Function overloading

```
/*
. Function overloading: Reusing the same function name with different sets of parameters
    . Overoading with different parameters
    . Overloading with pointer parameters
    . Overloading with references
    . Overloading with const parameters
    . Overloading with const pointer and pointer to const
    . Overloading with const references
    . Overloading with default parameters
*/
```

Function overloading: different parameters

```
//Overoading with different parameters
export int max(int a, int b)
{
  return (a > b) ? a : b;
}

export double max(double a, double b)
{
  return (a > b) ? a : b;
}

export double max(int a, double b)
{
  return (a > b) ? a : b;
}

export double max(double a, int b)
{
  return (a > b) ? a : b;
}

export double max(double a, int b, int c)
{
  return a;
}

export std::string_view max(std::string_view a, std::string_view b)
{
  return (a > b) ? a : b;
}
```

Function overloading: pointer parameters

```
//Overloading with pointer parameters
export double max(double *numbers, size_t count)
{
  fmt::println("doubles overload called");
  double maximum{ 0 };

  for (size_t i{ 0 }; i < count; ++i) {
    if (numbers[i] > maximum) maximum = numbers[i];
  }
  return maximum;
}

export int max(int *numbers, size_t count)
{
  fmt::println("ints overload called");

  int maximum{ 0 };

  for (size_t i{ 0 }; i < count; ++i) {
    if (numbers[i] > maximum) maximum = numbers[i];
  }
  return maximum;
}
```

Function overloading: reference parameters

```
//Overloading with references
// Ambiguous calls
export void say_my_name(const std::string &name) {
  fmt::println("Your name is (ref): {}", name);
}

export void say_my_name(std::string name) {
  fmt::println("Your name is (non ref): {}", name);
}

// Implicit conversions with references
export double max(double a, double b)
{
  fmt::println("double max called");
  return (a > b) ? a : b;
}

// int& max(int& a, int& b){
export const int &max(const int &a, const int &b)
{
  fmt::println("int max called");
  return (a > b) ? a : b;
}
```

Function overloading: const parameters

```
//Overloading with const parameters
//These two are similar in the eyes of the compiler
//If you change the copy in the body of the function,
//the outside variable won't be affected. const is meaningless here.
export int max(const int a, const int b)
{
  fmt::println("const int max called");
  return (a > b) ? a : b;
}

export int max(int a, int b)
{
  fmt::println("int max called");
  return (a > b) ? a : b;
}
```

Function overloading: const pointer and pointer to const

```
//Overloading with const pointer and pointer to const
export int max(int *a, int *b)
{
  fmt::println("max with int* called");
  return (*a > *b) ? *a : *b;
}

export int max(const int *a, const int *b)
{
  fmt::println("max with cont int* called");
  return (*a > *b) ? *a : *b;
}

//The two min functions are similar in the eyes of the compiler: Error!
export int min(const int* a, const int* b){
    return (*a < *b)? *a : *b;
}

export int min(const int *const a, const int *const b)
{
  fmt::println("&a : {}", fmt::ptr(&a));
  fmt::println("&b : {}", fmt::ptr(&b));
  return (*a < *b) ? *a : *b;
}
```

Function overloading: const references

```
//Overloading with const references
export int max(int &a, int &b)
{
  fmt::println("max with int& called");

  // Can change a and b through the reference
  // a = 200;// This change will be visible outside the function

  return (a > b) ? a : b;
}

export int max(const int &a, const int &b)
{
  fmt::println("max with const int& called");

  // Can NOT change a and b through the reference
  // a = 200; // Will give a compiler error.
  return (a > b) ? a : b;
}
```

Function overloading: default parameters

```
//Overloading with default parameters
export void print_age(int age = 18) {
    fmt::println( "Your age is( int version)  : {}" , age );
}

export void print_age(long int age) {
    fmt::println("Your age is (long int version) : {}", age);
}
```