POS1 – Übungen zu Arrays

Schuljahr: 2023/24

Inhalte

- Arrays
- Zufallszahlen
- Funktionen
- Header-Dateien
- Sortieren

Sie können bei den folgenden Übungen davon ausgehen, dass der Benutzer die Eingabe derart durchführt, wie sie vom Programm erwartet wird!

Lab4.1 Erstellung und Ausgabe eines Arrays (fixer Größe) mit ganzen Zufallszahlen

Erstellen Sie die Dateien `randoms.cpp` und `randoms.h`! Definieren Sie in der Datei `randoms.h` die Prototypen für die Funktionen `void fill(int a[], int max)` und `void print(int a[], int max)`. Implementieren Sie die beiden Funktionen zusätzlich zur Funktion `int main(void)` in der Datei `randoms.cpp`. Vergessen Sie nicht, die Header-Datei `randoms.h` in der Datei `randoms.cpp` mittels Präprozessor-Direktive `#include` einzubinden!

Erstellen Sie in der Funktion `main` eine Konstante `MAX` mit dem Wert 10. Erstellen Sie sodann ein Array von ganzen Zahlen mit der Größe der Konstanten `MAX`. Rufen Sie nun die Funktion `fill` mit dem Array und der Größe des Arrays (als Parameter) auf! In der Funktion `fill` soll nun jedem Element des Arrays eine Zufallszahl aus dem Bereich 0-99 zugeordnet werden. Gehen Sie dazu mit Hilfe einer Schleife zu jedem Element des Arrays und weisen Sie jedem Element eine Zufallszahl zu (siehe dazu Verwendung von `srand(…)` und `random()` zur Erstellung von Zufallszahlen, beispielsweise unter https://cplusplus.com/forum/beginner/29699/)! Nach dem Aufruf von `fill` (in der Funktion `main`) soll nun die Funktion `print` – wieder mit dem Array und der Größe des Arrays als Parameter – aufgerufen werden. In der Funktion `print` sollen nun alle Elemente des Arrays auf der Konsole ausgegeben werden. Wurde das Array tatsächlich mit Zufallszahlen gefüllt? Weshalb ist die Veränderung des Arrays in der Funktion `fill` auch in den Funktionen `main` und `print` sichtbar? Würde eine Veränderung auch dann sichtbar sein, wenn es sich beim ersten Parameter nicht um ein Array, sondern um eine einzelne Zahl handeln würde? Begründen Sie Ihre Antwort!

Lab4.2 Vermeidung von Duplikaten

Dazu ist es notwendig, dass die Implementierung der Funktion `fill` verändert wird. Klarerweise benötigen Sie weiterhin die Schleife, um über alle Elemente des Arrays zu iterieren. Auch die Erzeugung einer Zufallszahl bleibt uns nicht erspart, allerdings wird diese nicht sofort in das Array geschrieben, sondern in einer Hilfsvariablen „geparkt“. Gehen Sie mit Hilfe einer inneren (verschachtelten) Schleife über alle bereits zugewiesen Elemente und vergleichen Sie jedes einzelne bereits zugewiesene Element mit der Hilfsvariablen. Sollte eine Übereinstimmung vorliegen, ist eine neue Zufallszahl zu erzeugen, in der Hilfsvariablen zu speichern und wieder mit allen bereits zugewiesenen Elementen zu vergleichen. Dieser Vorgang (wiederum in Form einer Schleife implementiert, sinnvoller Weise einer `do-while` Schleife) wird solange fortgesetzt, bis die Hilfsvariable einen Wert angenommen hat, der sich noch nicht in den bereits zugewiesenen Werten befindet. Somit besteht der Algorithmus aus insgesamt drei ineinander verschachtelten Schleifen, um der Vorgabe, Duplikate zu vermeiden, gerecht zu werden.

Lab4.3 Berechnung der Summe aller im Array befindlichen Werte

Erstellen Sie in der Datei `randoms.cpp` eine Funktion `int sum(int a[], int max)`, welche mithilfe einer Schleife die Summe aller im Array befindlichen Werte berechnet und als Ergebnis zurück liefert. Vergessen Sie nicht auf die Definition des Prototypen der Funktion in der Datei `randoms.h`! Verwenden Sie nun die Funktion `sum` am Ende der Funktion `main` und geben Sie den Rückgabewert der Funktion auf der Konsole aus!

Lab4.4 Festlegung der Größe des Arrays zur Laufzeit

Lesen Sie am Beginn der Funktion `main` die Größe (ganze Zahl) für das Array von der Konsole ein anstatt die Konstante `MAX` zu verwenden. Beachten Sie, dass korrekterweise das Array nicht mehr als `int a[]` sondern als `int* a` definiert und Speicher mittels `a = new int[size]` angelegt werden muss! Überlegen Sie, ob nun auch die bereits vorhandenen Funktionen geändert werden müssen! Begründen Sie Ihre Antwort! Beachten Sie, dass der mit `new` angeforderte Speicher am Ende des Programms mit `delete` freigegeben werden muss!

Lab4.5 Festlegung einer Unter- und Obergrenze für die Werte der Zufallszahlen zur Laufzeit

Fügen Sie nach der Abfrage der Größe des Arrays in der Funktion `main` je eine Abfrage für die Unter- bzw. Obergrenze der Zufallszahlen ein! Beachten Sie, dass es zu einem Fehler kommt, wenn die Differenz aus Obergrenze und Untergrenze kleiner als die Größe des Arrays ist und keine doppelten Werte vorkommen dürfen!

Lab4.6 Sortieren des Arrays

Erstellen Sie in der Datei `randoms.cpp` eine Funktion `void sort(int a[], int max)`, welche mithilfe des BubbleSort-Algorithmus (siehe https://www.happycoders.eu/de/algorithmen/bubble-sort/) das gesamte Array aufsteigend sortiert. Überlegen Sie sich eine möglichst effiziente Implementierung des angegebenen Algorithmus! Vergessen Sie nicht auf die Definition des Prototypen der Funktion in der Datei `randoms.h`! Rufen Sie nun die Funktion `sort` am Ende der Funktion `main` auf und geben Sie das nun sortierte Array nochmals auf der Konsole aus. Wurde das Array tatsächlich sortiert? Weshalb ist die Veränderung des Arrays in der Funktion `sort` auch in den Funktionen `main` und `print` sichtbar? Begründen Sie Ihre Antwort!

Lab4.7 Berechnung und Ausgabe der Fibonacci-Zahlen

Erstellen Sie die Dateien `fibonacci.cpp` und `fibonacci.h`! Definieren Sie in der Datei `fibonacci.h` die Prototypen für die Funktionen `void fill(int a[], int max)` und `void print(int a[], int max)`. Implementieren Sie die beiden Funktionen zusätzlich zur Funktion `int main(void)` in der Datei `fibonacci.cpp`. Vergessen Sie nicht, die Header-Datei `fibonacci.h` in der Datei `fibonacci.cpp` mittels Präprozessor-Direktive `#include` einzubinden!

Lesen Sie am Beginn der Funktion `main` die Größe (ganze Zahl) für das Array von der Konsole ein. Beachten Sie, dass korrekterweise das Array nicht als `int a[]` sondern als `int* a` definiert und Speicher mittels `a = new int[size]` angelegt werden muss! Rufen Sie nun die Funktion `fill` mit dem Array und der Größe des Arrays (als Parameter) auf. In der Funktion `fill` sind nun die ersten beiden Stellen des Arrays mit dem Wert 1 zu befüllen. Die Werte aller anderen Stellen sind mit der Summe der beiden Vorgänger zu befüllen. Verwenden Sie nun die Funktion `print`, um die Werte des gesamten Arrays auf der Konsole auszugeben.

Beachten Sie, dass der mit `new` angeforderte Speicher am Ende des Programms mit `delete` freigegeben werden muss!

Lab4.8 Ermittlung sämtlicher natürlicher Primzahlen bis zu einer Obergrenze

Erstellen Sie die Dateien `prim.cpp` und `prim.h`! Definieren Sie in der Datei `prim.h` die Prototypen für die Funktionen `void fill(bool a[], int max)` und `void print(bool a[], int max)`. Implementieren Sie die beiden Funktionen zusätzlich zur Funktion `int main(void)` in der Datei `prim.cpp`. Vergessen Sie nicht, die Header-Datei `prim.h` in der Datei `prim.cpp` mittels Präprozessor-Direktive `#include` einzubinden! Lesen Sie am Beginn der Funktion `main` die Größe (ganze Zahl) für das Array von der Konsole ein. Beachten Sie, dass korrekterweise das Array nicht als `int a[]` sondern als `int* a` definiert und Speicher mittels `a = new int[size]` angelegt werden muss! Rufen Sie nun die Funktion `fill` mit dem Array und der Größe des Arrays (als Parameter) auf. In der Funktion `fill` sind nun die ersten beiden Stellen des Arrays mit dem Wert `false` zu befüllen. Die Werte aller anderen Stellen (`true` oder `false`) sind mithilfe des Sieb des Eratosthenes zu ermitteln (siehe https://www.mathe-lexikon.at/arithmetik/natuerliche-zahlen/teilbarkeit/primzahlen/sieb-des-eratosthenes.html). Überlegen Sie sich eine möglichst effiziente Implementierung des angegebenen Algorithmus! Verwenden Sie nun die Funktion `print`, um die Indizes jener Stellen im Array auszugeben, welche den Wert `true` beinhalten. Beachten Sie, dass der mit `new` angeforderte Speicher am Ende des Programms mit `delete` freigegeben werden muss!

Hinweise:

- Die Dateien mit dem Quellcode (`labxy.cpp`) ist mit einem Header, welcher zumindest Name, Klasse und Programmname enthält, zu versehen!
- Deklarieren Sie die notwendigen Funktionen in der Datei mit der Bezeichnung `labxy.h`!
- Fügen Sie sinnvolle Kommentare hinzu!
- Verwenden Sie sprechende englische Bezeichner für Variablen uns Konstanten!
- Beachten Sie richtige Einrückungen!
- Auf der Moodle-Plattform (www.eduvidual.at) finden Sie wertvolle Hinweise und Hilfestellungen zu diversen – in diesem Übungsbeispiel enthaltenen – Themen!
- Geben Sie das Beispiel bis zum angegebenen Termin über die Moodle-Plattform ab!