ETH zürich

Session 08 Recursion II, Custom Data Types

Informatik I
Lan Zhang
10.04.2025

Schedule

1. Exercise Feedback
2. Theory Recap
    - Recursion
    - Structs & Classes
3. In-Class Code Examples

Exercise Feedback

Reminder: Exercise feedback is by request

- At top of submission add a comment like // FEEDBACK [optionally, any questions]

1. DO NOT USE UNSIGNED INTEGERS

   ```
   for(int i = 0; i < int(vec.size()); ++i) // allowed
   ```

2. A non-const reference (int&) cannot be bound to a const variable
3. Use std::swap(x,y) if allowed

Struct

- Allows the implementation of customized data types
- Groups related data (member variables / functions) for easier management

Declaration:

```
struct Complex{
  double real;
  double imag;
};
```

To initialize variables of type `Complex`:

```
Complex x = {3.0,4.0}; // using initializer list
```

To access elements from the struct:

```
std::cout << x.real << “+” << x.imag << “i” << std::endl;
```

Data Encapsulation

To hide implementation details and provide controlled access through member functions.

Struct: public by default

Class: private by default

Example:

```
struct UserAccount {
        public: \\ accessible member variables
                void validate() const {}
        private: \\ inaccessible member variables except for member functions
                std::string username;
                double password;
};
```

Operator Overloading

- allow us to implement operators for our own data types

```
operator[op]()
```

- return references for (`+=, -=, *=,` ...) to modify the left operand

Example:

```
struct Complex {
    // Overloading the *= operator
    Complex& operator*=(const Complex& other) const{
        return Complex(
            real * other.real - imag * other.imag,
            real * other.imag + imag * other.real
        );
};
```

Operator Overloading: Examples

`a + b`	`T operator+(const T a, const T b);`
`a - b`	`T operator-(const T a, const T b);`
`-a`	`T operator-(T a);`
`a == b`	`bool operator==(const T a, const T b);`
`a += b`	`T& operator+=(T& a, const T b);`
`std::cout << a`	`std::ostream& operator<<(std::ostream &out, const T a);`
`std::cin >> a`	`std::istream& operator>>(std::istream &in, T& a);`

for more info refer to: operator overloading - cppreference.com

Function Overloading

- Allow us to define functions with the same name but different signatures
- Signature: Consists of name, type, number and order of arguments (but not the return value)
- Compiler chooses the function that best matches the call based on provided arguments

Example:

```
int area(int radius){...}
int area(int width, int length){...}

int square(int x){...}
double square(int x){...} // ambiguous!
double square(double x){...} // works
```

In-Class Code Example

PowerSet: Set of all subsets

- $P(\emptyset) = \{\emptyset\}$ → base case ($\emptyset$ subset of any set)
- $P(\{a\}) = \{\emptyset,\{a\}\} = \{\emptyset\}\cup\{\{a\}\}$ → append new element to each existing subset
- $P(\{a,b\}) = \{\emptyset,\{a\},\{b\},\{a,b\}\} = \{\emptyset,\{a\}\} \cup \{\{b\},\{a,b\}\}$

Note:

- $P(\emptyset) \subseteq P(\{a\})$, generally for sets $A \subseteq B \rightarrow P(A) \subseteq P(B)$
- $|P(S)| = 2^{|S|}$

Recursive algorithm for computing the power set of a set S:

1. Select some element $x \in S$, where $|S| = n$
2. Build a new set $S' := S \setminus \{x\}$
3. Compute $P(S')$, where $|S| = n-1$
4. Return $P(S) = P(S') \cup \{Y \cup \{x\} \mid Y \in P(S')\}$

In-Class Code Example

Towers of Hanoi

The game consists of 3 rods, on which disks of different sizes are placed. At the beginning, all disks are placed on left rod, ordered by their size, with the largest disk at the bottom and the smallest at the top.

The aim of the game is to move the entire stack of disks from the left rod to the right rod.

Rules:

1. Only one disk may be moved at a time.
2. The moved disk may not be placed on top of a smaller disk.

In-Class Code Example

```
void move(int n, const std::string& src, const std::string& aux, const
std::string& dst) {
   if (n == 1) {
     // base case ('move' the disc)
     std::cout << src << " --> " << dst << std::endl;
   } else {
     // recursive case
     move(n-1, src, dst, aux);
     move(1, src, aux, dst );
     move(n-1, aux, src, dst);
   }
 }
```