## C++ Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>boolean (true or false)</td>
</tr>
<tr>
<td>char</td>
<td>character ('a', 'b', etc.)</td>
</tr>
<tr>
<td>char[]</td>
<td>character array (C-style string if null terminated)</td>
</tr>
<tr>
<td>string</td>
<td>C++ string (from the STL)</td>
</tr>
<tr>
<td>int</td>
<td>integer (1, 2, -1, 1000, etc.)</td>
</tr>
<tr>
<td>long int</td>
<td>long integer</td>
</tr>
<tr>
<td>float</td>
<td>single precision floating point</td>
</tr>
<tr>
<td>double</td>
<td>double precision floating point</td>
</tr>
</tbody>
</table>

These are the most commonly used types; this is not a complete list.

## Operators

The most commonly used operators in order of precedence:

1. ++ (post-increment), -- (post-decrement)
2. | (not), ++ (pre-increment), -- (pre-decrement)
3. *, /, % (modulo)
4. +, -, +, -, =
5. <, <=, >, >=
6. == (equal-to), != (not-equal-to)
7. && (and)
8. || (or)
9. = (assignment), +=, -=, *=, /=, %=

## Console Input/Output

cout << console out, printing to screen
cin >> console in, reading from keyboard
cerr << console error

Example:
cout << "Enter an integer: ";
cin >> i;
cout << "Input: " << i << endl;

## File Input/Output

Example (input):
`ifstream inputFile;`  
`inputFile.open("data.txt");`  
Example (output):
`ofstream outFile;`  
`outFile.open("output.txt");`  

## Decision Statements

if (expression)  
statement;

if (expression)  
statement;
else  
statement;

switch (int expression)  
{
  case int-constant:
    statement(s);
    break;
  default:
    statement;
}

## Loops

### while Loop

```cpp
while (expression)  
{
  statement;
  x++;
}
```

### do-while Loop

```cpp
do  
  statement;
while (expression);
```

### for Loop

```cpp
for (initialization; test; update)  
{
  statement;
}
```

## Functions

Functions return at most one value. A function that does not return a value has a return type of `void`. Values needed by a function are called parameters.

```cpp
return_type function(type p1, type p2, ...)
{
  statement;
  statement;
  ...
}
```

### Passing Parameters by Value

`return_type function(type p1)`  
Variable is passed into the function but changes to `p1` are not passed back.

### Passing Parameters by Reference

`return_type function(type &p1)`  
Variable is passed into the function and changes to `p1` are passed back.

## Default Parameter Values

`return_type function(type p1=val)`  
`val` is used as the value of `p1` if the function is called without a parameter.

## Dynamic Memory

### Allocate Memory

```cpp
ptr = new type;
```

### Deallocate Memory

```cpp
delete ptr;
```

## Structures

```cpp
struct Hamburger
{
  // Members
  int patties;
  bool cheese;
  // Constructor
  Hamburger();
}
```

### Definition

```cpp
name* ptrName;
```

### Accessing Members

```cpp
ptrName->element = val;
```

Structures can be used just like the built-in data types in arrays.
Operator Overloading

C++ allows you to define how standard operators (+, -, *, etc.) work with classes that you write. For example, to use the operator + with your class, you would write a function named operator+ for your class.

Example
Prototype for a function that overloads + for the Square class:
Square operator+ (const Square &);

If the object that receives the function call is not an instance of a class that you wrote, write the function as a friend of your class. This is standard practice for overloading << and >>.

Example
Prototype for a function that overloads << for the Square class:
friend ostream & operator<< (ostream & a, const Square &);

Visibility of Members after Inheritance

<table>
<thead>
<tr>
<th>Inheritance Specification</th>
<th>Access Specifier in Base Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>private</td>
<td>private</td>
</tr>
<tr>
<td>protected</td>
<td>-</td>
</tr>
<tr>
<td>public</td>
<td>-</td>
</tr>
</tbody>
</table>

Function Templates

Example
template <class T>
T getMax(T a, T b) {
    if (a>b)
        return a;
    else
        return b;
}

// example calls to the function template
int a=9, b=2,
c = getMax(a, b);
float f=5.3, g=9.7, h;
h = getMax(f, g);

Class Templates

Example
template <class T>
class Point {
    public:
        Point(T x, T y);
        void print();
    private:
        T x;
        T y;
};

// examples using the class template
Point<int> p1(3, 2);
Point<float> p2(3.5, 2.5);
p1.print();
p2.print();

Exceptions

Example
try {
    // code here calls functions that might // throw exceptions
    quotient = divide(num1, num2);
    // or this code might test and throw // exceptions directly
    if (num3 < 0)
        throw -1; // exception to be thrown can // be a value or an object
}
catch (int)
{
    cout << "num3 can not be negative!";
    exit(-1);
}
catch (char* exceptionString)
{
    cout << exceptionString;
    exit(-2);
}
// add more catch blocks as needed

Class Templates