UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

C++ Algorithmen und die Standard Template Library (STL)

Samuel Hooper

03.11.2023

C++ Algorithmen und die Standard Template Library (STL)

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- Was ist ein Algorithmus?
    - Grundlegende Bausteine jeder Softwareanwendung
    - Intelligente, schrittweise Anweisungen, die einen bestimmten Satz von Problemen lösen
- C++ bietet eine umfangreiche Sammlung von vordefinierten Algorithmen in Form der STL, der Standard Template Library
    - STL bietet Containerklassen und Algorithmen, die die Arbeit mit Datenstrukturen erheblich erleichtern (z.B. Vektoren, Listen, Stacks, Queues oder Sortieralgorithmen)

Struktur

I. Die STL und ihre Komponenten
II. Die Algorithmus-Bibliothek
III. Suchalgorithmen
IV. Sortieralgorithmen
V. Stringalgorithmen
VI. Algorithmen für numerische Berechnungen
VII. Effizienz und Komplexität
VIII. Quellen

I. Die STL und ihre Komponenten

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- Sammlung von vorgefertigten Klassen und Funktionen in C++, die bei der Verwaltung von Datenstrukturen und Implementierung von Algorithmen unterstützen
- Stellt bewährte Strukturen für häufig auftretende Aufgaben bereit
- Wichtigste Containerklassen:
    - Vektoren sind dynamische Arrays, die es uns ermöglichen, Listen von Elementen effizient zu speichern und zu verwalten

```
std::vector<int> numbers = {1, 2, 3, 4, 5};
```

    - Listen sind doppelt verkettete Listen, die schnelle Einfüge- und Löschoperationen in der Mitte der Liste ermöglichen

```
std::list<std::string> names = {"Alice", "Bob", "Charlie"};
```

I. Die STL und ihre Komponenten

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- Eine Warteschlange ist eine abstrakte Datenstruktur, die nach dem Prinzip "First-In-First-Out" (FIFO) funktioniert

```
#include <iostream>
#include <queue>

int main() {
    // Erstellen einer leeren Warteschlange für ganze Zahlen
    std::queue<int> myQueue;

    // Elemente zur Warteschlange hinzufügen
    myQueue.push(5);
    myQueue.push(10);
    myQueue.push(15);

    // Die Vorderseite (vorderstes Element) der Warteschlange abrufen und aus
    std::cout << "Vorderstes Element: " << myQueue.front() << std::endl;
```

I. Die STL und ihre Komponenten

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- Ein Stapel arbeitet nach dem Prinzip "Last-In-First-Out" (LIFO)

```
#include <iostream>
#include <stack>

int main() {
    // Erstellen eines leeren Stapels für ganze Zahlen
    std::stack<int> myStack;

    // Elemente auf den Stapel legen (push)
    myStack.push(5);
    myStack.push(10);
    myStack.push(15);

    // Die oberste Position (oberstes Element) des Stapels abrufen und aus
    std::cout << "Oberstes Element: " << myStack.top() << std::endl;
```

II. Algorithmus-Bibliothek

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- Algorithmus-Bibliothek in C++ (Bestandteil der STL) stellt viele Algorithmen zur Verfügung, die auf einer Vielzahl von Datenstrukturen angewendet werden können
- Kategorien:
    - Suchalgorithmen wie `find`, `count`, `binary_search` und `lower_bound`, zum Finden und Analysieren von Elementen in Containern
    - Sortieralgorithmen wie `sort`, `partial_sort` und `nth_element`
    - Verarbeitungsalgorithmen wie `for_each`, `transform` und `accumulate` können Operationen auf Elementen eines Containers ausführen
    - Bereichsalgorithmen wie `copy`, `merge` und `unique` ermöglichen es, bestimmte Elemente aus einem Bereich in einen anderen zu kopieren oder zu transformieren
    - Zahlenalgorithmen wie `min`, `max`, `gcd` und `lcm`, mit denen numerische Berechnungen durchgeführt werden können

II. Algorithmus-Bibliothek

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Um die Algorithmus-Bibliothek in Ihren C++-Programmen zu verwenden, muss man die entsprechenden Header-Dateien inkludieren und die gewünschten Funktionen aufrufen

```
#include <iostream>
```

III. Suchalgorithmen

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

• Der `find`-Algorithmus wird verwendet, um das erste Vorkommen eines Elements in einem Container zu finden. Wenn das Element gefunden wird, gibt er einen Iterator darauf zurück, andernfalls wird das End-Iterator des Containers zurückgegeben.

```
std::vector<int> numbers = {10, 20, 30, 40, 50};
int target = 30;

auto it = std::find(numbers.begin(), numbers.end(), target);
```

• Der `binary_search`-Algorithmus wird auf sortierten Containern angewendet und überprüft, ob ein bestimmtes Element in einem Container vorhanden ist. Er verwendet eine effiziente binäre Suche, um das Element zu finden.

• Der `count`-Algorithmus wird verwendet, um die Anzahl der Vorkommen eines Elements in einem Container zu zählen.

IV. Sortieralgorithmen

- Der `sort`-Algorithmus ist einer der am häufigsten verwendeten Sortieralgorithmen. Er sortiert die Elemente in aufsteigender Reihenfolge und verwendet den *IntroSort*-Ansatz, um die Effizienz sicherzustellen.

```
std::vector<int> numbers = {45, 10, 30, 20, 5};

std::sort(numbers.begin(), numbers.end());
```

- Der `stable_sort`-Algorithmus ist ähnlich wie `sort`, mit dem Unterschied, dass er die relative Reihenfolge der Elemente beibehält, wenn man denselben *Schlüssel* hat. Das macht ihn besonders nützlich, wenn man auf mehreren Schlüsseln sortieren muss.
- Der `partial_sort`-Algorithmus sortiert die ersten Elemente eines Containers in aufsteigender Reihenfolge. Dies kann hilfreich sein, wenn man nur an den Top-N-Werten interessiert sind.

V. Stringalgorithmen

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- Die `find`-Funktion kann verwendet werden, um nach bestimmten Zeichenketten innerhalb eines Texts zu suchen.

```
std::string text = "Dies ist ein Beispieltext.";
std::size_t position = text.find("Beispiel");
```

- Der Algorithmus `replace` erlaubt es, Teile eines Textes durch andere Zeichenfolgen zu ersetzen.

```
std::string text = "Dies ist ein Beispieltext.";
std::string new_text = text;
std::string to_replace = "Beispiel";
std::string replacement = "Demo";
std::size_t position = new_text.find(to_replace);
if (position != std::string::npos) {
    new_text.replace(position, to_replace.length(), replacement);
}
```

V. Stringalgorithmen

- Mit Stringalgorithmen kann ein Text in sogenannte *Tokens* zerlegt werden, basierend auf Trennzeichen.

```
std::string text = "Apfel,Birne,Kirsche,Orange";
std::vector<std::string> tokens;
std::istringstream token_stream(text);
std::string token;
while (std::getline(token_stream, token, ',')) {
    tokens.push_back(token);
}
```

- Ein häufiges Anwendungsbeispiel für Stringalgorithmen ist die Analyse von Text in einer Suchmaschine. Hier kann die `find`-Funktion dazu verwendet werden, Schlüsselwörter zu identifizieren und die `replace`-Funktion, um Schreibfehler zu korrigieren. Ebenso ist die Tokenisierung von Text nützlich, um Benutzereingaben zu verarbeiten.

VI. Algorithmen für numerische Berechnungen

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- Numerische Integration:

```
double integrate(std::function<double(double)> f, double a, double b, int num_intervals) {
    double result = 0.0;
    double dx = (b - a) / num_intervals;
    for (int i = 0; i < num_intervals; ++i) {
        double x = a + i * dx;
        result += f(x) * dx;
    }
    return result;
}
```

- Numerische Differentiation:

```
double differentiate(std::function<double(double)> f, double x, double h) {
    return (f(x + h) - f(x)) / h;
}
```

VII. Effizienz und Komplexität

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- Es ist wichtig zu verstehen, wie sich Algorithmen in Bezug auf Zeit- und Speicherbedarf verhalten
- Wir betrachten die Laufzeitkomplexität von Algorithmen. Der lineare Suchalgorithmus hat eine Laufzeitkomplexität von O(n). Das bedeutet, dass die Anzahl der benötigten Schritte linear mit der Größe des Eingabegebiets wächst.

```
int linearSearch(const std::vector<int>& arr, int target) {
    for (int i = 0; i < arr.size(); ++i) {
        if (arr[i] == target) {
            return i;  // Element gefunden, Index wird zurückgegeben
        }
    }
    return -1;  // Element nicht gefunden
}
```

VII. Effizienz und Komplexität

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- Ein Algorithmus mit einer quadratischen Laufzeitkomplexität, wie das Bubble-Sort-Verfahren (O(n^2)), wird schnell langsamer, je größer das Eingabegebiet ist.

```
void bubbleSort(std::vector<int>& arr) {
    int n = arr.size();
    bool swapped;

    do {
        swapped = false;
        for (int i = 1; i < n; ++i) {
            if (arr[i - 1] > arr[i]) {
                std::swap(arr[i - 1], arr[i]);
                swapped = true;
            }
        }
    } while (swapped);
}
```

VII. Effizienz und Komplexität

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- Ein Algorithmus mit konstanter Laufzeitkomplexität O(1), wie das Einfügen in einen Vektor, bleibt unabhängig von der Eingabegröße schnell.

```
std::vector<int> numbers;
numbers.push_back(5); // Füge 5 hinzu
numbers.push_back(8); // Füge 8 hinzu
```

X. Quellen

1. Goodrich, Michael T.; Tamassia, Roberto; Mount, David M. “Data Structures & Algorithms in C++” , John Wiley & Sons, Inc., 2011

2. ChatGPT, Antworten zu “Gib ein kurzes Code-Beispiel zu (…)”, (https://chat.openai.com), Aufruf am 30.10.2023

3. cppreference.com: “Algorithms library” (https://en.cppreference.com/w/cpp/algorithm), Aufruf am 28.10.2023.

4. Microsoft Learn: “<algorithm>” (https://learn.microsoft.com/en-us/cpp/standard-library/algorithm?view=msvc-170), Aufruf am 29.10.2023.

5. Wikipedia: “C++ Standardbibliothek” (https://de.wikipedia.org/wiki/C%2B%2B-Standardbibliothek), Aufruf am 29.10.2023.