Programs:

1. **Compare cin, cin.get() and cin.getline()**

```
#include<iostream>
#include<conio.h>
using namespace std;

int main()
{
	const int s = 100;
	char str1[s], str2[s], str3[s];

	cout << "Enter a sentence: " << endl;
	cin >> str1;
	cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
	cout << "\nThe string read with cin was:\n" << str1 << endl;

	cout << "\nEnter a sentence: " << endl;
	cin.get(str2, 10);
	cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    cout << "\nThe string read with cin.get was:\n" << str2 << endl;

	cout << "\nEnter a sentence: " << endl;
	cin.getline(str3, 20);
	cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
	cout << "\nThe string read with cin.getline was:\n" << str3 << endl <<
endl;

	return 0;
}
```

Output:

```
Enter a sentence:
This is a test string

The string read with cin was:
This

Enter a sentence:
This is a test string

The string read with cin.get was:
This is a

Enter a sentence:
This is a test string

The string read with cin.getline was:
This is a test stri

Press any key to continue . . .
```

2. **Demostrate use of peek, putback and ignore**

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
	char ch;
	cout << "Enter a sentence: " << endl;
	while (cin.get(ch))
	{
		if (ch == '!')
			cin.putback('$');
		else
			cout << ch;

		while (cin.peek() == '#')
			cin.ignore(1, '#');
	}
	_getch();
}
```

Output:

```
Enter a sentence:
hello!
hello$
#ello!!!
ello$$$
```

3. Demostrate use of read, write and gcount.

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{

	const int s = 100;
	char str[s];
	cout << "Enter sentence:" << endl;
	cin.read(str, 20);
	cout << "The string read was:" << endl;
	cout.write(str,cin.gcount());
	cout << "\nThe string had " << cin.gcount() << " characters";
	_getch();
	return 0;
}
```

Output:

```
Enter sentence:
this is a test string
The string read was:
this is a test strin
The string had 20 characters
```

4. **Print an integer value in octal, hexadecimal and decimal and setbase.**

```
#include<iostream>
#include<conio.h>
#include<iomanip>
using namespace std;
int main()
{
	int n;
	cout << "Enter a decimal number" << endl;
	cin >> n;
	cout << n << " in HexaDecimal is: " << hex << n << endl;
	cout << dec << n << " in Octal is: " << oct << n << endl;
	cout << setbase(10) << n << " in decimal is: " << n << endl;
	_getch();
}
```

Output:

```
Enter a decimal number
44
44 in HexaDecimal is: 2c
44 in Octal is: 54
44 in decimal is: 44
```

5. Demostrate use of precision and setprecision

```
#include<iostream>
#include<conio.h>
#include<iomanip>
#include<cmath>
using namespace std;

int main()
{
	double r2 = sqrt(2.0);

	cout << "Square root of 2 with precision 0-9.\n";
	cout << "Precision set by ios_base member function precision: \n";
	cout << fixed;
```

```
	for (int i = 0;i <= 9;i++)
	{
	cout.precision(i);
	cout << r2 << endl;
	}
	cout << "\nPrecision set by io manipulator member function
setprecision: \n";
	for (int i = 0;i <= 9;i++)
		cout << setprecision(i) << r2 << endl;

	_getch();
	return 0;
}
```

Output:

```
Square root of 2 with precision 0-9.
Precision set by ios_base member function precision:
1
1.4
1.41
1.414
1.4142
1.41421
1.414214
1.4142136
1.41421356
1.414213562

Precision set by io manipulator member function setprecision:
1
1.4
1.41
1.414
1.4142
1.41421
1.414214
1.4142136
1.41421356
1.414213562
```

6. Demostrate use of
    a. showpoint,
    b. left, right and internal justification
    c. scientific and fixed notation
    d. boolalpha

```
#include <iostream>
#include<iomanip>
#include<conio.h>
using namespace std;
int main()
{
	cout << "Implemening showpoint:\n";
	cout << "Before using showpoint" << endl
		<< "9.9900 prints as: " << 9.9900 << endl
		<< "9.9000 prints as: " << 9.9000 << endl
```

```
            << "9.0000 prints as: " << 9.0000 << endl;
      cout << showpoint
            << "After using showpoint" << endl
            << "9.9900 prints as: " << 9.9900 << endl
            << "9.9000 prints as: " << 9.9000 << endl
            << "9.0000 prints as: " << 9.0000 << endl << endl;

      cout << "Implementing left, right and internal justification:\n";
      int a = 12345;
      cout << "Default is right justified:" << endl << setw(10) << a;
      cout << "\nUse std::left to left justify x:\n" << left << setw(10) <<
a;
      cout << "\nUse std::right to right justify x:\n" << right << setw(10)
<< a << endl << endl;

      cout << "Implementing scientific and fixed notation:\n";
      double x = 0.001234567;
      double y = 1.946e9;
      cout << "Displayed in default format:" << endl << x << '\t' << y <<
endl;
      cout << "Displayed in scientific format:" << endl << scientific << x <<
'\t' << y << endl;
      cout << "Displayed in fixed format:" << endl << fixed << x << '\t' << y
<< endl << endl;

      cout << "Implementing boolalpha:\n";
      bool booleanValue = true;
      cout << "BooleanValue is " << booleanValue << endl;
      cout << "BooleanValue (after using boolalpha) is " << boolalpha <<
booleanValue << endl;
      cout << "Switch booleanValue and use noboolalpha" << endl;
      booleanValue = false;
      cout << noboolalpha;
      cout << "BooleanValue is " << booleanValue << endl;
      cout << "BooleanValue (after using boolalpha) is " << boolalpha <<
booleanValue << endl;

      _getch();
      return 0;
}
```

Output:

```
Implemening showpoint:
Before using showpoint
9.9900 prints as: 9.99
9.9000 prints as: 9.9
9.0000 prints as: 9
After using showpoint
9.9900 prints as: 9.99000
9.9000 prints as: 9.90000
9.0000 prints as: 9.00000

Implementing left, right and internal justification:
Default is right justified:
     12345
Use std::left to left justify x:
12345
Use std::right to right justify x:
     12345

Implementing scientific and fixed notation:
Displayed in default format:
0.00123457      1.94600e+09
Displayed in scientific format:
1.234567e-03    1.946000e+09
Displayed in fixed format:
0.001235        1946000000.000000

Implementing boolalpha:
BooleanValue is 1
BooleanValue (after using boolalpha) is true
Switch booleanValue and use noboolalpha
BooleanValue is 0
BooleanValue (after using boolalpha) is false
```

7. program to create user defined output stream manipulators.

```
#include<iostream>
#include<conio.h>
using namespace std;
ostream& bell(ostream& output){
return output << "\a";
}
ostream& carraigeReturn(ostream& output){
return output << "\r";
}
ostream& tab(ostream& output){
return output << "\t";
}
ostream& endLine(ostream& output){
return output << "\n" << flush;
}
int main(){
      cout << "Use Of tab and endline manipulator" << endLine;
      cout << "a" << tab << "b" << tab << "c" << endLine;
      cout << "Use of carraigeReturn and bell maniplulator" << endLine;
      cout << bell;
      cout << carraigeReturn << "-----" << endLine;
      _getch();
      return 0;
}
```

Output:

```
Use Of tab and endline manipulator
a       b       c
Use of carraigeReturn and bell maniplulator
-----
```

8. Show stream error states with examples

using namespace std;

```
int main()
{
	int integerValue;

	cout << " Before a bad input operation:"
		<< "\n cin.rdstate(): " << cin.rdstate()
		<< "\n cin.eof(): " << cin.eof()
		<< "\n cin.fail(): " << cin.fail()
		<< "\n cin.bad(): " << cin.bad()
		<< "\n cin.good(): " << cin.good()
		<< "\n\n Expects an integer, but enter a character: ";
	cin >> integerValue;
	cout << endl;

	cout << " After a bad input operation:"
		<< "\n cin.rdstate(): " << cin.rdstate()
		<< "\n cin.eof(): " << cin.eof()
		<< "\n cin.fail(): " << cin.fail()
		<< "\n cin.bad(): " << cin.bad()
		<< "\n cin.good(): " << cin.good() << endl << endl;

	cin.clear();
	cout << " After cin.clear()" << "\n cin.fail(): " << cin.fail() << "\n
cin.good(): " << cin.good() << endl;

	_getch();
}
```

Output:

```
 Before a bad input operation:
 cin.rdstate(): 0
 cin.eof(): 0
 cin.fail(): 0
 cin.bad(): 0
 cin.good(): 1

 Expects an integer, but enter a character: c

 After a bad input operation:
 cin.rdstate(): 4
 cin.eof(): 0
 cin.fail(): 1
 cin.bad(): 0
 cin.good(): 0

 After cin.clear()
 cin.fail(): 0
 cin.good(): 1
```