Loops

For Loop

```cpp
for(i = 1; i < 10; i++) {
    cout << i << " ";
}
```

**Output:** 1 2 3 4 5 6 7 8 9

```cpp
for(i = 1; i <= 10; i++) {
    cout << i << " ";
}
```

**Output:** 1 2 3 4 5 6 7 8 10
Loops

While Loop

```cpp
int i;
i = 5;

while(i < 10) {
    cout << i << " ";
    i = i + 1;
}
```

Output: 5 6 7 8 9
Loops

Do...While Loop

If you want the loop to execute at least once.

```cpp
int i;
i = 5;
do {
    cout << i << " ";
} while(i < 1);
```

Output: 5
Loops

Nested Loops

for(i = 1; i <= 5; i++) {
    for(j = 1; j <= i; j++) {
        cout << "[]";
    }
    cout << endl;
}

Output:
[]
 [[]
 [ ]
 [ ]
 [ ]
 [ ]
 [ ]
 [ ]
 [ ]
 [ ]
 [ ]
**const correctness**

The `const` keyword declares that a variable is meant to be constant, i.e. not allowed to change. This is useful to the programmer to ensure your data is not destroyed.

```cpp
int const pi = 3.1415926535;

pi = 3;
```

**Output:** Compiler error!
const correctness

```cpp
void f(const int& y);

int main()
{
    int x = 1;
    f(x);
    cout << x << "\n";
}

void f(const int& y) {
    y = y + 1;
}
```

**Output:** Compiler error. “assignment of read-only reference ‘y’”
Passing by reference v.s. Passing by value

```cpp
main() {
    int i = 10, j = 20;
    swapThemByVal(i, j);
    cout << i << " " << j << endl; // displays 10 20
    swapThemByRef(i, j);
    cout << i << " " << j << endl; // displays 20 10
}

void swapThemByVal(int num1, int num2) {
    int temp = num1;
    num1 = num2;
    num2 = temp;
}

void swapThemByRef(int& num1, int& num2) {
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

(Above example is thanks to Dr. Carol Zander, from University of Washington, Bothell)
class AA
{
    public:
    AA( int x )
    {
        cout << "AA’s constructor " << "called with " << x << endl;
    }
};

class BB : public AA
{
    public:
    BB() : AA( 10 ) // construct the AA part of BB
    {
        cout << "BB’s constructor" << endl;
    }
};

int main()
{
    BB thing;
}
When do you want to use a constructor initialization list:

- When you have to pass a parameter to a parent constructor.
- When you have a field that is a `const`.
- When you have a field that is a reference. References are immutable, so they can only be initialized once.
- If one of the classes use does not have a default constructor.
### Member functions v.s. Non-member functions

```cpp
class Rectangle {
    int width, height;

public:
    void set_values (int, int);
    int area() {return width*height;}
};

void Rectangle::set_values(int x, int y)  // member function
{
    width = x;
    height = y;
}

void set_values(Rectangle& rect)  // non-member function
{
    rect.width = x;
    rect.height = y;
}

int main ()
{
    Rectangle rect;
    rect.set_values (3,4);
    cout << "area: " << rect.area();
    set_values(5,6);  // compiler error
    return 0;
}
```

Alex Lin
PIC 10B Discussion: Week 1 – Tues
Spring 2017
Direct v.s. Indirect initialization

double a1(1.5);  // direct initialization
double a2 = 1.5;  // copy initialization
Direct v.s. Indirect initialization

Direct initialization is performed in the following situations:

1. initialization with a nonempty parenthesized list of expressions
2. during list-initialization sequence, if no initializer-list constructors are provided and a matching constructor is accessible, and all necessary implicit conversions are non-narrowing.
3. initialization of a prvalue temporary by functional cast or with a parenthesized expression list
4. initialization of a prvalue temporary by a static cast expression
5. initialization of an object with dynamic storage duration by a new-expression with a non-empty initializer
6. initialization of a base or a non-static member by constructor initializer list
7. initialization of closure object members from the variables caught by copy in a lambda-expression
Direct v.s. Indirect initialization

Copy initialization is performed in the following situations:

1. when a named variable (automatic, static, or thread-local) of a non-reference type \( T \) is declared with the initializer consisting of an equals sign followed by an expression.

2. (until C++11) when a named variable of a scalar type \( T \) is declared with the initializer consisting of an equals sign followed by a brace-enclosed expression (Note: as of C++11, this is classified as list initialization, and narrowing conversion is not allowed).

3. when passing an argument to a function by value

4. when returning from a function that returns by value

5. when throwing or catching an exception by value

6. as part of aggregate initialization, to initialize each element for which an initializer is provided
Good coding practices

(Will talk it out)
Disclaimer about slides

I claim no originality of the examples or instructional material above. Some I have created, others I have copied. I owe a tremendous thanks to those that have made their instruction of C++ available online.