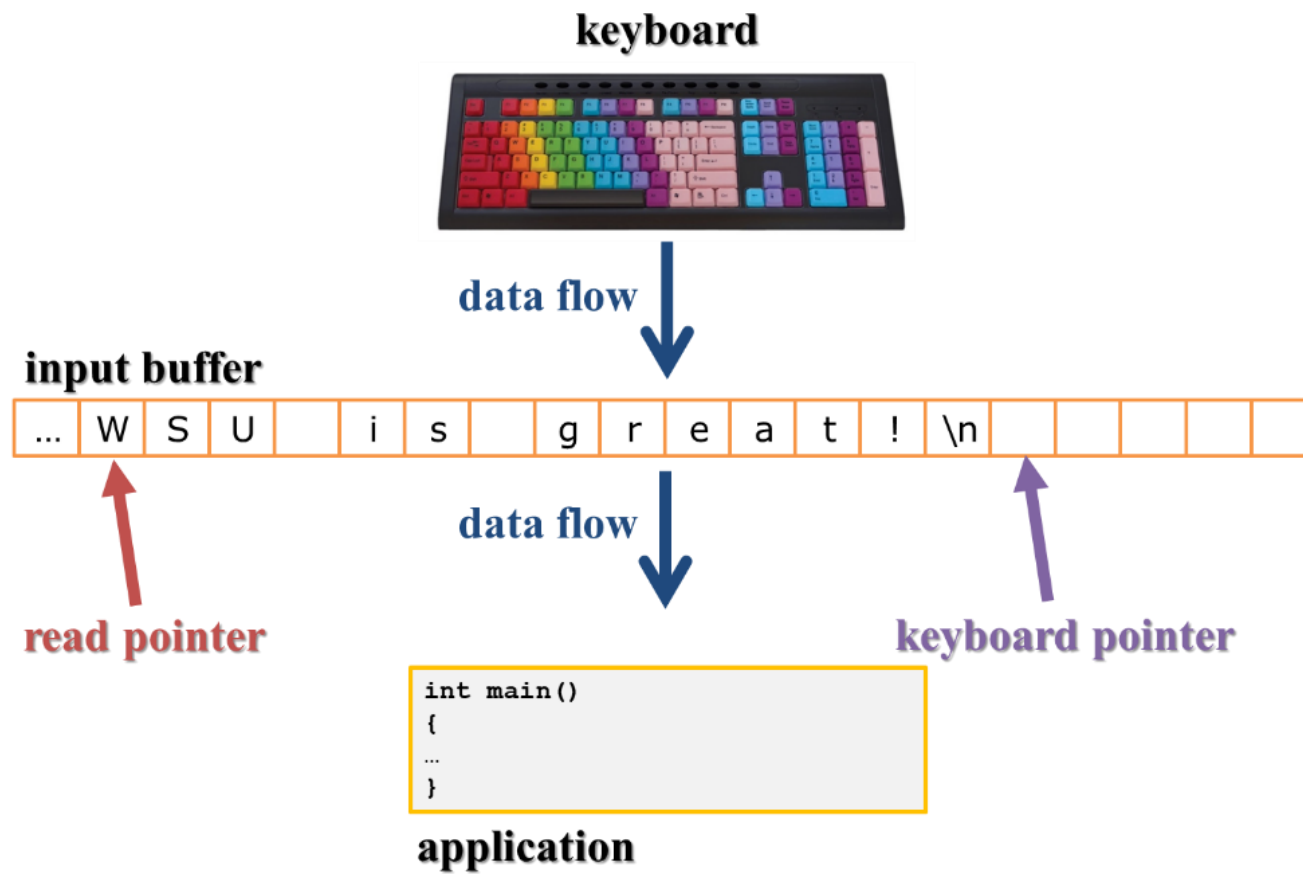


# Lecture 05

M M Imran

# Input Buffer

- The **input buffer** is an area within memory that holds data keyed by the user.
- The input buffer acts as an intermediary between the keyboard and the application.
- The input buffer may be visualized as an endless paper tape.
- Any data keyed by the user is appended to the end of the input buffer at the **keyboard pointer**.
- Each read by the application from the input buffer starts at the **read pointer**.




# Common In (cin)

- Object **cin** reads and stores a string (without spaces and tab characters) from the input buffer.
- **cin** is followed by the extraction operator (>>) followed by a variable name.
- cin syntax:
  - **cin >> <variable>;**
  - **cin >> <variable-1> >> <variable-2> >> ... >> <variable-n>;**
- Style alert: don't read more than one variable per cin.
- cin is almost always preceded by a cout statement.  
**cout << "Enter a value for the variable: ";**  
**cin >> <variable>;**

# cin performs these steps

- Once the user presses ENTER after the token is read, cin ends.
- Note that cin leaves at least one whitespace character in the input buffer, usually the ENTER character.
- When cin is executed and the user enters a value that matches the data type of the variable, good things happen.
- When cin is executed and the user DOES NOT enter a value that matches the data type of the variable, BAD things MAY happen.

<u>cin</u> behavior				
Data type casted to (data type of variable being read in to)	Data type of expression to be cast (data entered by user at keyboard)			
	int expression →	float/double expression →	string expression* →	char expression** →
int variable ←	Stores int.	Truncated value stored in int.	Value stored in int unchanged.	Value stored in int unchanged.
float/double variable ←	Stores int as float/double.	Stores float/double.	Value stored in float/double unchanged.	Value stored in float/double unchanged.
string variable ←	Stores int as string (number 45 stored as string “45”).	Stores float/double as string (number 7.4 stored as string “7.4”).	Stores string.	Stores char as string (char ‘M’ stored as “M”).
char variable ←	Stores first digit of int as char. If single digit int entered, okay. If <u>multiple-digit</u> int entered, not okay.	Stores first digit of float/double as char. If single digit float/double entered, okay. If <u>multiple-digit</u> float/double entered, not okay.	Stores first character of string as char.	Stores char.



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

int main() {
    int num;

    cout << "Enter a number: ";

    // take integer input
    cin >> num;

    cout << "You entered: " << num;

    return 0;
}
```

## Output

```
Enter a number: 25
You entered: 25
```

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

int main() {
    int num1, num2, num3;

    cout << "Enter a number: ";

    // for single input
    cin >> num1;

    cout << "Enter 2 numbers:" << endl;

    // for multiple inputs
    cin >> num2 >> num3;

    cout << "Sum = " << (num1 + num2 + num3);

    return 0;
}
```

## Output

```
Enter a number: 9
Enter 2 numbers:
1
5
Sum = 15
```



# cin with Member Functions

- **cin.get(char &ch):** Reads an input character and stores it in ch.
- **cin.getline(char \*buffer, int length):** Reads a stream of characters into the string buffer, It stops when
  - it has read *length-1* characters or
  - when it finds an end-of-line character '*\n*' or the end of the file *eof*.
- **cin.read(char \*buffer, int n):** Reads n bytes (or until the end of the file) from the stream into the buffer.
- **cin.ignore(int n):** Ignores the next n characters from the input stream.
- **cin.eof():** Returns a non-zero value if the end of file (eof) is reached.

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

int main() {
    char name[20], address[20];

    cout << "Name: ";

    // use cin with getline()
    cin.getline(name, 20);

    cout << "Address: ";

    cin.getline(address, 20);

    cout << endl << "You entered " << endl;
    cout << "Name = " << name << endl;
    cout << "Address = " << address;

    return 0;
}
```

## Output

```
Name: Sherlock Holmes
Address: Baker Street, UK

You entered
Name = Sherlock Holmes
Address = Baker Street, UK
```

# C++ Strings

- A string variable contains a collection of characters surrounded by double quotes:
- String syntax: **string greeting = "Hello";**
- To use strings, you must include an additional header file in the source code, the `<string>` library: **#include <string>**
- There are two types of strings commonly used in C++ programming language:
  - Strings that are objects of string class (The Standard C++ Library string class)
  - C-strings (C-style Strings)
- A string variable can hold from 0 to 2,147,483,647 characters.

# C-strings

- In C programming, the collection of characters is stored in the form of arrays. This is also supported in C++ programming. Hence it's called C-strings.
- C-strings are arrays of type char terminated with null character.
- define a C-string

```
char str[] = "C++";  
char str[4] = "C++";  
char str[] = {'C', '+', '+', '\\0'};  
char str[4] = {'C', '+', '+', '\\0'};  
// it is not necessary to use all the space allocated for the string  
char str[100] = "C++";
```

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

int main()
{
    char str[100];

    cout << "Enter a string: ";
    cin >> str;
    cout << "You entered: " << str << endl;

    cout << "\nEnter another string: ";
    cin >> str;
    cout << "You entered: "<<str<<endl;

    return 0;
}
```

## Output

Enter a string: C++

You entered: C++

Enter another string: Programming is fun.

You entered: Programming

Notice that, in the second example only "Programming" is displayed instead of "Programming is fun".

This is because the extraction operator >> works as scanf() in C and considers a space " " has a terminating character.

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

int main()
{
    char str[100];
    cout << "Enter a string: ";
    cin.get(str, 100);

    cout << "You entered: " << str << endl;
    return 0;
}
```

### Output

```
Enter a string: Programming is fun.
You entered: Programming is fun.
```

First argument is the name of the string (address of first element of string) and second argument is the maximum size of the array.

In the above program, str is the name of the string and 100 is the maximum size of the array.

# string Object

- Unlike using char arrays, string objects has no fixed length, and can be extended as per your requirement.
- It is possible to use the extraction operator >> on cin to display a string entered by a user:
- However, cin considers a space (whitespace, tabs, etc) as a terminating character, which means that it can only display a single word (even if you type many words):

```
string fullName;  
cout << "Type your full name: ";  
cin >> fullName;  
cout << "Your name is: " << fullName;
```

```
// Type your full name: John Doe  
// Your name is: John
```

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

int main()
{
    // Declaring a string object
    string str;
    cout << "Enter a string: ";
    getline(cin, str);

    cout << "You entered: " << str << endl;
    return 0;
}
```

## Output

```
Enter a string: Programming is fun.
You entered: Programming is fun.
```

When working with strings, we often use the **getline()** function to read a line of text. It takes **cin** as the first parameter, and the **string variable** as second.



# Difference Between C++ String and Character Array

- Character array is an array, but a string is a class that defines an object.
- Character array size is pre-allocated statically. Hence, extra memory is not available during runtime, and it wastes the memory that is not used. Since string size is not pre-allocated, there is no wastage, and extra memory is available during run time.
- Character array has the risk of array decay, but that is not the case with string.
- Strings are slower than character arrays when it comes to implementation.
- The String class offers more in-built functions to work with and manipulate strings.

# C++ String Functions

Sr.No	Function & Purpose
1	<b>strcpy(s1, s2);</b> Copies string s2 into string s1.
2	<b>strcat(s1, s2);</b> Concatenates string s2 onto the end of string s1.
3	<b>strlen(s1);</b> Returns the length of string s1.
4	<b>strcmp(s1, s2);</b> Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
5	<b>strchr(s1, ch);</b> Returns a pointer to the first occurrence of character ch in string s1.
6	<b>strstr(s1, s2);</b> Returns a pointer to the first occurrence of string s2 in string s1.

Function/constant*	Purpose
<code>&lt;str-var&gt;.length()</code> OR <code>&lt;str-var&gt;.size()</code>	Return the number of characters in <code>&lt;str-var&gt;</code> .
<code>&lt;str-var&gt;.max_size()</code>	Return the maximum number of characters that may be stored in <code>&lt;str-var&gt;</code> . It is always 2,147,483,647.
<code>&lt;str-var&gt;.at(&lt;int-exp&gt;)</code>	Return the character within <code>&lt;str-var&gt;</code> at position <code>&lt;int-exp&gt;</code> .
<code>&lt;str-var&gt;.substr(&lt;start-int-exp&gt;, &lt;length-int-exp&gt;)</code>	Return part of <code>&lt;str-var&gt;</code> , beginning at <code>&lt;start-int-exp&gt;</code> for <code>&lt;length-int-exp&gt;</code> characters.
<code>&lt;str-var&gt;.find(&lt;str-exp&gt;)</code>	Return the index of the first occurrence of <code>&lt;str-exp&gt;</code> in <code>&lt;str-var&gt;</code> . If <code>&lt;str-exp&gt;</code> doesn't occur, return <code>npos</code> .
<code>&lt;str-var-1&gt;.replace(&lt;start-int-exp&gt;, &lt;length-int-exp&gt;, &lt;str-var-2&gt;)</code>	Replace part of <code>&lt;str-var-1&gt;</code> with <code>&lt;str-var-2&gt;</code> , beginning at <code>&lt;start-int-exp&gt;</code> for <code>&lt;length-int-exp&gt;</code> characters.
<code>&lt;str-var-1&gt;.append(&lt;str-var-2&gt;)</code>	Add <code>&lt;str-var-2&gt;</code> to end of <code>&lt;str-var-1&gt;</code> .
<code>stoi(&lt;str-exp&gt;)</code>	Attempt to convert <code>&lt;str-exp&gt;</code> to an integer. This must be placed within a try-catch statement. A run-time error is generated if <code>&lt;str-exp&gt;</code> is not a valid integer.
<code>stod(&lt;str-exp&gt;)</code>	Attempt to convert <code>&lt;str-exp&gt;</code> to a real-number (double). This must be placed within a try-catch statement. A run-time error is generated if <code>&lt;str-exp&gt;</code> is not a valid real-number.

```
// Take any string
string s1 = "Geeks";

// Copy three characters of s1 (starting
// from position 1)
string r = s1.substr(1, 3);

// prints the result
cout << "String is: " << r;
```

String is: eek

```
// Take any string
string s = "dog:cat";

// Find position of ':' using find()
int pos = s.find(":");

// Copy substring after pos
string sub = s.substr(pos + 1);

// prints the result
cout << "String is: " << sub;
```

## Output

String is: cat

# Useful String and character functions

## Uses ctype header

Function*	Purpose
(char) toupper(<chr>)	Return the upper case version of <chr>.
(char) tolower(<chr>)	Return lower case version of <chr>.
isdigit(<chr>)	Return true (non-zero) if <chr> is a digit, false (zero) if not.
isalpha(<chr>)	Return true (non-zero) if <chr> is a letter, false (zero) if not.
isalnum(<chr>)	Return true (non-zero) if <chr> is a letter or digit, false (zero) if not.
islower(<chr>)	Return true (non-zero) if <chr> is a lower-case letter, false (zero) if not.
isupper(<chr>)	Return true (non-zero) if <chr> is an upper-case letter, false (zero) if not.
ispunct(<chr>)	Return true (non-zero) if <chr> is a punctuation character, false (zero) if not.

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

int main() {

    // convert 'a' to uppercase
    char ch = toupper('a');

    cout << ch;

    return 0;
}

// Output: A
```

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

int main() {

    // check if '27' is an alphabet
    int result = isalpha('27');

    cout << result;

    return 0;
}

// Output: 0
```



# Types of errors in C++

1. Syntax Errors
2. Run-time Error
3. Linker Errors
4. Logical Errors
5. Semantic errors

# Syntax Errors

- When you violate the rules of writing C/C++ syntax are known as syntax errors.
- A compiler error indicates that something needs to be fixed before the code can be compiled.
- The compiler detects all of these errors, so they are known as compile-time errors.
- The most frequent syntax errors are:
  - Missing Parenthesis (})
  - Printing the value of a variable without declaring it

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

void main()
{
    int x = 10;
    int y = 15;

    cout << " " << (x, y) // semicolon missed
}
```

## Error

```
error: expected ';' before '}' token
```

# Run-time Errors

- Errors which occur during program execution(run-time) after successful compilation are called run-time errors.
- Division by zero, also known as a division error, is one of the most common run-time errors.
- The compiler does not directly point to the line where the error occurs for these types of errors.
- Runtime errors are commonly called referred to as “bugs” and are often found during the debugging process before the software is released.

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

void main()
{
    int n = 9, div = 0;

    // wrong logic number is divided by 0,
    // so this program abnormally terminates
    div = n/0;

    cout << "result = " << div;
}
```

### Error

```
warning: division by zero [-Wdiv-by-zero]
    div = n/0;
```

These types of error are hard to find as the compiler does not point to the line at which the error occurs.

# Linker Errors

- These error occurs when after compilation we link the different object files with main's object using Ctrl+F9 key(RUN)
- When an executable of the program cannot be generated, these errors are generated.
- Incorrect function prototyping, incorrect header files may be to blame.
- Writing Main() instead of main() is one of the most common linker errors.

```
#include <bits/stdc++.h>
using namespace std;

void Main() // Here Main() should be main()
{
    int a = 10;
    cout << " " << a;
}
```

# Logical Errors

- When certain input values are given to a program during compilation and execution, desired output is not obtained.
- Logic errors are errors that appear to be error-free but provide incorrect output.
- They are among the most common programming errors made by beginners.
- Errors like these depend solely on the logic of the programmer and are easily detected if we follow the path of execution and find out why the program follows that path.
- Example: find a number greater than 10 and smaller than 5.



# Semantic errors

- A semantic error occurs when the statements written in the program do not make sense to the compiler.

```
#include <bits/stdc++.h>
using namespace std;
void main()
{
    int a, b, c;
    a + b = c; //semantic error
}
```

## Error

```
error: lvalue required as left operand of assignment
    a + b = c; //semantic error
```

# Increment and Decrement Operator

- In programming (Java, C, C++, JavaScript etc.), the increment operator `++` increases the value of a variable by 1. Similarly, the decrement operator `--` decreases the value of a variable by 1.
- If you use the `++` operator as a prefix like: `++var`, the value of `var` is incremented by 1; then it returns the value.
- If you use the `++` operator as a postfix like: `var++`, the original value of `var` is returned first; then `var` is incremented by 1.

# Postfix Operators

**Post-increment operator:** A post-increment operator is used to increment the value of a variable after executing the expression in which the operator is used. With the post-increment operator, the value of the variable is first used in an expression and then incremented.

**Post-decrement operator:** A post-decrement operator is used to decrement the value of a variable after executing the expression in which the operator is used. With the post-decrement operator, the value of the variable is first used in an expression and then decremented.

```
int x = 10;
```

```
int a;
```

```
a = x++;
```

```
int x = 10;
```

```
int a;
```

```
a = x--;
```

# Prefix Operators

**Pre-increment operator:** A pre-increment operator is used to increment the value of a variable before using it in an expression. With the pre-increment operator, the value of the variable is first incremented and then used in an expression.

**Pre-decrement operator:** A pre-decrement operator is used to decrement the value of a variable before using it in an expression. With the pre-decrement operator, the value of the variable is first decremented and then used in an expression.

```
int x = 10;
```

```
int a;
```

```
a = ++x;
```

```
int x = 10;
```

```
int a;
```

```
a = --x;
```

```
#include <iostream>

using namespace std;

int main()
{
    int x = 7;

    cout << "Value of x is " << x << endl;

    cout << "Value of x is now" << x++ << endl;

    cout << "Value of x is now " << x << endl;

    cout << "Value of x is now " << ++x << endl;

    cout << "Value of x is now " << x << endl;

    return 0;
}
```

```
Value of x is now 7
Value of x is now 8
Value of x is now 9
Value of x is now 9
```

```
#include <iostream>

using namespace std;

int main()
{
    int a = 21;
    int c ;
    c = a++;
    cout << c << endl;
    cout << a;

    return 0;
}
```

21  
22