COMSW 1003-1

Introduction to Computer Programming in C

Lecture 23

Spring 2011

Instructor: Michele Merler

http://www1.cs.columbia.edu/~mmerler/comsw1003-1.html

Today

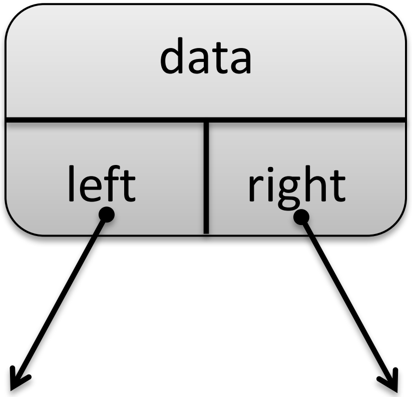
- Trees (from PCP Chapter 17)

- C++ and object oriented programming

Trees

Node struct

```
struct t_node {
    char *data;
    struct t_node *left;
    struct t_node *right;
};
```



```
typedef struct t_node node;
```

Trees

1) Root pointer = top of the tree

```
static node *root;
```

Global variable, everything refers to it, like the head in a linked list

2) save_string utility function

Malloc() + some checks

```
char *save_string( char *string ){

    char *new_string;

    new_string = malloc( (unsigned) (strlen(string) + 1));

    if( new_string == NULL ){
        memory_error();
    }

    strcpy( new_string, string );

    return( new_string );
}
```

Trees

3) `enter` function to insert a node in the tree (recursive!)

Example invocation: `enter( &root, “hello” );`

```
void enter( node **n, char *word){
    int result;
    if( (*n) == NULL ) {

        (*n) = malloc( sizeof(node) );
        if( (*n) == NULL )
            memory_error();

        (*n)->data = save_string( word );
        (*n)->left = NULL;
        (*n)->right = NULL;

        return;
    }
                    .
                    .
                    .
```

Trees

3) `enter` function to insert a node in the tree (recursive!)

Address of the node ↑

```
void enter( node **n, char *word){
    int result;
    if( (*n) == NULL ) {
        (*n) = malloc( sizeof(node) );
        if( (*n) == NULL )
            memory_error();

        (*n)->data = save_string( word );
        (*n)->left = NULL;
        (*n)->right = NULL;
        return;
    }
```

⋮

Reached bottom of the tree, must create and append new node

Allocate new node in memory

Initialize value of new node
Allocate new string (word) in memory

Trees

3) enter function to insert a node in the tree

```
void enter( node **n, char *word){
                    .
                    .
                    .
    result = strcmp( (*n)->data, word );

    if( result == 0 )
       return;

    if( result < 0 ){
       enter( &(*n)->right, word );
    }
    else{
       enter( &(*n)->left, word );
    }

}
```

Trees

3) enter function to insert a node in the tree

```
void enter( node **n, char *word){
                          :
    result = strcmp( (*n)->data, word );

    if( result == 0 )
       return;

    if( result < 0 ){
       enter( &(*n)->right, word );
    }
    else{
       enter( &(*n)->left, word );
    }

}
```

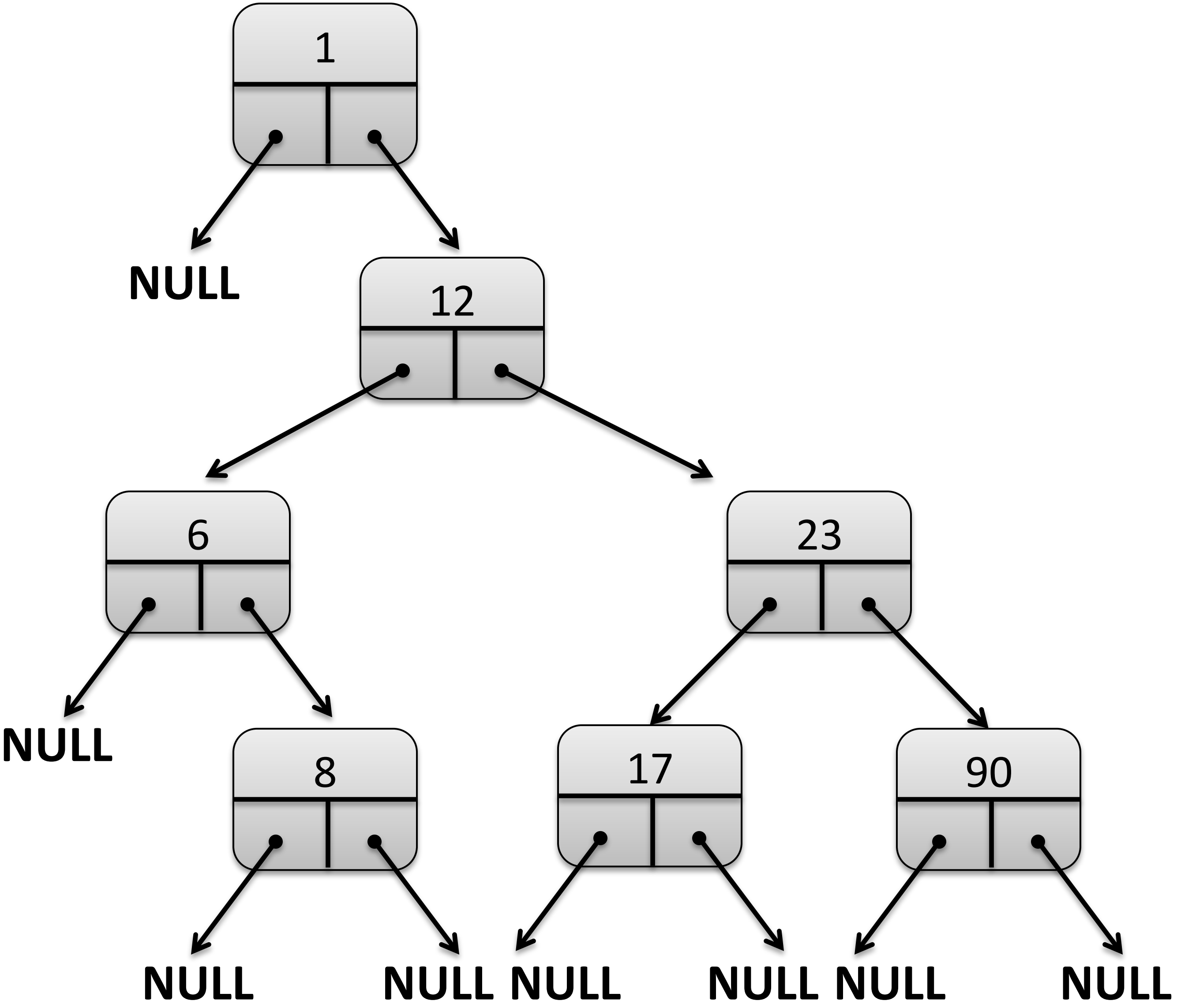
Comparison, check if we should go right or left

A node with this value (word) already exists, no need to insert another one

Recursive call!

Trees

Example: [1 12 6 23 17 90 8]



Trees

4) print_tree function to print the tree

```
void print_tree( node *top ){

     if( top == NULL ) {
            return;
     }

     print_tree( top->left );

     printf(”%s\n”, top->data );

     print_tree( top->right );
}
```

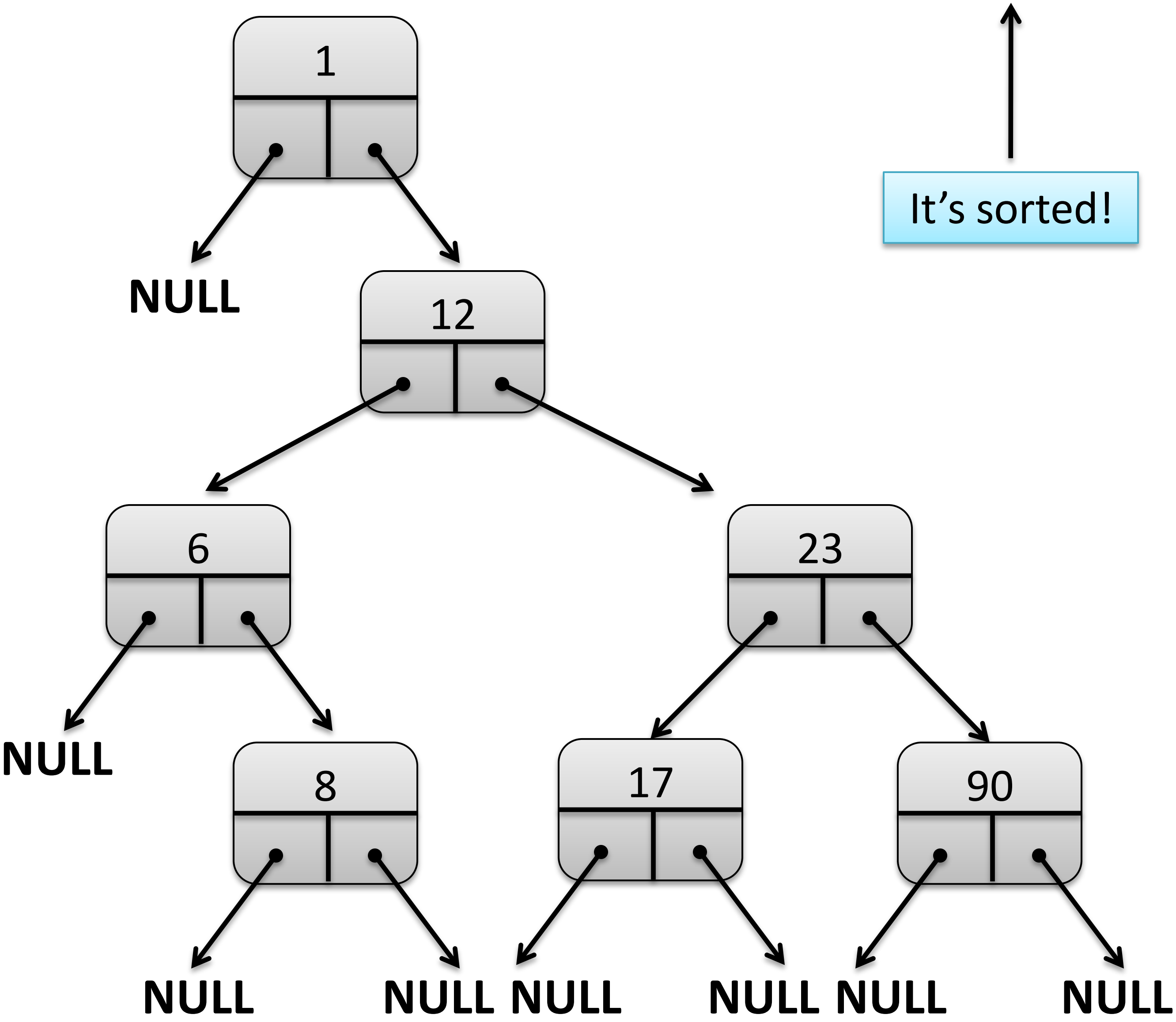
→ Empty tree

Recursive call!

Trees

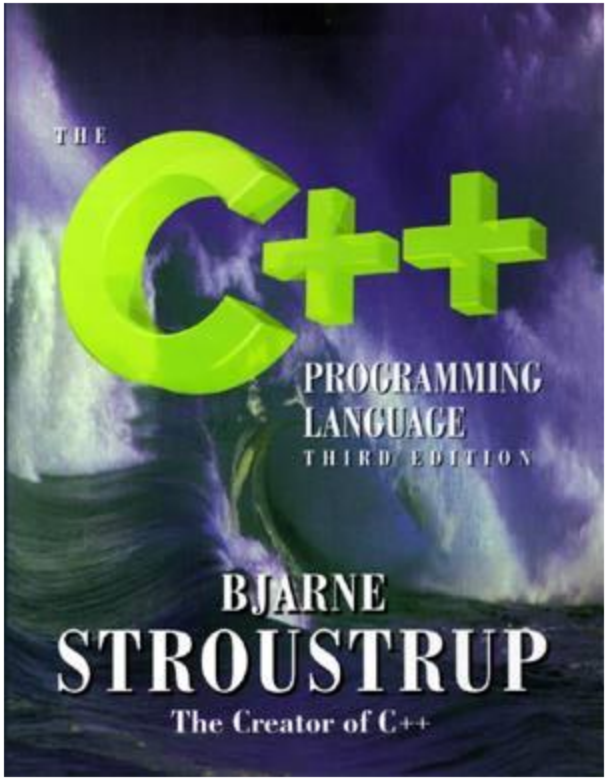
Insertion order: [1 12 6 23 17 90 8]

Print order: [1 6 8 12 17 23 90]



It's sorted!

C++



- Younger brother of C
- Appeared in 1983
- Object Oriented
- Can be compiled with gcc, usually **g++** is used

C++

- Main factors differentiating C++ from C:
  - Slightly different syntax, contains type `bool`
  - Functions overloading
  - Object oriented

Hello World++

Hello.cpp

- File extension **.cpp** (C++ uses also .h)
- I/O : <iostream>, <fstream>

```
cin >> ,cout <<, endl
(i/o)fstream()
```

- Automatic casting when reading variables
- Variables can be declared anywhere

```
for( int i=0; i<10; i++ )
```

- `bool` type

```
bool x;
x = true || false;
```

dynmem.cpp

Dynamic Memory Allocation

- New (equivalent of malloc() / calloc())

```
float *arr = new float[7];
```

C →

```
float *arr = (float *) malloc( 7 * sizeof(float) );
```

- Delete (equivalent of free())

```
delete [] arr;
```

C →

```
free( arr );
```

- No realloc() !

templates.cpp

C++ Standard Template Library(STL)

Provides **special C++ "types" (class templates)**.

Anything from the standard library must be preceded by the `std::` prefix
Alternatively, we can put `using namespace std` at the beginning

- **Vector**
  - Array, at declaration must specify type
  - Assignment between whole arrays
  - Functions to determine array size, swap elements, etc.
- List
- Queue
- Stack

⋮

Dynamic memory allocation managed by C++ !

http://www.cplusplus.com/reference/stl/

templates.cpp

Strings

- Enhanced functionalities wrt C string
- Perhaps the most interesting is the use of **+** to concatenate strings
- find_fist_of,() find_last_of(), substr(), etc.
- Dynamic memory allocation managed by C++

http://www.cplusplus.com/reference/string/string/

funOverload.cpp

Functions Overloading

- Use function with same name in different fashions
- Behavior of function depends on:
  - The number of arguments
  - The data type of arguments
  - The order of appearance of arguments
- C++ automatically determines which implementation of the function to use given arguments

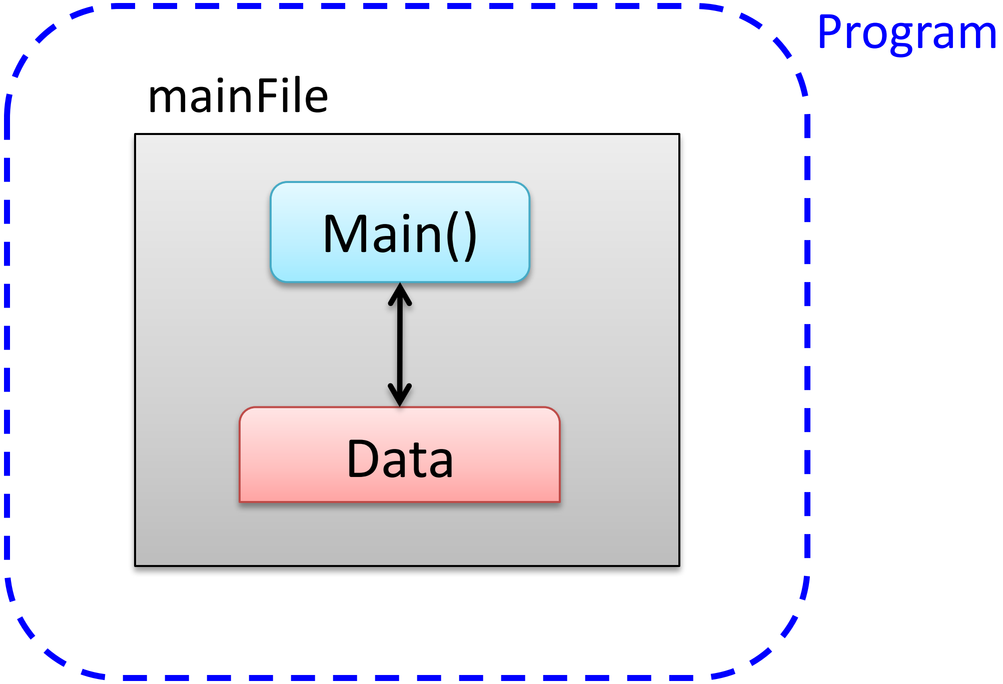
Object Oriented Programming

Programming Paradigms

- Unstructured Programming
- Procedural Programming
- Modular Programming
- Object Oriented Programming

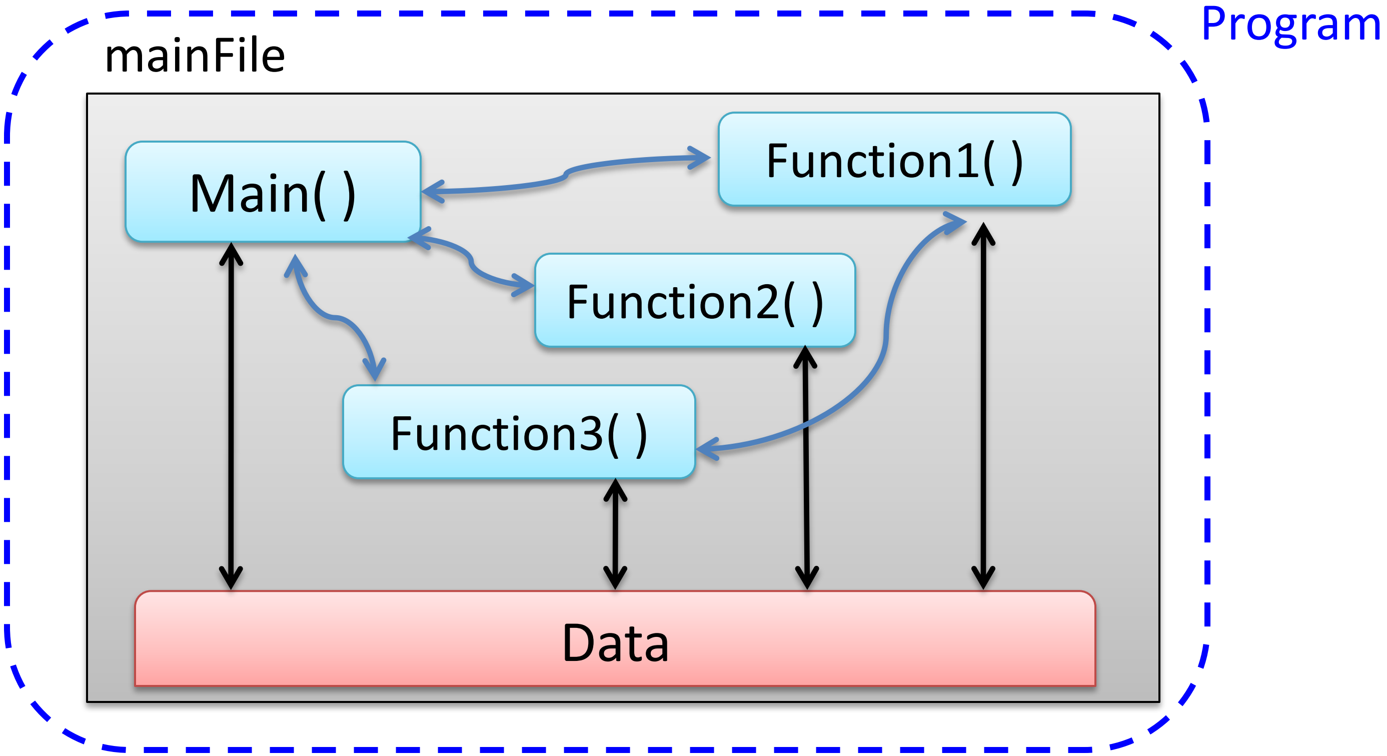
Unstructured Programming

- One single file
- Only one block of code: the main() function
- Data manipulated sequentially inside main()



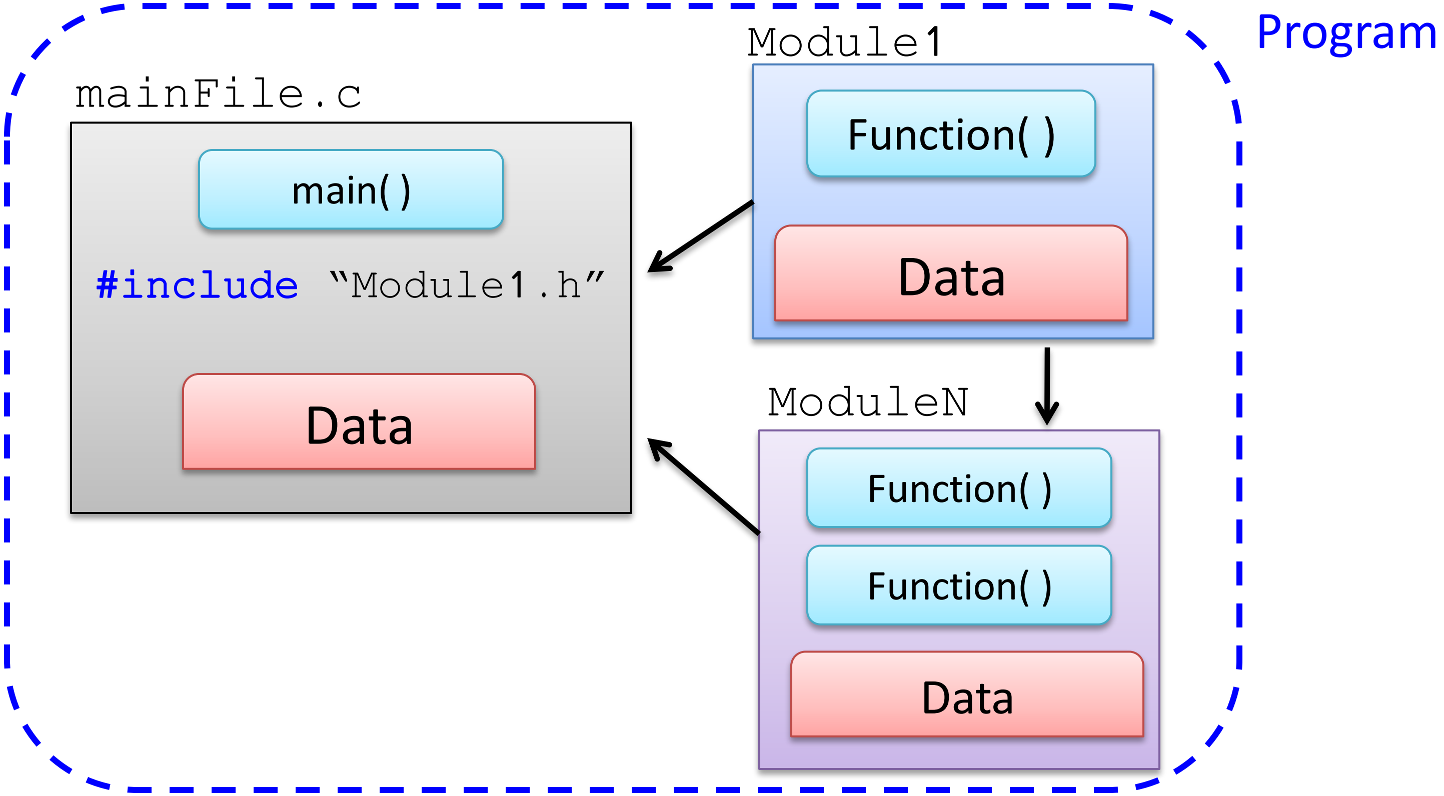
Procedural Programming

- One single file
- Multiple blocks of code grouped in functions (or ***procedures***)
- Data manipulated inside functions ()



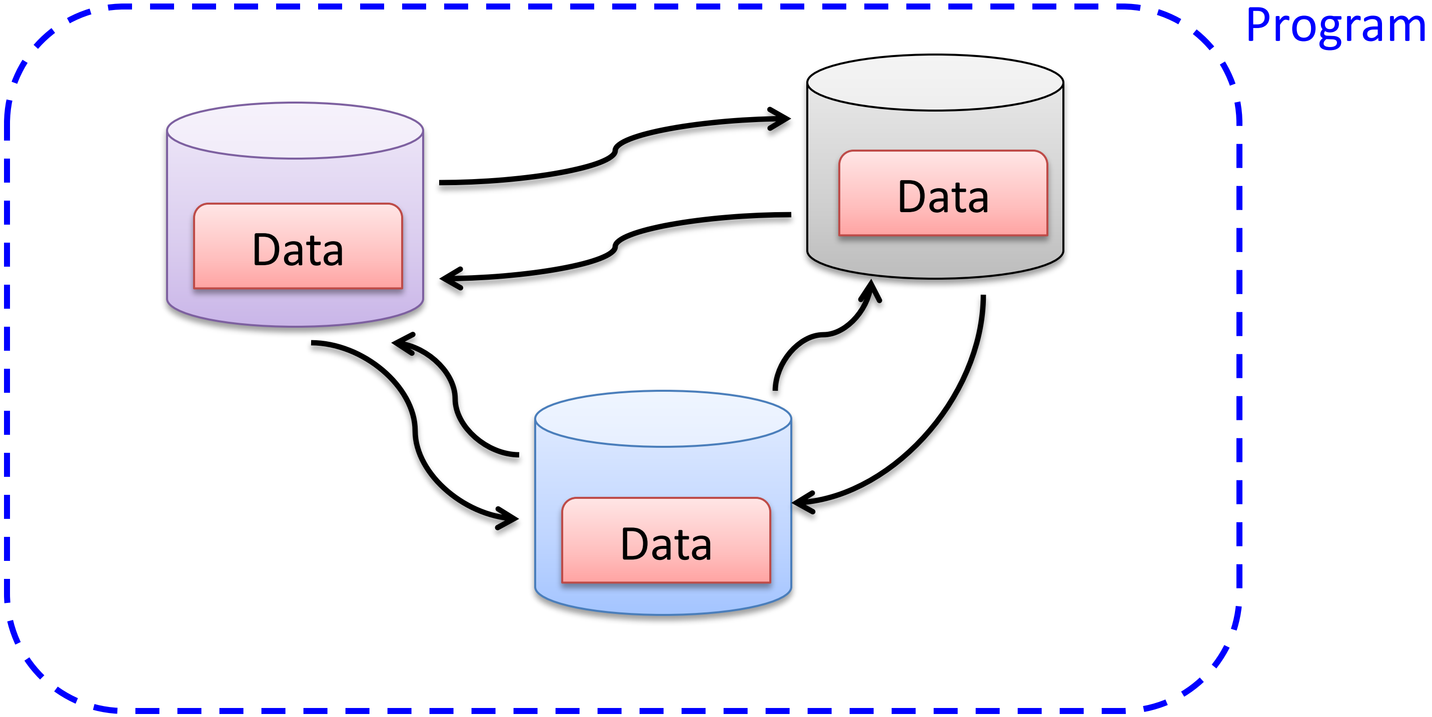
Modular Programming

- Multiple files
- Functions of similar logical goal grouped into ***modules***
- Different data manipulated inside functions in modules



Object Oriented Programming

- Based on ***objects*** interacting with each other
- Objects exchange messages, but maintain their state and data
- Usually associated also with modular programming



Object oriented programming

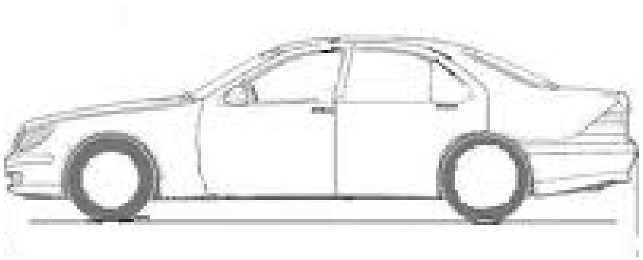
- Classes
- Objects
- Inheritance
- Polymorphism

Objects

- An **object** is an entity, for example a car, a building, a phone…

- An object can be defined by its ***features*** (attributes) or by its ***behavior*** (functions it provides)

- For example a car:
  - has wheels, seats, an engine
  - has make, model, year
  - can run at 100mph, transport people



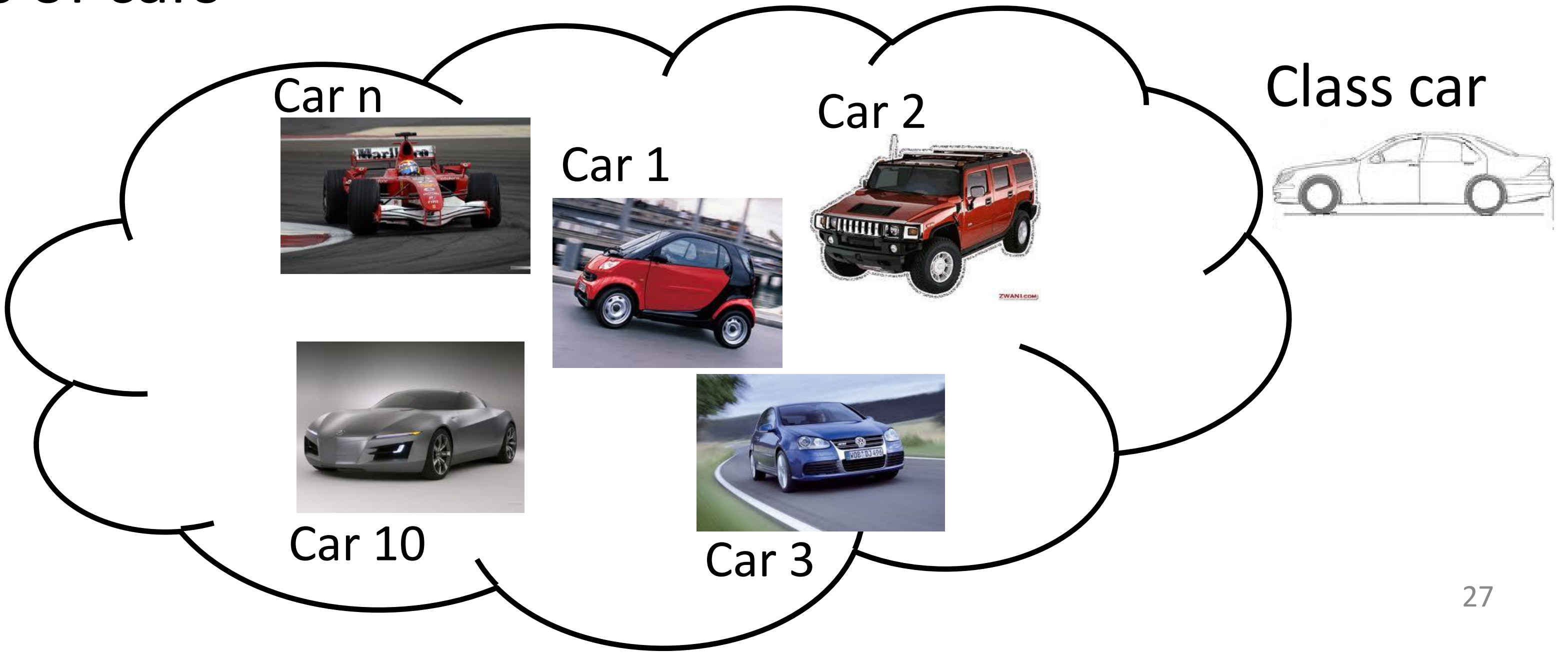
- In programming, we can think of an object as a struct variable, but with enhanced capabilities

Classes

- Defining an object through features and functionalities it too generic, we are defining a class of objects

- An object is a single instance of a **class**

- For example my Ferrari F40, with 120K miles and a scratch on the side is an object, and belongs to the class of cars



Car n

Car 1

Car 2

Class car

Car 10

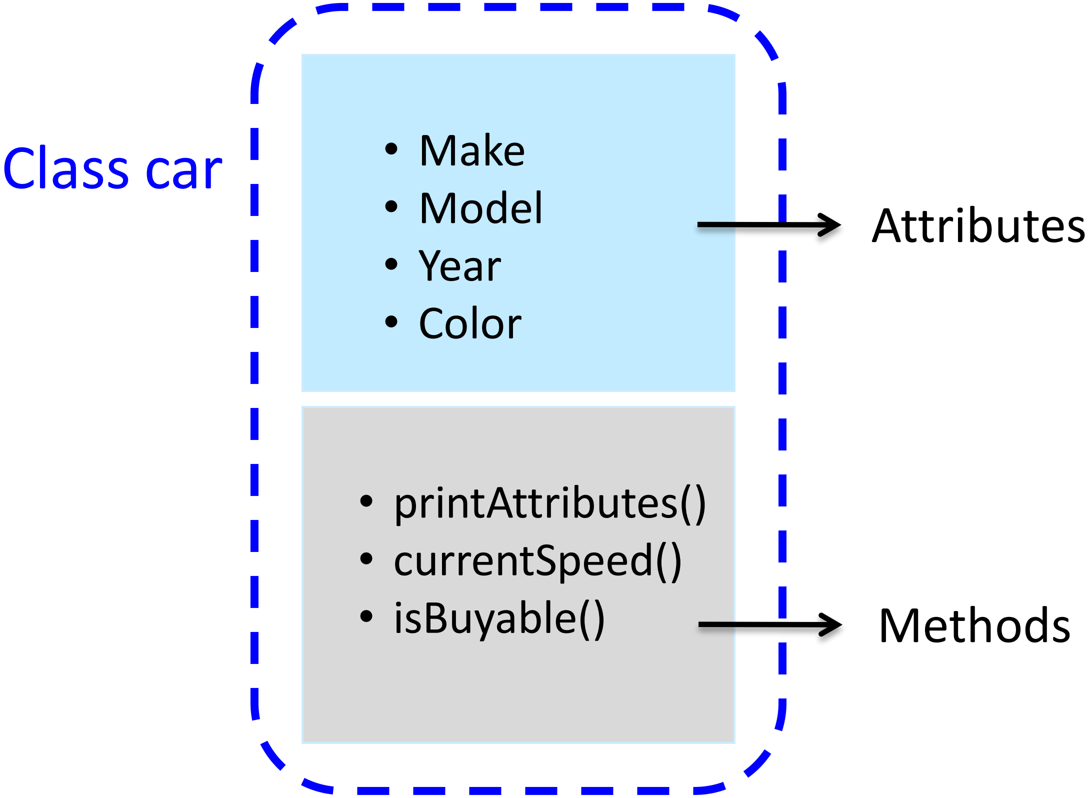
Car 3

Classes

- We can think of classes and objects in terms of familiar C types
- A **class** is an enhanced `struct`
  - `class car`
- An **object** is a variable of type class
  - `car c1`

Attributes and Methods

- We can think of a **class** as a `struct` with enhanced capabilities. A class has
  - Attributes (variables, like the fields in struct)
  - Methods (member functions)



Class car

- Make
- Model
- Year
- Color

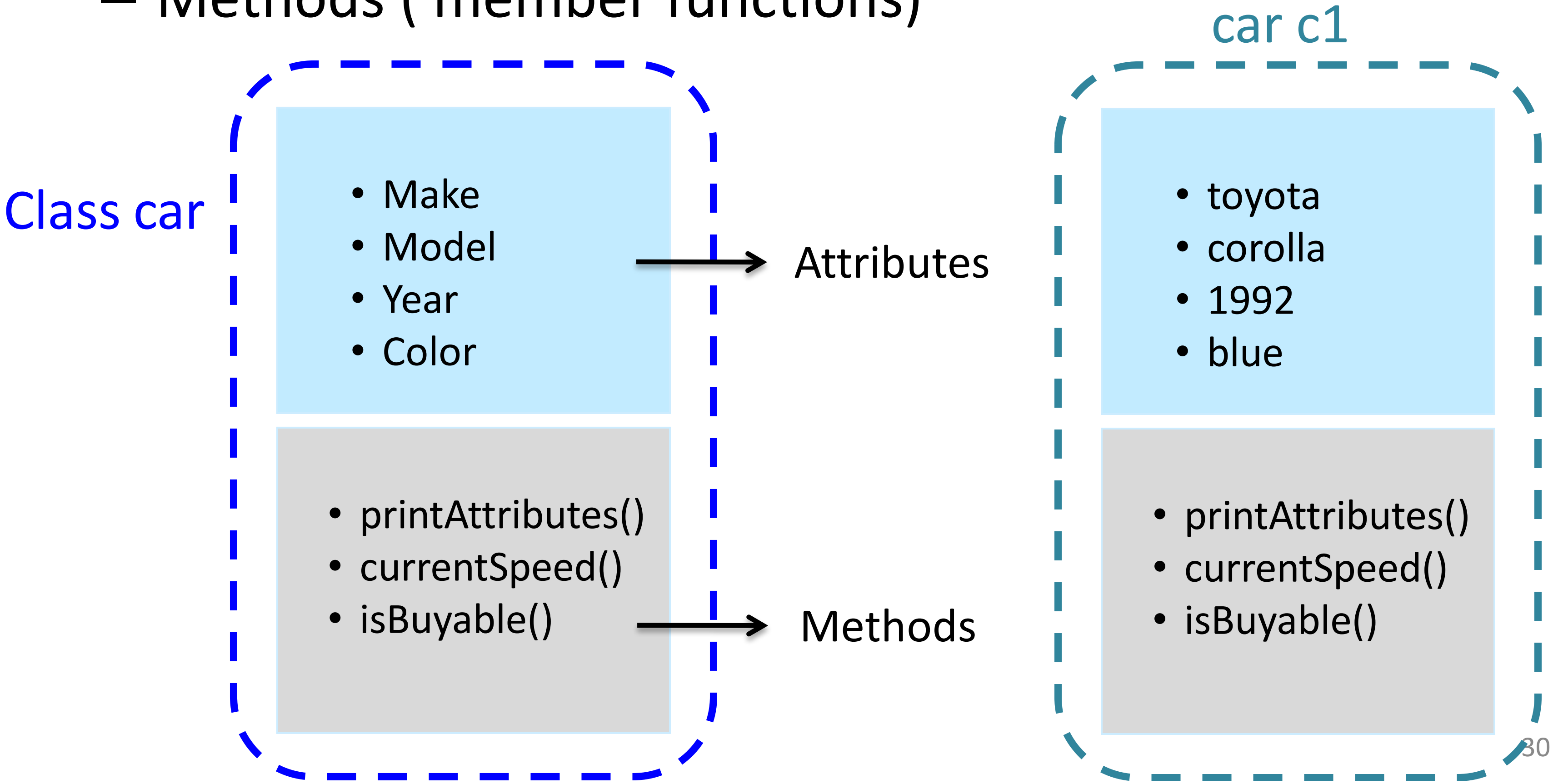
Attributes

- printAttributes()
- currentSpeed()
- isBuyable()

Methods

Attributes and Methods

- We can think of a **class** as a `struct` with enhanced capabilities. A class has
  - Attributes (variables, like the fields in struct)
  - Methods (member functions)



Class car

- Make
- Model
- Year
- Color

- printAttributes()
- currentSpeed()
- isBuyable()

car c1

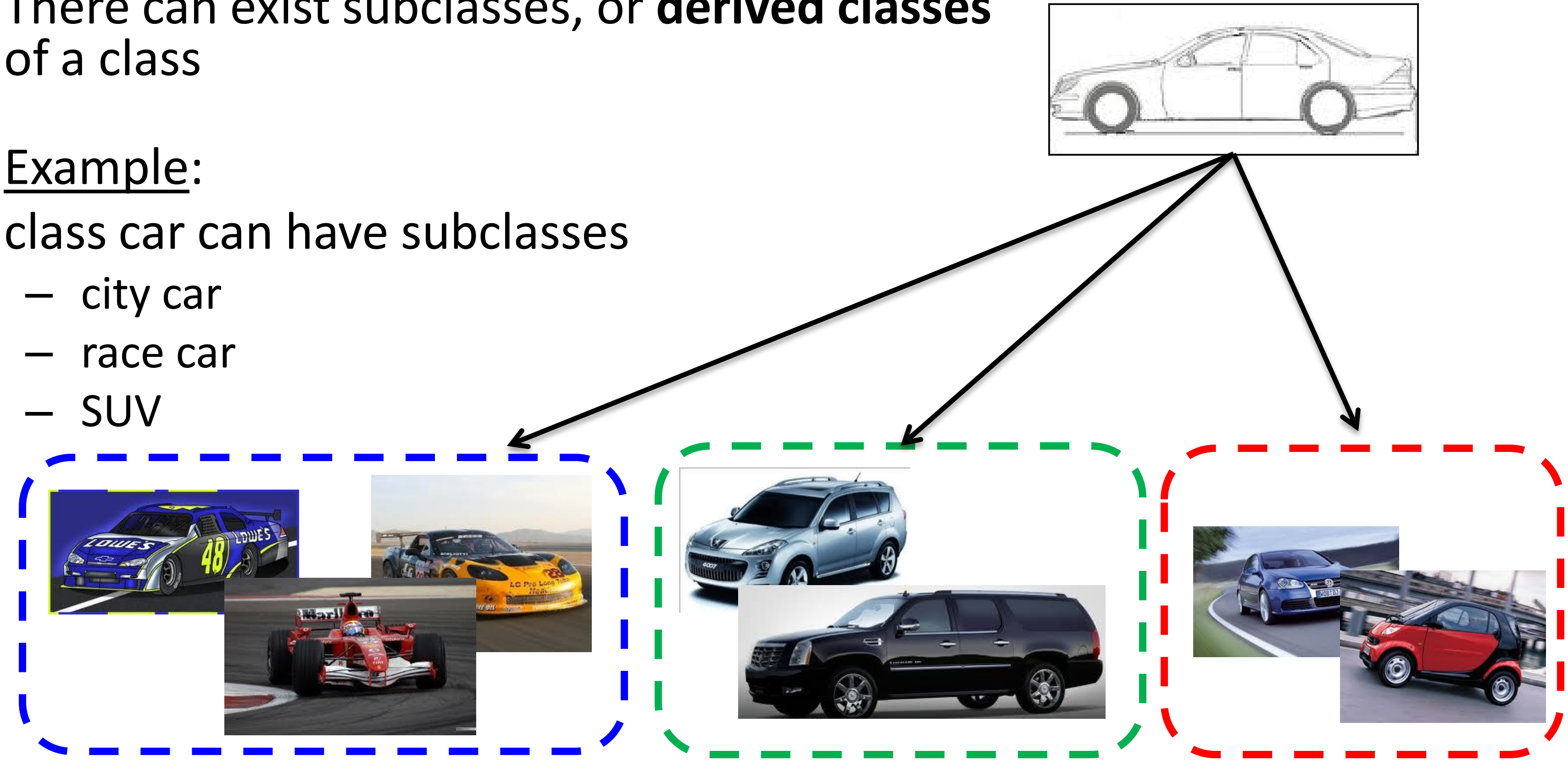
- toyota
- corolla
- 1992
- blue

- printAttributes()
- currentSpeed()
- isBuyable()

Inheritance

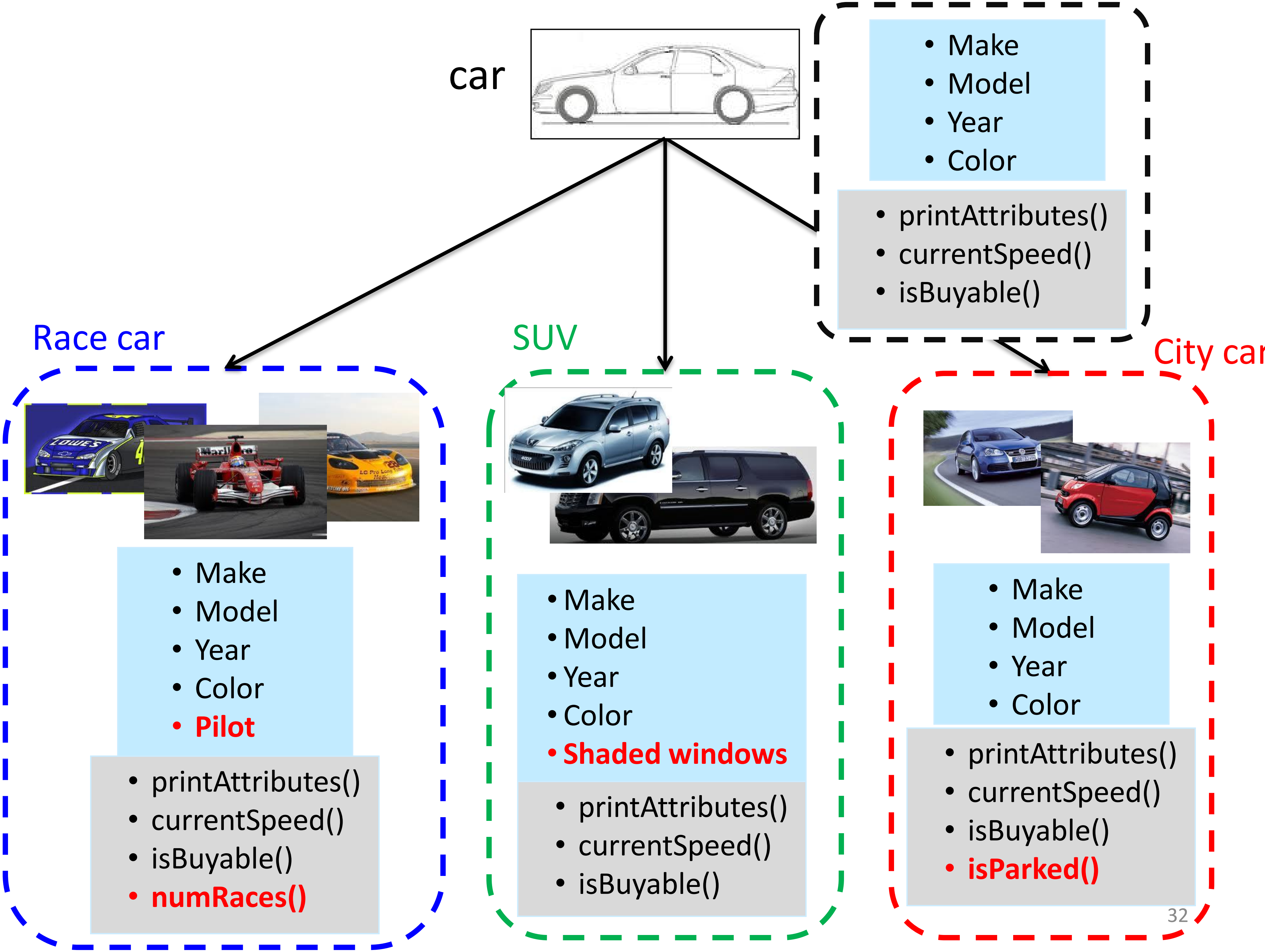
- There can exist subclasses, or **derived classes** of a class

- Example:
  class car can have subclasses
  - city car
  - race car
  - SUV



- All derived classes **inherit** the attributes and methods of the parent class

- They also add their new attributes and/or methods



car

- Make
- Model
- Year
- Color

- printAttributes()
- currentSpeed()
- isBuyable()

Race car

- Make
- Model
- Year
- Color
- **Pilot**

- printAttributes()
- currentSpeed()
- isBuyable()
- **numRaces()**

SUV

- Make
- Model
- Year
- Color
- **Shaded windows**

- printAttributes()
- currentSpeed()
- isBuyable()

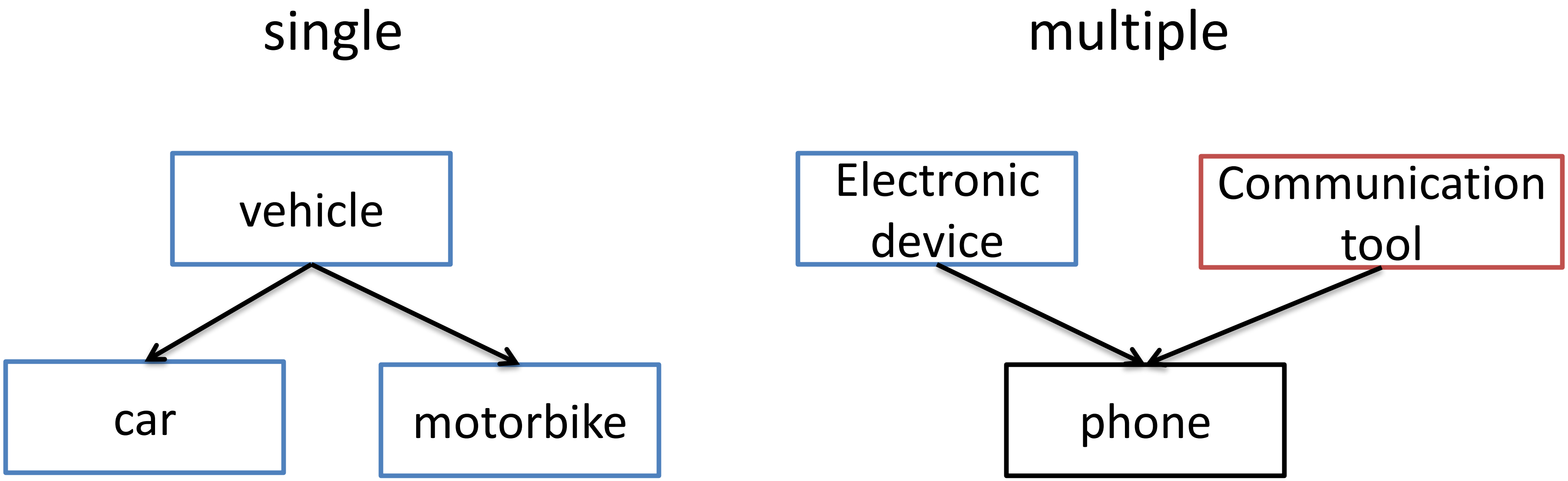
City car

- Make
- Model
- Year
- Color

- printAttributes()
- currentSpeed()
- isBuyable()
- **isParked()**

Inheritance

- Inheritance can be
  - Single : a class derives from **only one** other class
  - Multiple: a class derives from **multiple** classes



single

vehicle

car

motorbike

multiple

Electronic device

Communication tool

phone

Polymorphism

- Different subclasses can have different implementations of a function declared in their parent class

- Example:
  - **printAttributes** ()