Data types
Functional paradigm

Course: Object Oriented Programming (OOP)
CTU, FS, U12110
Matouš Cejnek

Contents

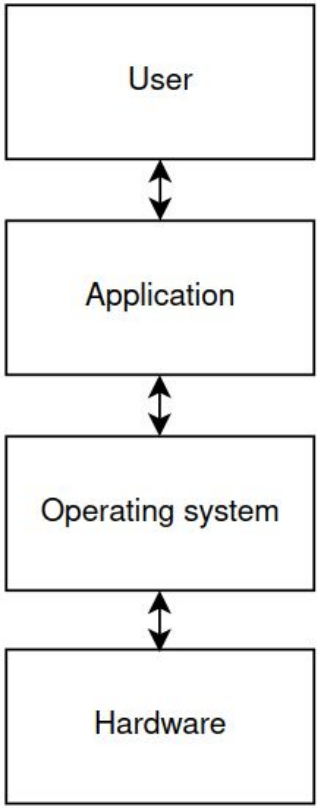
Memory management

Memory management

- Memory management is a form of resource management applied to computer memory.
- Memory management dynamically allocate portions of memory to programs at their request, and free it for reuse when no longer needed.

Memory management

- Memory management can be done at application or operating system level.
- It is complicated especially when more processes exists at the same time.



Memory management types

Manual - programmer specifies memory operations

Automatic - system handles memory management

Manual memory management

- Manual memory management is usage of manual instructions by the programmer to identify and deallocate unused objects (garbage).
- The main manually managed languages still in widespread use today are C and C++.

Automatic memory management

In many programming language implementations, the runtime environment automatically allocates memory in the **call stack** for non-static local variables of a subroutine, called **automatic variables**, when the subroutine is called, and automatically releases that memory when the subroutine is exited.

Automatic variable

- **Automatic variable** is a local variable which is allocated and deallocated automatically when program flow enters and leaves the variable's **scope**.
- **Scope** is a **name binding** in the part of a program where the name binding is valid (scope helps prevent name collisions).

CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Call stack

- **Call stack** is a stack data structure that stores information about the active subroutines of a computer program.
- **Subroutine** is a sequence of program instructions that performs a specific task, packaged as a unit (function, method, etc.)
- Alternative names: execution stack, program stack, control stack, run-time stack, or machine stack.

Data types

A data type, is a specification of a variable and what type of mathematical, relational or logical operations can be applied to it without causing an error.

Data types are implemented differently in various programming languages - different operations, names, value ranges etc.

Primitive data types

Primitive data types are a set of basic data types from which all other data types are constructed. Common primitives are:

- Integer (number)
- Float (number with floating point)
- String (text)
- Boolean

Integer

An integer is the number zero, a positive natural number or a negative integer with a minus sign.

In computer programming, we further categorize integer data types according:

- Only positive (unsigned integers) or all integers
- Minimum and maximum possible value (32bit, 64bit, …)

Integer

Example of integer implementation in some languages:

Bits	Range	Implementations				
		C/C++	C#	Java	FORTRAN	Rust
8	Signed: From −128 to 127	int8_t	sbyte	byte	integer(1)	i8
	Unsigned: From 0 to 255	uint8_t	byte	—	—	u8
16	Signed: From −32,768 to 32,767	int16_t	short	short	integer(2)	i16
	Unsigned: From 0 to 65,535	uint16_t	ushort	char[c]	—	u16
32	Signed: From −2,147,483,648 to 2,147,483,647	int32_t	int	int	integer(4)	i32
	Unsigned: From 0 to 4,294,967,295	uint32_t	uint	—	—	u32
64	Signed: From −9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	int64_t	long	long	integer(8)	i64
	Unsigned: From 0 to 18,446,744,073,709,551,615	uint64_t	ulong	—	—	u64

Float (real)

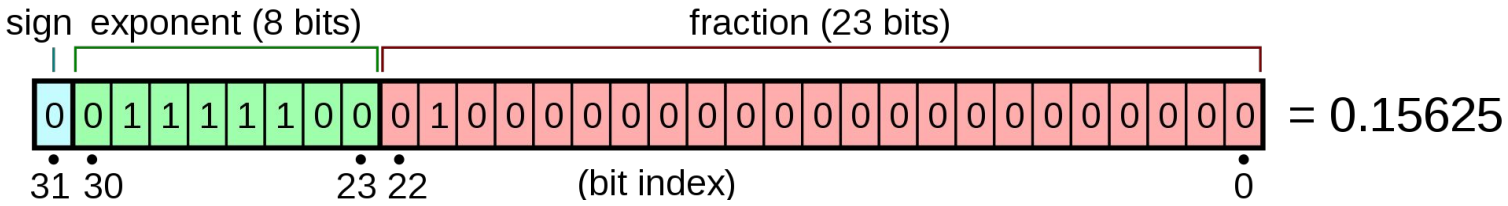
In programming, a floating-point or float is a variable type that is used to store floating-point number values.

The floating point number operations are defined in the IEEE Standard for Floating-Point Arithmetic (IEEE 754) from 1985. Many languages follow the standard at least partially.

Float

The number is represented as $\texttt{sign } 2^{\texttt{exponent}} \texttt{ * fraction}$

IEEE 754 implementation (32 bits version):



String

- String is a sequence of characters (unformatted text).
- Common implementation is an array of integers (encoded characters).

String

Programming languages contain various string functions to manipulate a string or query information about a string. Typical string operations:

- Query / indexing / slicing: `a[1:3]`
- Comparison `a == b`
- Concatenation `a + b`
- Find / search
- Replace
- Change of case

CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Boolean

- A boolean or bool is a data type with two possible values (true / false, or 0 / 1, etc.).
- Common operations with booleans are: AND, OR, NOT, NOR, NAND, and XOR.
- The Boolean data type is primarily associated with **conditional statements** (if conditions and similar).

Less common data types

Those data types are often provided in popular languages in form of a standard or a non standard library:

- Complex numbers
- Matrix
- Datetime, time
- …

Container/collection data types

Containers (or collections) are more complex data types

- Data types to store sequences and mappings
- They can have a form of list, array, dictionary, set, etc.
- Different collections provide different operations

Collections

- Collections may force same data type of items
- Collections may allow nesting
- Collections may be mutable
- Various languages feature various collections with various features

Collection examples

C# (strongly typed list of objects)

```
var colors = new List<string>();
colors.Add("red");
colors.Add("black");
colors.Add("gray");
```

Collection examples

Python list (with some nested content)

```
random_stuff = [[1, 2 ,3], True, 'abc', 0.5]
```

Collection examples

Python dictionary

```
ages = {
    'Alice': 30,
    'Bob': 24,
    'Chris': 58
}
```

Collection examples

Some languages features set like collections.

Python set

```
a = {1, 3, 5, 6}
b = {3, 6, 8}
print(a.union(b))
print(a.difference(b))
print(b.difference(a))
```

Output:

```
{1, 3, 5, 6, 8}
{1, 5}
{8}
```

Class as a custom data type

- In OOP languages, user can create own data type as a class.
- Such a class can store values of various types and structures and can provide all necessary functionality in its own methods.
- Some languages provide specific classes prepared for this purpose - for example: data classes in Python, records in C#.

Functions

Functions

In mathematics, a function from a set X to a set Y assigns to each element of X exactly one element of Y.

In computer science, a function is an encapsulation of a task.

In other words, a function is a block of organized, reusable code.

Functions are not designed to store data.

Function vs method

- A function is a set of instructions or procedures to perform a specific task
- A method is a set of instructions that are associated with an object.

Functions

Generally in all common paradigms:

- Functions may return output
- Functions may accept input
- Functions may be nested
- Functions may return function
- Output and Input types of functions may be restricted

Function elements

Functions is commonly defined by:

- Function name (how the functions is **call**ed)
- Input **parameter**s / **argument**s
- Output that function **return**s

1. Function is **call**ed by **name**
2. **Parameter**s are provided during **call**
3. Function **return** output

Function examples

Function example (y = 2 * x):

Python

```
def y(x):
    return 2 * x

print(y(3))
```

Ruby

```
def y(x)
   y = 2 * x
End

puts(y(3))
```

Function examples

General form in C language:

```
return_type function_name( parameter list ) {
   body of the function

   return [expression];
}
```

Recursive function

- **Recursion** is a method of solving a computational problem where the solution depends on solutions to smaller instances of the same problem
- **Recursive functions** is breaking down complex inputs into simpler ones, with each recursive call.
- The input problem must be simplified in such a way that eventually the base case must be reached.

Recursive function example

Solution: 5! = 5 * f(4) = 5 * (4 * f(3)) = 5 * (4 * (3 * f(2))) = 5 * (4 * (3 * (2 * f(1)))) = 5 * (4 * (3 * (2 * 1)))

```
def factorial_recursive(n):
    if n == 1:
        return 1
    else:
        return n * factorial_recursive(n - 1)
```

Recursive solution of factorial

```
def factorial_iterative(n):
    result = 1
    for x in range(n):
        result *= x + 1
    return result
```

Iterative solution of factorial

Functional paradigm

CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Functional programming paradigm

- Lambda calculus (1930, Alonzo Church)
- The first high-level functional programming language, LISP, was developed in the late 1950s
- Functional programming is still well used today (for example language Haskell)
- Many languages include functional programming (Python, …)

CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Functional vs procedural example (Python)

Conventional imperative loop:

```
result = 0
for item in [1, 2, 3, 5, 8]:
    if not item % 2:
        result += item * 10
```

Functional Programming with higher-order functions:

```
result = sum(map(lambda x: x * 10 if not x % 2 else 0, [1, 2, 3, 5, 8]))
```

Alternative with list comprehension:

```
result = sum([x * 10 for x in [1, 2, 3, 5, 8] if not x % 2])
```

Bonus content

Ternary (conditional) operator

In computer science, a ternary operator is an operator that takes three arguments.

In many programming languages, it is used for conditional expressions.

In some languages, this operator is directly referred as conditional operator.

Ternary operator examples

Python

```
result = a + b if a > b else a - b
```

Java

```
result = (a>b) ? (a+b):(a-b)
```

CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

List comprehension

A list comprehension is a syntactic construct available in some programming languages for creating a list based on existing iterables.

Python example:

```
old_list = [1, 3, 4, 10]
print([item * 2 for item in old_list])
```

Haskell example:

```
main = do
let old_list = [1, 2, 4, 10]
print([(x) | x <- old_list])
```

Generators

- An **iterator** is an object that enables a programmer to traverse a container, particularly lists.
- A **generator** is a routine that can be used to control the iteration behaviour of a loop. All generators are also iterators.
- Generators build output on the fly (unlike list comprehension).
- Generators do not store all iterations in memory.

Generator examples

Generator that returns increasing integers from 1.

Python (since v2.2)

```
def my_generator():
    n = 0
    while True:
        n += 1
        yield n
```

PHP (since v5.5)

```
function my_generator()
{
    $n = 0;
    while (true) {
        $n += 1;
        yield $n;
    }
}
```

CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Containers vs generators

Feature	Container	Generator
Slicing, indexing	Yes	No
Preparation necessary	Yes	No
Require memory	Yes	No
Measureable	Yes	No