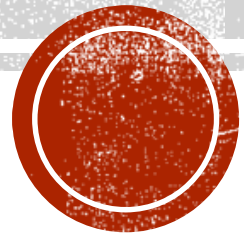
FUNCTION OVERLOADING



WHAT IS FUNCTION OVERLOADING?

- The C language forces us to use different names for similar functions.
- Example:

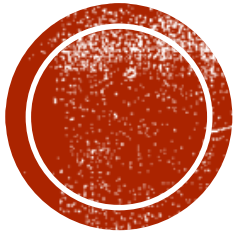
```
int abs(int);
double fabs(double);
long labs(long);
```

- But C++ lets us use just one name for all similar functions.
  - Function overloading is a feature in C++
- Allows two or more functions to have the same name with different parameters.
- This is an example of polymorphism feature in C++.

FUNCTION OVERLOADING

- Defining two or more functions using the same name but different parameters.
- Overloaded functions must differ in their parameter lists
  - must have either different numbers of parameters or
  - different types of parameters.
- When an overloaded function is called, the **compiler uses the argument list** to determine which function of that name is being invoked

```
int abs(int i);
double abs(double d);
long abs(long l);
```

PRACTICE EXAMPLE

Absoulte.cpp

FUNCTION OVERLOADING & RETURN TYPES

```
int getValue()
{
    return 1;
}

float getValue()
{
    return 1.2;
}
```

Compilation Error

FUNCTION OVERLOADING & RETURN TYPES

- Functions can not be overloaded if they differ only in the return type.

```
int getValue(int i);
float getValue(int i);

  // This attempt to overload will fail
```


AVOIDING AMBIGUITY

- When overloading functions, it is possible to create **ambiguous situations** where the compiler is unable to choose between two or more overloaded functions.
- If there are Ambiguous statements the compilations errors will occur.

Implicit type conversion

```
float getSquare(float f);
double getSquare(double d);

//Function call
cout << getSquare(10);
```


DEFAULT ARGUMENTS

- Default arguments are one possible cause of ambiguity.

```
void addPerson(char* firstName, char* lastName, int age = 0);

void addPerson(char* firstName, char* lastName);
```

- Attempt to call the second version of **addPerson()** will be ambiguous.

```
addPerson("Jane", "Citizen");
```

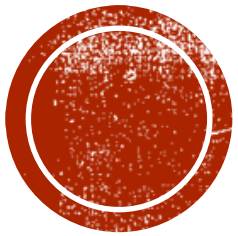

RESOLVING CALLS TO OVERLOADED FUNCTIONS

The compiler uses the following selection algorithm to help it make the best choice when resolving calls to overloaded functions:

1. Use an exact match if possible.
2. Try standard type promotions (e.g. **char** to **int**).
3. Try standard type conversions (e.g. **int** to **float**).
4. Try user-defined type conversions (e.g. conversion constructor or operator conversion function).

FUNCTION OVERLOADING IN SUMMARY

- Define multiple definitions for the same function name in the same scope.
- The definition of the function must differ from each other by the types and/or the number of arguments in the argument list.
- Cannot overload function declarations that differ only by return type.



QUIZ—FUNCTION OVERLOADING

QUESTION 01

Which of the following in Object Oriented Programming is supported by Function overloading?

1. Inheritance
2. Polymorphism
3. Encapsulation
4. None of the above

Answer: Polymorphism

QUESTION 02

Answer: All

Which of the following overloaded functions are NOT allowed in C++?

1. Function declarations that differ only in the return type

```
int fun(int x, int y);
void fun(int x, int y);
```

2. Functions that differ only by static keyword in return type

```
int fun(int x, int y);
static int fun(int x, int y);
```

3. Parameter declarations that differ only in a pointer * versus an array []

```
int fun(int *ptr, int n);
int fun(int ptr[], int n);
```

4. Two parameter declarations that differ only in their default arguments

```
int fun( int x, int y);
int fun( int x, int y = 10);
```