C++: BITWISE OPERATORS

- Suppose that the bitwise representation of a variable, a, is 00000101, and another variable, b, is 00001001

Bitwise operator:		Example:
~exp	bitwise **complement**	~a // 11111010
exp & exp	bitwise **and**	a&b // 00000001
exp | exp	bitwise **or**	a|b // 00001101
exp ^ exp	bitwise **exclusive-or (XOR)**	a^b // 00001100
exp1 << exp2	**shift** exp1 **left** by exp2 bits	b<<1 // 00010010
exp1 >> exp2	**shift** exp1 **right** by exp2 bits	b>>1 // 00000100

The **left-shift operator** fills with zeros.
The **right-shift operator** fills with zeros if exp1 is an unsigned or positive variable.
Otherwise, it fills with 1 if exp1 is negative.

C++: INPUT/OUTPUT

- To use the input/output functions, you must include a header in the source code; the <iostream> library

Example:

```
#include <iostream>
using namespace std;
cout<< "Enter temperature"; // prints the statement
cin>> f; // asks user for input, the value is saved in f
```

```
std::cin>> f; // if “using namespace std;” is not used
```

C++: IF-ELSE STATEMENTS

- **if-else statements:**

E.g.

```
int x = 20;
int y = 18;
if (x > y)
  {cout << "x is greater than y"; }
else if (x < y)
  {cout << "x is smaller than y"; }
else
  {cout << "x equals y"; }

//Output: x is greater than y
```

This code:

```
int z = (x < y ? x : y);
```

is a shorthand for:

```
int z;
if ( x < y )
  {z = x;}
else
  {z = y;}
```

C++: SWITCH

- **Switch:** to select one of many code blocks to be executed.

Syntax

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
    break;   }
```

Example

```
Day d = SUN;
switch(d) {
  case FRI:
    // code block
    break;
  case SAT:
    // code block
    break;
  default: // i.e. if not FRI or SAT
   // code block
    break;     }
```

C++: WHILE LOOP

Example

Infinite Loop!

```
int i = 0;
while (true) //condition
{
    cout<< i <<endl;
    i++;    //increment by 1
}
```

Output:

```
0
1
2
3
4
.
.
.
```

C++: DO/WHILE LOOP

- This loop executes the code block once. Then, it checks the condition, and repeats the loop as long as the condition is true.

Example

```
int i = 0;
do {
  cout<< i <<endl;
  i++;
}
while (i < 5);
```

Output:

```
0
1
2
3
4
```

C++: FOR LOOP

Example

```
for (int i = 5; i > 0; i--) {
      cout<< i <<endl;
    }
```

Output:

```
5
4
3
2
1
```

C++: BREAK/CONTINUE

- The **break** statement can be used to escape a **loop** when a condition is met.

- The **continue** statement skips one iteration (in the loop) when a condition is met, and continues with the next iteration in the loop.

C++: BREAK/CONTINUE

Example of break

```
for (int i = 0; i < 6; i++) {
    if (i == 2) {
        break;
    }
    cout << i << ”,” ;
}
//This prints: 0,1
```

Example of continue

```
for (int i = 0; i < 6; i++) {
    if (i == 2) {
        continue;
    }
    cout << i << ”,” ;
}
//This prints: 0,1,3,4,5
```

C++: STRUCTURES

- A structure is a **user-defined type** that contains multiple fields, also known as “members”.

Example:

```
enum MealType { NO_PREF,  REGULAR,  LOW_FAT,  VEGETARIAN };
struct  Passenger {
    string name;  // the passenger’s name
    MealType mealPref; // the passenger’s meal preference
    bool isFreqFlyer; // is the passenger in the Frequent-Flyer?
    string freqFlyerNo; // the passenger’s Frequent-Flyer number
};
//Now we can define variables of type “Passenger”
Passenger x  = {“John Smith”, LOW_FAT, true, X72199};
x.mealPref= VEGETARIAN; // We use “ .” to access any members of x
```

C++: FUNCTIONS

Syntax

```
ReturnType  nameFunction( type1 arg1,  type2 arg2,  … )
 {
    /* The “body” of the function, i.e., all the instructions
   inside it. */
 }
```

C++: FUNCTIONS

Example

```
int max(int x1, int x2, int x3)
{
   if(x1 > x2) {
     if(x1 > x3) return x1 ;
     else return x3;
   }
   else if(x2 > x3)  return x2;
   else return x3;
}
```

#To indicate that the function does not return anything, use the type “void”

```
void printMax(int x1, int x2, int x3)
{
   if(x1 > x2) {
     if(x1 > x3) cout << x1 ;
     else cout << x3 ;
   }
   else if(x2 > x3)  cout << x2 ;
   else cout << x3 ;
}
```

C++: THE "MAIN" FUNCTION

The "main" function is the one that gets executed automatically:

- By convention, the function `main` returns the value **zero** to indicate success and returns a nonzero value to indicate failure.
- The constant **EXIT SUCCESS** equals **zero**. Thus, it is OK if you write "`return 0;`" instead of "`return EXIT_SUCCESS;`"

Example:

```
int max(int x1, int x2){
   if(x1 > x2) return x1 ;
   else return x3;
}
int main(){
    int a = 5;
    int b = 7;
    int c = max(a,b);
    cout << "The max is: " << c;
    return EXIT_SUCCESS;
}
```

C++: FUNCTION OVERLOADING

- You can define functions that have **the same name but different types of arguments**. This is called "function overloading"

Example

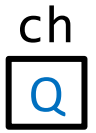
```
// Define a function called "print" with an integer argument
void print(int x)
{ cout << x; }

/* The function below has the same name as the one above, but
   that's OK, because the argument type is different! */
void print(Passenger x) {
     cout << x.name;
     cout << x.mealPref;
     if (x.isFreqFlyer )
          cout << pass.freqFlyerNo;   }
```

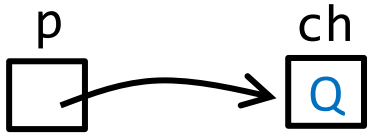
C++: POINTERS

- **Pointer:** a variable that stores a memory address

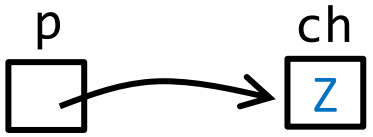
```
char ch = 'Q';
```



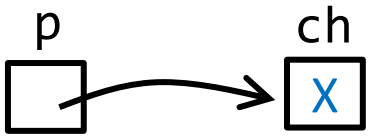

```
char* p = &ch; // p holds the address of ch
cout << *p;    // outputs the character 'Q'
```




```
ch = 'Z'; // ch now holds 'Z'
cout << *p;   // outputs the character 'Z'
```




```
*p = 'X'; // ch now holds 'X'
cout << ch;   // outputs the character 'X'
```



C++: POINTERS

- In C++, the name of an array is equivalent to a pointer to the array's first element

Example:

```
char c[ ] = {'c', 'a', 't'};
char* p = c; // p points to c[0]
char* q = &c[0]; // q also points to c[0]; &c[0] is equivalent to c
cout << c[2] << p[2] << q[2];    // outputs "ttt"
```