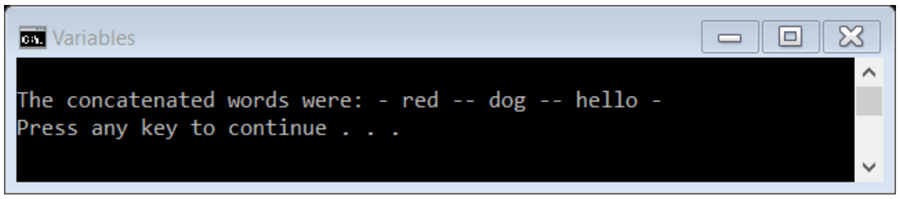
Concatenated or add strings

Strings can be **added** or **concatenated** using the plus operators +

For example, we can create a program to concatenate three words:

Algorithm	C++ code
Create three strings with initial value.	`string animal="- dog -",color="- red -",phrase="- hello -";`
Concatenate the three strings in variable **`allWords`**	`string allWords = color + animal + phrase;`
Print a message that will display the concatenated string	`cout<<"The concatenated words were: "<<allWords;`

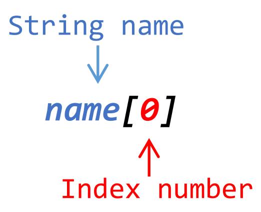
Display – Output



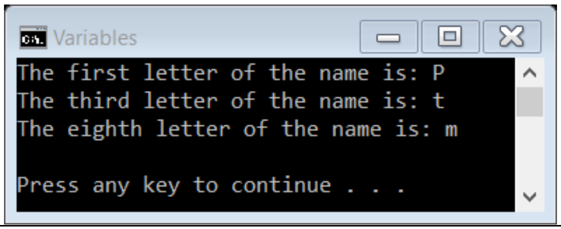
```
The concatenated words were: - red -- dog -- hello -
Press any key to continue . . .
```

String index

Each character in a string be read using an index number. An index indicates the character location starting from 0 for the first character in the left. The index must be enclosed in a squared brackets **[ ]**



For example, write a program to print the first, third, and eighth character of the name *Peter Smith*:

Algorithm	C++ code
Create string **`name`** with value *Peter Smith*	**`string name="Peter Smith";`**
Print the first letter of variable **`name`**	**`cout<<"The first letter of the name is: "<<name[0]<<endl;`**
Print the third letter of variable **`name`**	**`cout<<"The third letter of the name is: "<<name[2]<<endl;`**
Print the eighth letter of variable **`name`**	**`cout<<"The eighth letter of the name is: "<<name[7]<<endl;`**
Display – Output 	

```
The first letter of the name is: P
The third letter of the name is: t
The eighth letter of the name is: m

Press any key to continue . . .
```

String function

Although C++ lacks a simple data type to directly manipulate strings, there is a **`string function`** that may be used to process strings in a manner similar to the data type.

A function is a group of related commands which can be executed on demand. For example, **main** is the primary function of a C++ program because it executes the C++ program when we run the **.exe** file.

There are a variable of functions available for string objects. To insure compatibility across multiple compilers the **string library** should be included at the top of the program to support these functions:

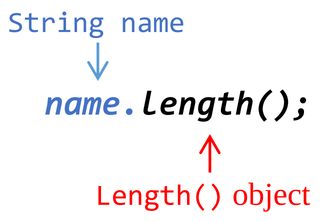
```
#include<string>
```

After it, in the **`main()`** function, you can declare variables of type **`string`** just as you declare variables of other types such **`int`** or **`char`**. Also the string value must encloses in quotation mask.

The string function takes each letter of the word and position them into an array as a character. Therefore, C++ uses class string to work with strings. One of the things we can do with string function are to find the length of a string, and extract, insert, erase, or find character/s in a string.

Length of a string

The length of the string can be determined with the **`length( )`** object. The syntax of **`length( )`** object is:

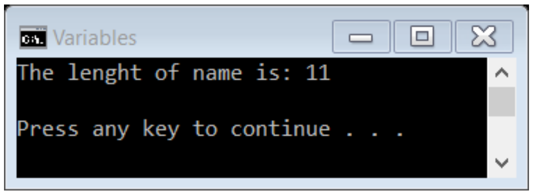


```
name.length();
```

For example, to display the length of the name Peter Smith:

```
string name="Peter Smith";
cout<<"The length of name is: "<<name.length()<<endl;
```

Display output

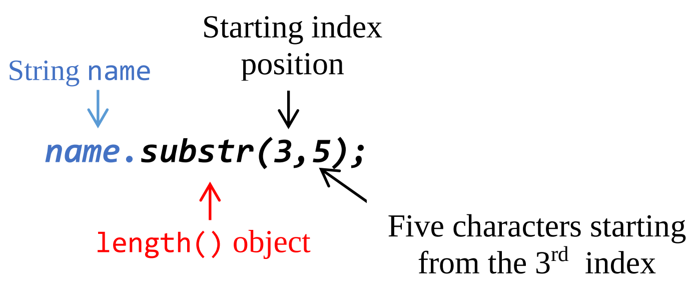


```
The lenght of name is: 11

Press any key to continue . . .
```

Subtracting characters in a string

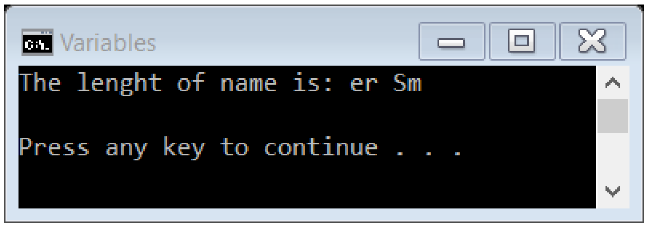
A subset of a string can be extracted using the **`substr( )`** object. This function has two input parameters, the starting index and the number of characters to include (from the start index on). It returns the new subtracting based upon the input specifications. The syntax for **`substr()`**



For example, if want to print five characters starting from the 3rd index, which the four character:

```
string name="Peter Smith";
cout<<"The lenght of name is: "<<name.substr(3,5)<<endl;
```

Display output

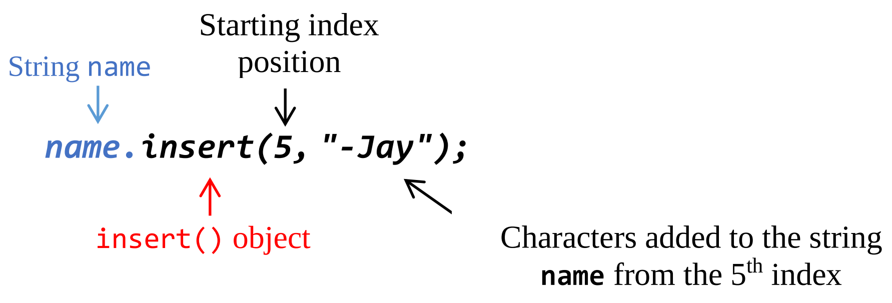


```
The lenght of name is: er Sm

Press any key to continue . . .
```

Inserting characters into a string

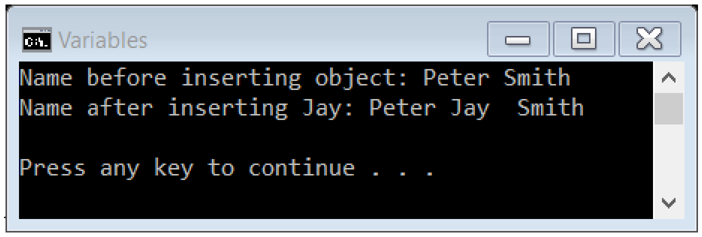
Text can be inserted into a string using the **`insert()`** object. This function accepts two input parameters, the starting index and the string to be inserted at that location. The syntax for **`insert()`** object is:



For example, add the characters –**Jay** to name *Peter Smith:*

```
string name="Peter Smith";
string newName = name.insert(5, "-Jay");
cout<<"Name before inserting object: "<<name<<endl;
cout<<"Name after inserting Jay: "<<newName<<endl;
```

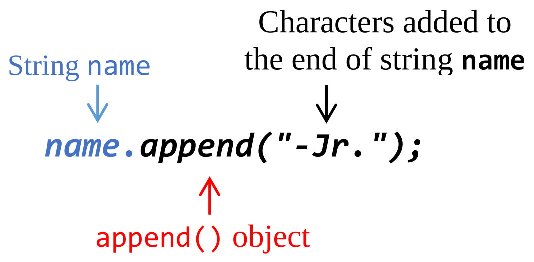
Display output



```
Name before inserting object: Peter Smith
Name after inserting Jay: Peter Jay  Smith

Press any key to continue . . .
```

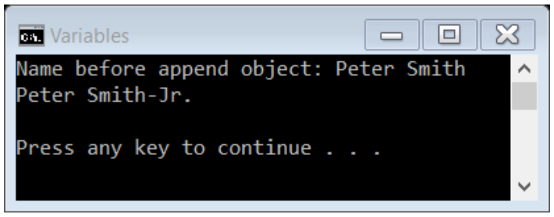
We can add characters to the end of a string using the **`append()`** object. This object accepts one input parameter, which is the characters to append.



For example, add the character ***`-Jr.`*** to the name *Peter Smith*

```
string name="Peter Smith";
cout<<"Name before append object: "<<name<<endl;
string newName = name.append("-Jr.");
cout<<newName<<endl;
```

Display output

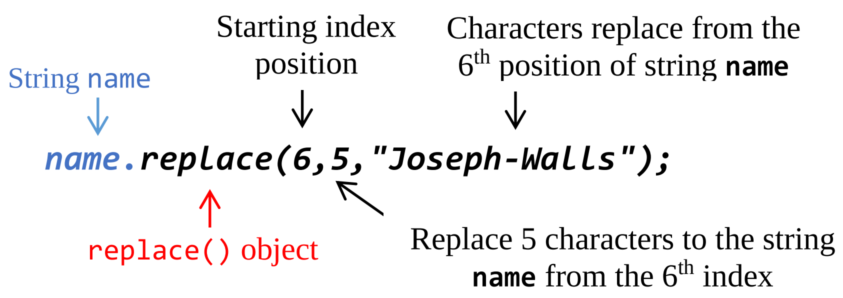


```
Name before append object: Peter Smith
Peter Smith-Jr.

Press any key to continue . . .
```

Replace characters in a string

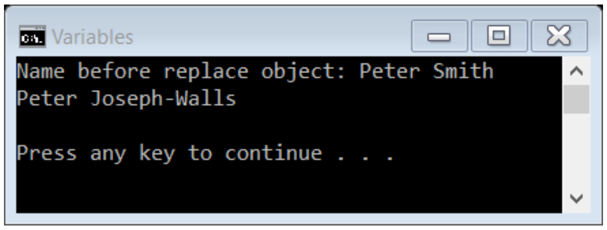
Characters can be replace in a string using the **replace()** object. This object accepts three parameters, the starting index, the number of characters to replace from the starting index, and the new string that will replace the specific text.


Starting index position
Characters replace from the 6th position of string name
String name
name.replace(6,5,"Joseph-Walls");
replace() object
Replace 5 characters to the string name from the 6th index

For example, replace the last name *Smith* from the name *Peter Smith* with the characters *Joseph-Walls:*

```
string name="Peter Smith";
cout<<"Name before replace object: "<<name<<endl;
string newName = name.replace(6,5,"Joseph-Walls");
cout<<newName<<endl;
```

Display output

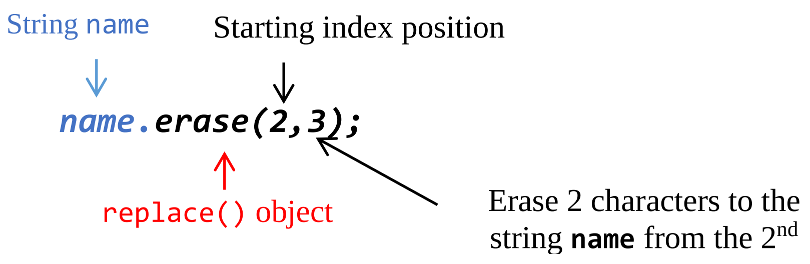


```
Name before replace object: Peter Smith
Peter Joseph-Walls

Press any key to continue . . .
```

Erase characters from a string

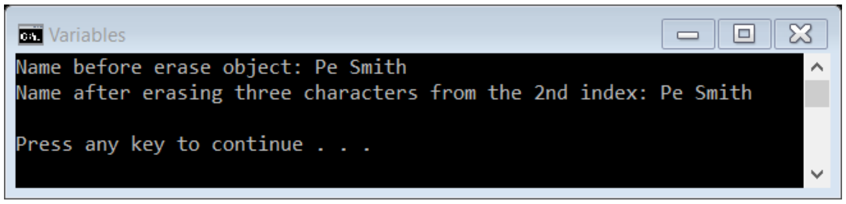
Characters can be erased from a string using **`erase()`** object. This object accepts two parameters: the start index and the number of characters to erase from the string.



For example, erase three characters after the 2nd index in name *Peter Smith*

```
string name="Peter Smith";
string newName = name.erase(2,3);
cout<<"Name before erase object: "<<name<<endl;
cout<<"Name after erasing three characters from the 2nd index: "<<newName
   << endl;
```

Display output



```
Name before erase object: Pe Smith
Name after erasing three characters from the 2nd index: Pe Smith

Press any key to continue . . .
```

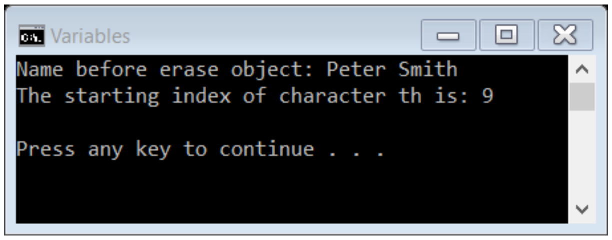
Finding characters in a string

Characters can be found using the **`find()`** object. This object accepts one parameter: the characters to search from a string. In return, the program returns the starting index of the searched characters.

For example, find the index of characters *th* in name *Peter Smith*

```
string name="Peter Smith";
cout<<"Name before erase object: "<<name<<endl;
cout<<"The starting index of character th is: "<<name.find("th")<<endl;
```

Display output



```
Name before erase object: Peter Smith
The starting index of character th is: 9

Press any key to continue . . .
```