Cimple

Graham Barab

Shankara Pailoor

Pancham preet Kaur

Motivation

Struggles of a C programmer

- No **Code-Reuse**, except standard library;
  With *Inheritance*, Cimple to the rescue.

- **Resource Management** - all the `malloc`, `calloc`, `realloc`, `free` take away from the real problem;
  Cimple's approach bi-dimensional - `make` or `clean`.

- **Coding style** – limited scope for compactness, readability, memory-efficient code;
  Cimple gives *Anonymous functions*, unnamed hence need no storage.

Comparison

Features	C	Cimple
Speed	✓	✓
Programming Style	Imperative	Imperative, OO
Library Support	✓	
Pointers	✓	✓
Inheritance		✓
Anonymous functions		✓
Interfaces		✓
Garbage Collection	Manual & Cumbersome	Manual but Convenient

Inheritance

- Inheritance in Cimple modeled after Java
- Inheriting struct 'extends' another struct

```
struct Bicycle {
        int cadence;
        int gear;
        int speed;
        Bicycle(int start_cadence, int start_speed, int start_gear) {
                gear = start_gear;
                cadence = start_cadence;
                speed = start_speed;
        }

        ~Bicycle() {
                printf("Bicycle destructor");
        }
 };
```

```
struct MountainBike extends Bicycle {
        int seatHeight;

         MountainBike(int start_height,
                      int start_cadence,
                      int start_speed,
                      int start_gear) {
                super(start_cadence, start_speed, start_gear);
                seatHeight = start_height;
        }

        ~MountainBike() {
                printf("MountainBike destructor");
        }
};
```

Interfaces

- Allow more flexible inheritance than rigid parent-child hierarchy
- Define a contract for behavior of 'implementing' structs, using Method Sets

Cimple Syntax

```
interface Shape {
        int getArea();
        int getPerimeter();
};
```

Compiled C syntax

```
struct _interfaceShape{
void* body;
int(*getPerimeter)(void*);
int(*getArea)(void*);
}
```

Methods

Describes behavior of structs

Example:

```
struct _virtualSquare{
int(*getArea)(void*);
int(*getPerimeter)(void*);
}
;
```

```
int (Square *s) getArea() {
        return s.x * s.x;
}

int (Square *s) getPerimeter() {
        return 4*s.x;
}
```

```
struct _structSquare{
        struct _interfaceShape _Shape_Square;
        struct _virtualSquare* _virtual;
        int x;
};
```

Anonymous Functions

- Introduced as a measure to make long programs better readable
- Syntax :

Declaration :- func (return-type)(arg_1, arg_2, …) { statements }

Call :-

```
r-type outer-function(arg_1, func(return-type)(arg_1, arg_2, …) *func_pointer, ...){

                    // statements

        func_pointer(arg_1);

                    // rest of the function body
```

Anonymous Functions

```
int plus(int x, func(int)(int) myFunc)
{
    string str = "hello_from_plus_function\n";

    apply_op(str, func()(string s){
                printf("%s", s);
            });

    return 0;
}
```

Anonymous Functions

```
int main(int argc, string **argv) {

        struct Person *graham;

        Struct grahamsName = “Graham Barab”;

        graham = make Person(func(string) { return
grahamsName;});

        printf(“Person’s name is %s”, graham.name);

        return 0;

}
```

Heap Memory Management

- Handled using two keywords - `make` and `clean`
- `make` invokes the constructor, `clean` invokes the destructor

Architecture