Data Structure and Algorithms [CO2003]

Chapter 1 - Introduction

Lecturer: Duc Dung Nguyen, PhD.
Contact: nddung@hcmut.edu.vn

Faculty of Computer Science and Engineering
Hochiminh city University of Technology

Contents

Basic concepts

What is Data?

(Source: datorama.com)

Data

Data is information that has been translated into a form that is more convenient to calculate, analyze.

Example

- Numbers, words, measurements, observations or descriptions of things.

- Qualitative data: descriptive information,
- Quantitative data: numerical information (numbers).
  - Discrete data can only take certain values (like whole numbers)
  - Continuous data can take any value (within a range)

Class of data objects that have the same properties.

Data type

1. A set of values
2. A set of operations on values

Example

Type	Values	Operations
integer	$-\infty, ..., -2, -1, 0, 1, 2, ..., \infty$	$*, +, -, \%, /, ++, --, ...$
floating point	$-\infty, ..., 0.0, ..., \infty$	$*, +, -, /, ...$
character	$\backslash 0$, ..., 'A', 'B', ..., 'a', 'b', ..., $\sim$	$<, >, ...$

What is a data structure?

1. A combination of elements in which each is either a data type or another data structure
2. A set of associations or relationships (structure) that holds the data together

Example

An array is a number of elements of the same type in a specific order.

1	2	3	5	8	13	21	34

The concept of abstraction:

- Users know what a data type can do.
- How it is done is hidden.

Definition
An **abstract data type** is a data declaration packaged together with the operations that are meaningful for the data type.

1. Declaration of data
2. Declaration of operations
3. Encapsulation of data and operations

Abstract data type

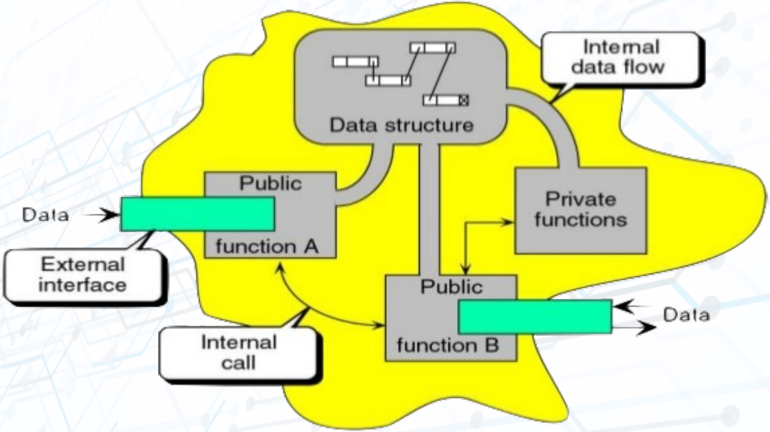


Figure 1: Abstract data type model (source: Slideshare)

Interface

- **Data**: sequence of elements of a particular data type
- **Operations**: accessing, insertion, deletion

Implementation

- Array
- Linked list

What is an algorithm?
The logical steps to solve a problem.

What is a program?
Program = Data structures + Algorithms (Niklaus Wirth)

Pseudocode

- The most common tool to define algorithms
- English-like representation of the algorithm logic
- Pseudocode = **English** + **code**
  - English: relaxed syntax being easy to read
  - Code: instructions using basic control structures (sequential, conditional, iterative)

Pseudocode

Algorithm Header

- Name, Parameters and their types
- Purpose: what the algorithm does
- Precondition: precursor requirements for the parameters
- Postcondition: taken action and status of the parameters
- Return condition: returned value

Algorithm Body

- Statements
- Statement numbers: decimal notation to express levels
- Variables: important data
- Algorithm analysis: comments to explain salient points
- Statement constructs: sequence, selection, iteration

```
Algorithm average
Pre nothing
Post the average of the input numbers is printed
i = 0
sum = 0
while all numbers not read do
    i = i + 1
    read number
    sum = sum + number
end
average = sum / i
print average
End average
```

Algorithm 1: How to calculate the average

Revision

Data structures

Data structures can be declared in C++ using the following syntax:

```
struct [type_name] {
   member_type1 member_name1;
   member_type2 member_name2;
   member_type3 member_name3;
   ...
} [object_names];
```

- Where `type_name` is a name for the structure type, `object_names` can be a set of valid identifiers for objects that have the type of this structure.
- Within braces { }, there is a list with the data members, each one is specified with a type and a valid identifier as its name.
- **struct** requires either a `type_name` or at least one name in `object_names`, but not necessarily both.

Example

```
struct car_t {
    int year;
    string brand;
};

car_t toyota;
car_t mercedes, bmw;
```

Example

```
struct {
    int year;
    string brand;
} toyota, mercedes, bmw;
```

A member of an object can be accessed directly by a dot (.) inserted between the object name and the member name.

Example

```
toyota.year
toyota.brand
mercedes.year
mercedes.brand
bmw.year
bmw.brand
```

- `toyota.year`, `mercedes.year`, and `bmw.year` are of type `int`.
- `toyota.brand`, `mercedes.brand`, and `bmw.brand` are of type `string`.

Example

```
// example about structures
#include <iostream>

using namespace std;

struct car_t {
  int year;
  string brand;
} mycar;

int main () {
  mycar.brand = "Audi";
  mycar.year = 2011;
  cout << "My␣favorite␣car␣is:" << endl;
  cout << mycar.brand << "␣(" << mycar.year << ")";
  return 0;
}
```

Example

```
#include <iostream>
using namespace std;

struct car_t {
  int year;
  string brand;
} mycar;
void printcar(car_t);

int main () {
  mycar.brand = "Audi";
  mycar.year = 2011;
  printcar(mycar);
  return 0;
}
void printcar(car_t c) {
  cout << "My favorite car is:" << endl;
  cout << c.brand << " (" << c.year << ")";
}
```

Exercise

- Define a data structure `student_t` containing a student's name, firstname and age.
- Write a code in C++ to take input your data and display it.

Classes

Classes are defined using keyword `class`, with the following syntax:

```
class class_name {
  access_specifier_1: member1;
  access_specifier_2: member2;
  ...
} object_names;
```

- Where `class_name` is a valid identifier for the class, `object_names` is an optional list of names for objects of this class.
- The body of the declaration can contain `members`, which can either be data or function declarations, and optionally `access_specifiers`.

Example

```
class Rectangle {
    int width, height;
  public:
    void set_values (int,int);
    int area (void);
} rect;
```

Example

```
#include <iostream>
using namespace std;
class Rectangle {
        int width, height;
    public:
        void set_values (int,int);
        int area (void);
};

void Rectangle::set_values (int x, int y) {
    width = x;
    height = y;
}

int Rectangle::area () {
    return width*height;
}

int main () {
    Rectangle rectA, rectB;
    rectA.set_values (3,4);
    rectB.set_values (5,6);
    cout << "rectA area: " << rectA.area() << endl;
    cout << "rectB area: " << rectB.area() << endl;
    return 0;
}
```

Constructors

- Automatically called whenever a new object of a class is created.
- Initializing member variables or allocate storage of the object.
- Declared with a name that matches the class name and without any return type; not even `void`.

Example

```
class Rectangle {
    int width, height;
  public:
    Rectangle (int,int);
    int area (void);
};
```

Example

```
#include <iostream>
using namespace std;
class Rectangle {
        int width, height;
    public:
        Rectangle (int,int);
        int area (void);
};

Rectangle::Rectangle (int x, int y) {
    width = x;
    height = y;
}

int Rectangle::area () {
    return width*height;
}

int main () {
    Rectangle rectA (3,4);
    Rectangle rectB (5,6);
    cout << "rectA area: " << rectA.area() << endl;
    cout << "rectB area: " << rectB.area() << endl;
    return 0;
}
```

Initialization

- Member initialization:

```
class Rectangle {
   int width;
   const int height;
  public:
   Rectangle(int, int);
   ...
};
Rectangle(int x, int y) : height(y) {
   width = x;
}

int main() {
   Rectangle rectA(3,4);
   ...
}
```

Definition
A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location.

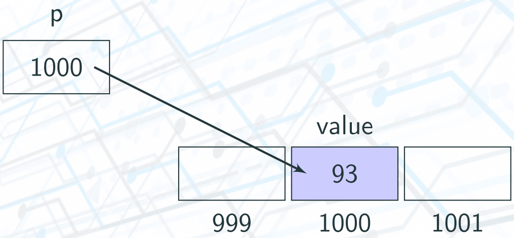
Address-of operator (&)
The address of a variable can be obtained by preceding the name of a variable with an ampersand sign (**&**), known as `address-of operator`. For example:

```
p = &value;
```

Dereference operator (*)
To access the variable pointed to by a pointer, we precede the pointer name with the `dereference operator` (*****).

```
value = *p;
```

Pointers




```
p = &value;
value = *p;
```

Example

```
int main () {
  int v1 = 5, v2 = 15;
  int * p1, * p2;
  p1 = &v1;
  p2 = &v2;
  *p1 = 10;
  *p2 = *p1;
  p1 = p2;
  *p1 = 20;
  cout << "v1 = " << v1 << '\n';
  cout << "v2 = " << v2 << '\n';
  return 0;
}
```

Exercise

What is the output?

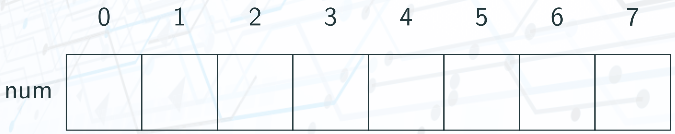
Definition

An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by a unique identifier with an index.

```
type var_name[number_of_elements];
```

Example

```
int num[8];
```



Initializing arrays

```
int num[8];
int num[8] = { };
int num[8] = { 1, 2, 3, 5, 8, 13, 21, 34 };
int num[8] = { 1, 2, 3, 5, 8 };
int num[] = { 1, 2, 3, 5, 8, 13, 21, 34 };
int num[] { 1, 2, 3, 5, 8, 13, 21, 34 };
```

Exercise

For each declaration of `num`, what is the output?

```
for (int i=0; i<8; i++) {
    cout << num[i] << endl;
}
```

Pointers and arrays

The concept of arrays is related to that of pointers. Arrays work very much like pointers to their first elements, and, actually, an array can always be implicitly converted to the pointer of the proper type.

For example, consider these two declarations:

```
int myarray [10];
int * mypointer;
```

The following assignment operation would be valid:

```
mypointer = myarray;
```

Example

```
#include <iostream>
using namespace std;
int main () {
  int num[5];
  int * p;
  p = num;   *p = 1;
  p++;  *p = 2;
  p = &num[2];   *p = 3;
  p = num + 3;   *p = 5;
  p = num;   *(p+4) = 8;
  for (int n=0; n<5; n++)
    cout << num[n] << ", ";
  return 0;
}
```

Exercise

What is the output? Explain.

Structures can be pointed to by its own type of pointers:

```
struct car_t {
  string brand;
  int year;
};
car_t mycar;
car_t * pcar;
```

- mycar is an object of structure type car_t.
- pcar is a pointer to point to an object of structure type car_t.

The following code is valid:

```
pcar = &mycar;
```

The value of the pointer pcar would be assigned the address of object mycar.

arrow operator (->)
The arrow operator (->) is a dereference operator that is used exclusively with pointers to objects that have members. This operator serves to access the member of an object directly from its address.

```
pcar->year
```

Difference:

- Two expressions pcar->year and (*pcar).year are equivalent, and both access the member year of the data structure pointed by a pointer called pcar.
- Two expressions *mycar.year or *(mycar.year) are equivalent. This would access the value pointed by a hypothetical pointer member called year of the structure object mycar (which is not the case, since year is not a pointer type).

Combinations of the operators for pointers and for structure members:

Expression	Equivalent	What is evaluated
a.b		Member b of object a
pa->b	(*pa).b	Member b of object pointed to by pa
*a.b	*(a.b)	Value pointed to by member b of object a

Exercise

- Define a data structure `student_t` containing a student's name, firstname and age.
- Write a code in C++ using pointers to structures to take input your data and display it.

Structures can also be nested in such a way that an element of a structure is itself another structure:

Example

```
struct car_t {
  string brand;
  int year;
};

struct friends_t {
  string name;
  string email;
  car_t favorite_car;
} bobby, tommy;

friends_t *pfriend = &bobby;
```

After the previous declarations, all of the following expressions would be valid:

Example

```
tommy.name
tommy.email
tommy.favorite_car.brand
tommy.favorite_car.year


bobby.name | pfriend->name
bobby.email | pfriend->email
bobby.favorite_car.brand | pfriend->favorite_car.brand
bobby.favorite_car.year | pfriend->favorite_car.year
```

Example

```cpp
#include <iostream>
using namespace std;
class Rectangle {
    int width, height;
  public:
    Rectangle(int x, int y) : width(x), height(y) {}
    int area(void) { return width * height; }
};
int main () {
  Rectangle rectA (3, 4);
  Rectangle * rectB = &rectA;
  Rectangle * rectC = new Rectangle (5, 6);
  cout << "rectA area: " << rectA.area() << endl;
  cout << "rectB area: " << rectB->area() << endl;
  cout << "rectC area: " << rectC->area() << endl;
  delete rectB;
  delete rectC;
  return 0;
}
```