if.cc

```
int main()
{
    int a {2};
    if ( a = 0 )
        cout << "A is zero\n";
    else
        cout << "Value of A is " << a << endl;
}
```

init.cc

```
int main()
{
    int i = 3.5;
    cout << i;
}
```

init-2.cc

```
int main()
{
    int i (3.5);
    cout << i;
}
```

init-3.cc

```
int main()
{
    int i {3.5};
    cout << i;
}
```

ref.cc

```
void fun(int const &){
    cout << 1;
}

void fun(int &){
    cout << 2;
}

void fun(int &&){
    cout << 3;
}

int main(){
    int a;
    int const c {};
    fun(23);
    fun(a);
    fun(c);
}
```

ref-2.cc

```
struct T{};
void fun(T const &){
    cout << 1;
}

void fun(T &&){
    cout << 3;
}

int main(){
    T a;
    T const c {};
    fun(T{});
    fun(a);
    fun(c);
}
```

templ.cc

```
template <typename T>
void foo(T)  {
    cout << 1;
}

void foo(int const &) {
    cout << 2;
}

int main() {
    int a;
    int const b{};
    foo(a);
    foo(3);
    foo(b);
}
```

templ-2.cc

```
template <typename T>
void foo(T &)  {
    cout << 1;
}

void foo(int const &) {
    cout << 2;
}

int main() {
    int a;
    int const b{};
    foo(a);
    foo(3);
    foo(b);
}
```

templ-3.cc

```
template <typename T>
void foo(T &&) {
    cout << 1;
}

void foo(int const &) {
    cout << 2;
}

int main() {
    int a;
    int const b{};
    foo(a);
    foo(3);
    foo(b);
}
```

arr.cc

```
int global_arr[20];
int main()
{
    int local_arr[20];
    cout << global_arr[0] << local_arr[0];
}
```

arr-2.cc

```
int global_arr[20];
int main()
{
    int local_arr[20] {};
    cout << global_arr[0] << local_arr[0];
}
```

static-const.cc

```
struct CLS
{
    static void fun() const
    {
        cout << "fun";
    }
};

int main()
{
    CLS c;
    c.fun();
    CLS::fun();
}
```

virt.cc

```
struct Base
{
    void fun() {
        cout << "Base::fun";
    }
};

struct Derived: Base
{
    void fun() {
        cout << "Derived::fun";
    }
};

void foo(Base const & b) {
    b.fun();
}

int main() {
    foo(Derived{});
}
```

virt-2.cc

```
struct Base
{
    void fun() {
        cout << "Base::fun";
    }
};

struct Derived: Base
{
    void fun() {
        cout << "Derived::fun";
    }
};

void foo(Base & b) {
    b.fun();
}

int main() {
    foo(Derived{});
}
```

virt-3.cc

```
struct Base
{
    void fun() {
        cout << "Base::fun";
    }
};

struct Derived: Base
{
    void fun() {
        cout << "Derived::fun";
    }
};

void foo(Base & b) {
    b.fun();
}

int main() {
    Derived d;
    foo(d);
}
```

virt-4.cc

```
struct Base
{
    virtual void fun() {
        cout << "Base::fun";
    }
};

struct Derived: Base
{
    virtual void fun() const {
        cout << "Derived::fun";
    }
};

void foo(Base & b) {
    b.fun();
}

int main() {
    Derived d;
    foo(d);
}
```

virt-5.cc

```
struct Base
{
    virtual void fun() const {
        cout << "Base::fun";
    }
};

struct Derived: Base
{
    void fun() const override {
        cout << "Derived::fun";
    }
};

void foo(Base const & b) {
    b.fun();
}

int main() {
    Derived d;
    foo(d);
}
```

unique.cc

```
int main()
{
    vector<int> vals {2,1,4,1};
    unique(begin(vals), end(vals));
    copy(begin(vals), end(vals), ostream_iterator<int>{cout, " "});
}
```

init-cls.cc

```
struct Base{
    Base(){
        cout << 3;
    }
    ~Base(){
        cout << 1;
    }
};

struct Derived{
    Derived(){
        cout << 5;
    }
    ~Derived(){
        cout << 8;
    }
};

int main(){
    Derived{};
}
```

init-member.cc

```
struct Data{
    Data(){
        cout << 3;
    }
    ~Data(){
        cout << 1;
    }
};

struct My_Class{
    My_Class(){
        cout << 5;
    }
    ~My_Class(){
        cout << 8;
    }
    Data a;
};

int main(){
    My_Class{};
}
```

sfinae.cc

```
template <typename T>
enable_if_t<is_integral<T>::value>
foo(T &&) {
    cout << 1;
}

int main()
{
    foo(2);
}
```

sfinae-2.cc

```
template <typename T>
enable_if_t<is_integral<T>::value>
foo(T &&) {
    cout << 1;
}

int main()
{
    int i{};
    foo(i);
}
```

sfinae-3.cc

```
template <typename T>
enable_if_t<is_integral<decay_t<T>>::value>
foo(T &&) {
    cout << 1;
}

int main()
{
    int i{};
    foo(i);
}
```

str.cc

```
int main()
{
    string{"*", 10};
    string{'*', 10};
    string('*', 10);
    string{"*"s, 10};
    string(10, '*');
    string{"**********"};
}
```

vartemp.cc

```
int foo(int, char)
{
    return 4;
}

template <typename Ret, typename Fun, typename ...Args>
Ret call(Fun f, Args... args)
{
    return f(args...);
}

int main()
{
    int a;
    char c;
    cout << call(foo, a, c);
}
```

vartemp-2.cc

```
int foo(int, char)
{
    return 4;
}

template <typename Ret, typename Fun, typename ...Args>
Ret call(Fun f, Args... args)
{
    return f(args...);
}

int main()
{
    int a;
    char c;
    cout << call<int>(foo, 4, c);
}
```

vartemp-3.cc

```
int foo(int, char &)
{
    return 4;
}

template <typename Ret, typename Fun, typename ...Args>
Ret call(Fun f, Args... args)
{
    return f(args...);
}

int main()
{
    int a;
    char c;
    cout << call<int>(foo, 4, c);
}
```

vartemp-4.cc

```
int foo(int &&, char &)
{
    return 4;
}

template <typename Ret, typename Fun, typename ...Args>
Ret call(Fun f, Args... &&args)
{
    return f(forward<Args>(args)...);
}

int main()
{
    int a;
    char c;
    cout << call<int>(foo, 4, c);
}
```

vartemp-5.cc

```
int foo(int &&, char &)
{
    return 4;
}

template <typename Ret, typename Fun, typename ...Args>
Ret call(Fun f, Args... &&args)
{
    return f(forward<Args>(args)...);
}

int main()
{
    int a;
    char c;
    cout << call<int>(foo, a, c);
}
```