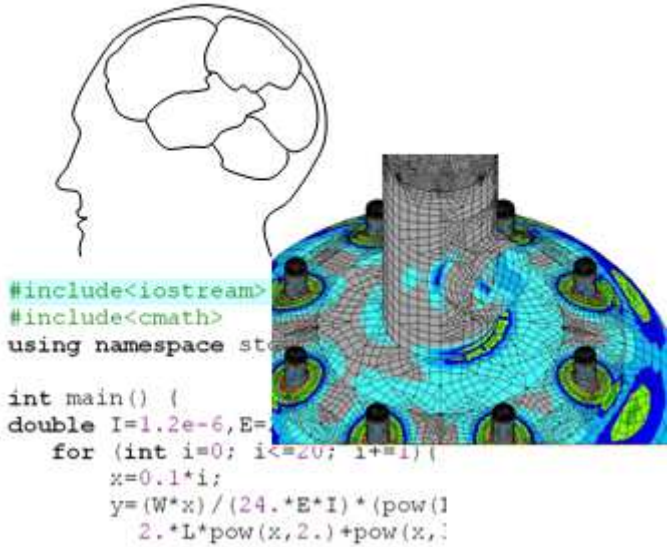
ME 240
Computation for Mechanical Engineering



Lecture 4

C++ Data Types

Introduction

In this lecture we will learn some fundamental elements of C++:

Introduction
Data Types
Identifiers
Variables
Constants

NOTE THAT

The C and C++ programming languages are quite different from each other, even though they share some common syntax.

Data Types

A data type determines the type of the data that will be stored, in the computer memory (RAM).

***C++ provides 6 fundamental data types*:**

```
char
int
float
double
bool
wchar_t
```

There are also some <u>qualifiers</u> that can be put in front of the <u>numerical</u> data types to form derivatives:

`short, long, signed, unsigned`

For example:

```
short int
unsigned char
```

The table below shows the fundamental data types in C++, as well as the range of values.

Integer types in C++	Approximate Range
short	-32767 ... 32767
unsigned short	0 ... 65535
int	**-2 147 483 647 ... 2 147 483 647**
unsigned	0 ... 4 294 967 295
long	-2 147 483 647 ... 2 147 483 647
unsigned long	0 ... 4 294 967 295

Floating point types in C++	Approximate Range	Significant Digits
float	10^{-37} ... 10^{+38}	6
double	**10^{-307} ... 10^{+308}**	**15**
long double	10^{-4931} ... 10^{+4932}	19

The red coloured int and double data types are all you needin basic level programming.

Notes:

1. The unqualified **`char`**, **`short`**, **`int`**, (**`long int`**) are **`signed`** by default.

2. You don't need to write **`int`** after using **`short`** and **`long`** keywords. i.e.

`short s;`	means	**`short int s;`**
`long k;`	means	**`long int k;`**

Identifiers

An identifier is a string of alphanumeric characters. It is used for naming variables, constants, functions, structures and classes.

A valid identifier

- must begin with a letter or underscore (_),
- can consist only of letters (**a-z**, **A-Z**), digits(**0-9**), and underscores.
- should not match with any C++ reserved keywords which are:

```
asm, auto, bool, break, case, catch, char, class, const,
const_cast, continue, default, delete, do, double,
dynamic_cast, else, enum, explicit, export, extern, false,
float, for, friend, goto, if, inline, int, long, mutable,
namespace, new, operator, private, protected, public,
register, reinterpret_cast, return, short, signed, sizeof,
static, static_cast, struct, switch, template, this, throw,
true, try, typedef, typeid, typename, union, unsigned,
using, virtual, void, volatile, wchar_t, while
```

According to these rules, the following are <u>valid</u> identifiers:

```
mass
peynir
pos12
speed_of_light
SpeedOfLight
isPrime
```

while the following are <u>not valid</u>:

```
2ndBit
speed of light
yağmur
c++
float
```

Remember to use only the <u>English alphabet</u>:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

Note that C++ is <u>case sensitive</u>.

That is, it distinguishes between uppercase and lowercase letters.

So, `Food` and `food` are different identifiers.

Variables

- A variable is a symbolic name given to a value and is associated with a data storage location in the computer's memory.
- A variable's name has to be a valid identifier.
- A variable must be declared before it is used.
- When declaring a variable, you must tell the compiler what ***kind*** of variable it is; **`int`**, **`char`**, **`double`**, …
- A variable has a limited scope within a program section where it is visible and accessible.

- Example declarations

```
int i, j;
long k;
float w, x, y, z;
double speed, dragForce;
```

- When a variable is declared, you can *initialize* it in two alternative but equivalent ways

```
int cake = 122;
```

or

```
int cake(122);
```

Example Program: *Declaration and manipulation of variables*

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

int main () {

  short x = 22, y = 11, z;
  z = x - y;
  cout << "z = " << z << endl;

  int p = 3;
  int q = x*y*z - 2*p;
  cout << "q = " << q << endl;

  return 0;

}
```

Output:

```
z = 11
q = 2656
```

Example Program: The **scope** of variables

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

int k = 11;          this k is global (visible throughout the whole program
                     including in any other functions defined in the program)
int main ()
{

  int k = 22;        this k is local in the main() function block
  {
    int k = 33;      this k is local inside this block
    cout << "Inside internal block: k = " << k << endl;
  }
  cout << "Inside main(): k = " << k << endl;
  cout << "Global k = " << ::k << endl;

}
```

Output

```
Inside internal block: k = 33
Inside main(): k = 22
Global k = 11
```

Constants

- To help promote safety, variables can be made *constant* with the **`const`** qualifier. Since **`const`** variables cannot be assigned during execution, they must be initialized at the point of declaration.

```
const double pi = 3.14159265358979;
```

Here, **`pi`** is a type double variable storing the value 3.14159265358979 (remember type double variables store floating values to about 15 digit precision). The **`const`** qualifier tells the compiler to not allow us to change the value of the variable during execution of the program.

- *Symbolic* constants are defined via the **`#define`** preprocessor directive.

```
#define pi 3.14159265358979
```

Here the preprocessor (before compilation) replaces any occurrences of **`pi`** with the literal constant **`3.14159265358979`**. But this can be dangerous!

Representation of Integer and Floating Point Numbers

- Integer literal constants can be represented by three different bases: base-10 (decimal), base-8 (octal) and base-16 (hexadecimal)

`i = 75;`	base-10 (default)
`i = 0113;`	base-8 representation of decimal 75
`i = 0x4B;`	base-16 representation of decimal 75
`i = 0x4b;`	is also base-16

- Floating point literals express numbers with decimals and/or exponents. The symbol **E** or **e** is used in the exponent.

`x = 123.456;`	decimal floating-point number
`x = 1234.56e-1;`	exponent (means 1234.56×10^{-1})
`c = 1.6E-19;`	exponent (means 1.6×10^{-19})
`A = 6.02e23;`	exponent (means 6.02×10^{23})

SOME OF PRINTABLE ASCII CHARACTERS

Decimal	Binary	ASCII	Decimal	Binary	ASCII	Decimal	Binary	ASCII
32	00100000	SP (Space)	53	00110101	4	74	01001010	I
33	00100001	!	54	00110110	5	75	01001011	J
34	00100010	"	55	00110111	6	76	01001100	K
35	00100011	#	56	00111000	7	77	01001101	L
36	00100100	$	57	00111001	8	78	01001110	M
37	00100101	%	58	00111010	9	79	01001111	N
38	00100110	&	59	00111011	:	80	01010000	O
39	00100111	'	60	00111100	;	81	01010001	P
40	00101000	(Apostrophe)	61	00111101	<	82	01010010	Q
41	00101001	(	62	00111110	=	83	01010011	R
42	00101010	)	63	00111111	>	84	01010100	S
43	00101011	*	64	01000000	?	85	01010101	T
44	00101100	+	65	01000001	@	86	01010110	U
45	00101101	, (Comma)	66	01000010	A	87	01010111	V
46	00101110	- (Hyphen)	67	01000011	B	88	01011000	W
47	00101111	. (Period)	68	01000100	C	89	01011001	X
48	00110000	/	69	01000101	D	90	01011010	Y
49	00110001	0	70	01000110	E	91	01011011	Z
50	00110010	1	71	01000111	F	92	01011100	[
51	00110011	2	72	01001000	G	93	01011101	\
52	00110100	3	73	01001001	H	94	01011110	]

char

- C++ represents character values as numeric codes
- A variable of data type **`char`** can store a single character
- To represent a character constant in a program, we enclose the character in single quotes (apostrophes):

```
'A'        'b'        ' '        ';'
```

- Since characters are represented by integer codes , C++ permits conversion of type `char` to type `int` and vice versa

- For example, you could use the following fragment to find out the code your implementation uses for a question mark:

```
int  qmarkCode= '?';

cout << "Code for ? = " << qmarkCode << endl;
```

Strings

- One of the C++ standard libraries provides a data type string to represent a group of characters.
- Use of this data type requires inclusion of the preprocessor directive:

```
#include <string>
```

- String variables and name constants are declared, initialized, input, and displayed in a manner comparable to numbers and characters
- Notice that you must use double quotes to enclose a string's value in a program, and you may include any of the special characters discussed earlier.
- For example:

```
string a,b;
a="Sample 1";
b="Sample 2";
cout<<"a="<<a<<endl;
cout<<"b="<<b<<endl;
```

Escape codes

There are additional character literals called *escape codes or escape sequences which* are preceded by a backslash (****):

Escape Code	Description	Example
\a	alert (beep)	cout << "Error !\a";
\b	backspace	cout<<"Gazia\b antep";
\r	return to column 1	cout<<"gaziantep\rG";
\n	newline	cout <<'Gazi\nantep';
\t	horizontal tab	cout << x << '\t' << y;
\'	quote	cout << "Gaziantep\' te ";
\\	backslash	cout << "Gaziantep\Sahinbey";

Boolean Literals

- C++ defines a data type named bool that has only two possible values `true` and `false`.

- This is the type of the conditional expressions, such as
- `x > 100` and `y < = 0`, that we will study in selective structures structures.

- Sometimes a program uses a variable of type `bool` to keep track of whether a certain event has occurred.

- The variable will be initialized to `false`, and after the event occurs the variable will be set to `true`.

```
int main(){
bool a,b;
a=true;
b=1>2;
cout<<"a="<<a<<endl;
cout<<"b="<<b<<endl;
```