Übungsstunde

Woche 11

Adel Gavranović
adel.gavranovic@inf.ethz.ch

Overview

Heutige Themen

Intro

Bedeutung von & und ∗

References vs Pointers

Pointer-Arithmetik

Aufgabe: ”Push Back”

Outro

Links

- polybox zum Material für die Übungsstunden
- Mail an Assistenten

Follow-up aus vorheriger Übungsstunde

- Haben alle das Self-Assessment III gemacht?

Kommentare zu [code]expert

Bonus Exercise 1: ”Dots and Boxes”

- Die meisten Fehler sind darauf zurückzuführen, dass das Spielfeld nicht ausgegeben wurde, bevor man nach dem Zug gefragt hat. Dies ist leider kaum erkennbar, wenn man nur die ”expected output” mit dem ”actual output” vergleicht, aber man sieht es schnell, wenn man einfach eine Runde spielt...
- Weitere Fehlerquelle: Typos! (Vergesst nicht, der Autograder ist pedantisch!)

Kommentare zu [code]expert

Bonus Exercise 2: ”Nonogram”

- Die zwei grossen ”Timeout-Quellen”
  1. Immer *alle* Reihen und Spalten nach Validität kontrollieren, statt einfach direkt ein `return false // invalid` zurückzugeben, sobald man auf einen invaliden Eintrag trifft (da ja sofort das gesamte Nonogram damit invalidiert wird)
  2. Kopien
- Lösungen
  1. So schnell wie möglich aus Rekursionen und Schleifen aussteigen
  2. Pass-by-reference (`&`) statt pass-by-value für grosse Objekte (wie eben `std::vector`)

Kommentare zu [code]expert

E9:T1: ”Trains”

- unbedingt auf die Zusammenfassung für die Prüfung...
- PRE/POSTs noch spezifischer (`// returns TRUE if valid train, then consumes it`)
- ”nested `if` ” sind unschön
- soweit es geht mit Rekursion statt mit Loops arbeiten

```
1  if(consume(is,'<') && open(is) && loco(is) && consume(is,'>')){
2    return true;
3  } else {
4    return false;
5  }
6  // or even...
7  // return (consume(is, '<')    &&
8  //         open(is)            &&
9  //         loco(is)            &&
10 //         consume(is, '>')    );
```

Kommentare zu [code]expert

E9:T2: ”Money”

- Sehr viele haben die Dokumentation komplett vernachlässigt :(

E9:T3: ”Prefix to Infix”

- gratuliere, falls ihr das geschafft habt (4/17)!
- falls nicht: don’t worry. Habe ich damals auch nicht :’)

Fragen zu [code]expert eurerseits?

Lernziele

- ☐ Den Unterschied zwischen Pointers und Referenzen kennen und erklären können
- ☐ Programme, die Pointer(-arithmetik) nutzen, tracen können
- ☐ Programme, die Pointer(-rithmetik) nutzen, schreiben können
- ☐ Programme, die dynamischen Speicher nutzen, tracen können
- ☐ Programme, die dynamischen Speicher nutzen, schreiben können

Bedeutungen von `&`

Das Symbol `&` hat in C++ viele Bedeutungen. Das ist verwirrend.
Es hat *3 verschiedene Bedeutungen*, je nach Position im Code:

Bedeutung von `&`

1. als AND-operator
   ```
   bool z = x && y;
   ```
2. um eine Variable als Alias zu *deklarieren*
   ```
   int& y = x;
   ```
3. um die *Adresse*[1] einer Variable zu erhalten (address-operator)
   ```
   int *ptr_a = &a;
   ```

[1] ein ”*d*” im Deutschen, zwei ”*d*” im Englischen

Bedeutungen von `*`

Dito mit dem Symbol `*` :

Bedeutung von `*`

1. als (arithmetischer) Multiplikation-operator
   ```
   z = x * y;
   ```
2. um eine Pointer-Variable zu deklarieren
   ```
   int* ptr_a = &a;
   ```
3. um auf eine Variable via ihrem Pointer zuzugreifen (dereference-operator)
   ```
   int a = *ptr_a;
   ```

Fragen/Unklarheiten?

References

```
1  // TODO: Trace program and write expected output
2  void references(){
3    int a = 1;
4    int b = 2;
5    int& x = a;
6    int& y = x;
7    y = b;
8
9    std::cout
10   << a << " "
11   << b << " "
12   << x << " "
13   << y << std::endl;
14 }
```

Pointers

```
1  // TODO: Trace program and write expected output
2  void pointers(){
3    int a = 1;
4    int b = 2;
5    int* x = &a;
6    int* y = x;
7
8
9    std::cout
10   << a << " "
11   << b << " "
12   << x << " "
13   << y << std::endl;
14
15   y = 0;
16 }
```

Pointers & Addresses

```
1  // TODO: Trace program and write expected output
2  void ptrs_and_addresses(){
3    int a = 5;
4    int b = 7;
5
6    int* x = nullptr;
7    x = &a;
8
9    std::cout << a << "\n";
10   std::cout << *x << "\n";
11
12   std::cout << x << "\n";
13   std::cout << &a << "\n";
14
15   x = &b;
16   *x = 1;
17 }
```

Tracing noch drauf?

▸ Tracing Guideline einfach runterscroller zum Teil zu Pointer

Fragen/Unklarheiten?

Pointer Program

```
int* a = new int[5]{0, 8, 7, 2, -1};
int* ptr = a;                          // pointer assignment
++ptr;                                 // shift to the right
int my_int = *ptr;                     // read target
ptr += 2;                              // shift by 2 elements
*ptr = 18;                             // overwrite target
int* past = a+5;
std::cout << (ptr < past) << "\n"; // compare pointers
```

Pointer Program

Find and fix at least 3 problems in the following program.

```
#include <iostream>
int main () {
    int* a = new int[7]{0, 6, 5, 3, 2, 4, 1};
    int* b = new int[7];
    int* c = b;
    // copy a into b using pointers
    for (int* p = a; p <= a+7; ++p) {
        *c++ = *p;
    }
    // cross-check with random access
    for (int i = 0; i <= 7; ++i) {
        if (a[i] != c[i]) {
            std::cout << "Oops, copy error...\n";
        }
    }
    return 0;
}
```

Pointer Program

```
#include <iostream>
int main () {
    int* a = new int[7]{0, 6, 5, 3, 2,
    int* b = new int[7];
    int* c = b;
    // copy a into b using pointers
    for (int* p = a; p <= a+7; ++p) {
        *c++ = *p;
    }
    // cross-check with random access
    for (int i = 0; i <= 7; ++i) {
        if (a[i] != c[i]) {
            std::cout << "Oops, copy error...\n";
        }
    }
    return 0;
}
```

p = a+7 is dereferenced

Solution:
Use < instead of <=

(From: Script Exercise 117)

Pointer Program

```
#include <iostream>
int main () {
    int* a = new int[7]{0, 6, 5, 3, 2, 4
    int* b = new int[7];
    int* c = b;
    // copy a into b using pointers
    for (int* p = a; p <= a+7; ++p) {
        *c++ = *p;
    }
    // cross-check with random access
    for (int i = 0; i <= 7; ++i) {
        if (a[i] != c[i]) {
            std::cout << "Oops, copy error
        }
    }
    return 0;
}
```

p = a+7 is dereferenced

Solution:
Use < instead of <=

Same problem as above

(From: Script Exercise 117)

Pointer Program

```
#include <iostream>
int main () {
    int* a = new int[7]{0, 6, 5, 3, 2, 4, 1};
    int* b = new int[7];
    int* c = b;
    // copy a into b using pointers
    for (int* p = a; p <= a+7; ++p) {
        *c++ = *p;

    // cross-check with random access
    for (int i = 0; i <= 7; ++i) {
        if (a[i] != c[i]) {
            std::cout << "Oops, copy err
        }
    }
    return 0;
}
```

`c` doesn't point to `b[0]` anymore.

Solution:
Use `b` instead of `c`

`p = a+7` is dereferenced

Solution:
Use `<` instead of `<=`

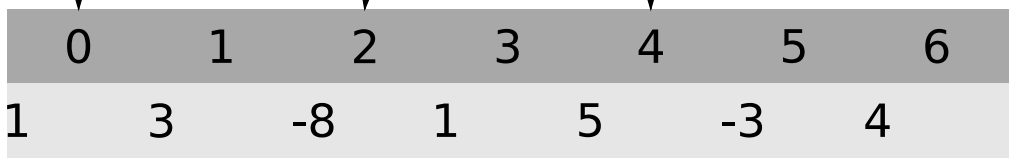
Same problem as above

Exercise – Applying Pointers

Exercise – Applying Pointers

```
// PRE: [b, e) and [o, o+(e-b)) are disjoint
//      valid ranges
void f (int* b, int* e, int* o) {
    while (b != e) {
        --e;
        *o = *e;
        ++o;
    }
}
```

Variable	Value
b	●
e	●
o	●



Exercise – Applying Pointers

Now determine a POST-condition for the function.

```
// PRE: [b, e) and [o, o+(e-b)) are disjoint
//      valid ranges
void f (int* b, int* e, int* o) {
    while (b != e) {
        --e;
        *o = *e;
        ++o;
    }
}
```

Exercise – Applying Pointers

```
// PRE: [b, e) and [o, o+(e-b)) are disjoint
//      valid ranges
// POST: The range [b, e) is copied in reverse
//       order into the range [o, o+(e-b))
void f (int* b, int* e, int* o) {
    while (b != e) {
        --e;
        *o = *e;
        ++o;
    }
}
```

(From: Script Exercise 113)

Exercise – Valid Inputs

Exercise – Valid Inputs

- Which of these inputs are valid?

```
int* a = new int[5];
// Initialise a.
  a)  f(a, a+5, a+5);
  b)  f(a, a+2, a+3);
  c)  f(a, a+3, a+2);
```

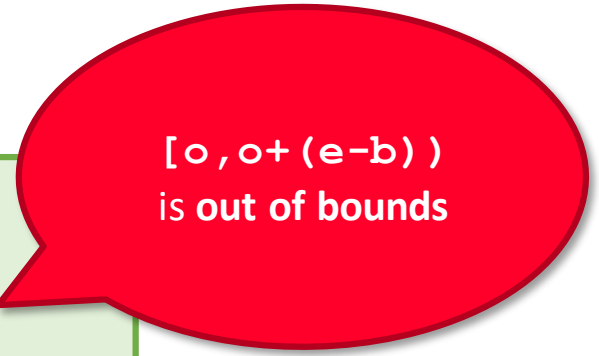
```
// PRE: [b, e) and [o, o+(e-b)) are disjoint
//      valid ranges
void f (int* b, int* e, int* o) {
    while (b != e) {
        --e;
        *o = *e;
        ++o;
    }
}
```

(From: Script Exercise 113)

Exercise – Valid Inputs

- Which of these inputs are valid?

```
int* a = new int[5];
// Initialise a.
  a)  f(a, a+5, a+5);
  b)  f(a, a+2, a+3);
  c)  f(a, a+3, a+2);
```




```
// PRE: [b, e) and [o, o+(e-b)) are disjoint
//      valid ranges
void f (int* b, int* e, int* o) {
    while (b != e) {
        --e;
        *o = *e;
        ++o;
    }
}
```

(From: Script Exercise 113)

Exercise – Valid Inputs

- Which of these inputs are valid?

```
int* a = new int[5];
// Initialise a.
  a)  f(a, a+5, a+5);
  b)  f(a, a+2, a+3);
  c)  f(a, a+3, a+2);
```

[o,o+(e-b)) is out of bounds

```
// PRE: [b, e) and [o, o+(e-b)) are disjoint
//      valid ranges
void f (int* b, int* e, int* o) {
    while (b != e) {
        --e;
        *o = *e;
        ++o;
    }
}
```

Exercise – Valid Inputs

- Which of these inputs are valid?

```
int* a = new int[5];
// Initialise a.
  a)  f(a, a+5, a+5);
  b)  f(a, a+2, a+3);
  c)  f(a, a+3, a+2);
```

[o,o+(e-b)) is **out of bounds**

Ranges **not disjoint**

```
// PRE: [b, e) and [o, o+(e-b)) are disjoint
//      valid ranges
void f (int* b, int* e, int* o) {
    while (b != e) {
        --e;
        *o = *e;
        ++o;
    }
}
```

(From: Script Exercise 113)

Exercise – `const` Correctness

Exercise – `const` Correctness

- Make the function `const`-correct.

```
// PRE: [b, e) and [o, o+(e-b)) are disjoint
//      valid ranges
void f (int* b, int* e, int* o) {
    while (b != e) {
        --e;
        *o = *e;
        ++o;
    }
}
```

(From: Script Exercise 113)

Exercise – `const` Correctness

- Make the function `const`-correct.

```
// PRE: [b, e) and [o, o+(e-b)) are disjoint
//      valid ranges
void f (const int* const b, const int* e, int* o) {
    while (b != e) {
        --e;
        *o = *e;
        ++o;
    }
}
```

Constness und Pointers[2]

`const` (Zeiger)	Zeiger **Konstantheit**

Es gibt zwei Arten von Konstantheit:

kein Schreibzugriff auf Target: `const int* a_ptr = &a;`
kein Schreibzugriff auf Zeiger: `int* const a_ptr = &a;`

```
int a = 5;
int b = 8;

const int* ptr_1 = &a;
*ptr_1 = 3; // NOT valid (change target)
ptr_1 = &b; // valid (change pointer)

int* const ptr_2 = &a;
*ptr_2 = 3; // valid (change target)
ptr_2 = &b; // NOT valid (change pointer)

const int* const ptr_3 = &a;
*ptr_3 = 3; // NOT valid (change target)
ptr_3 = &b; // NOT valid (change pointer)
```

[2] Falls euch das `const*const&`-Zeugs noch immer verwirrt, schaut euch die Summary zum Thema auf der Course Page an.

Fragen/Unklarheiten?

Aufgabe ”Push Back”

Aufgabe

1. Öffnet ”Push Back” auf [code]expert
2. Versucht es zu implementieren
3. High-level involviert das folgende Schritte:
   - 3.1 Alloziiert einen neuen Memory-Block der um einen Slot grösser ist
   - 3.2 Kopiert alle Elemente aus dem alten Memory-Block in den neuen
   - 3.3 Hängt das neue Element ganz am Ende des neuen Memory-Blocks an
4. Teilt und vergleicht eure Implementationen untereinander

Was zum f*&k ist `this->`?

Basically[3]

- " `this->` " hat zwei Teile: `this` und `->`
- `this` ist ein Pointer zum aktuellen Objekt (Class oder Struct), also ist es vom Typ `T*`
- `->` ist ein sehr cool aussehender Operator `this->member_element` ist äquivalent zu `*(this).member_element` Der Pfeil-Operator dereferenziert einen Pointer zu einem Objekt, um auf einen seiner Members zuzugreifen (Funktionen oder Variablen)
- Mehr Details folgen...

[3] a word I like to preface bad explanations and oversimplifications with

Fragen/Unklarheiten?

Tipps für [code]expert

E11:T2: ”Averager”

- easy Aufgabe, wenn man die Syntax kennt
- unbedingt mit Zusammenfassung zur Syntax angehen

E11:T3: ”Understanding Pointers”

- die ersten drei sind Einzeiler
- die letzte Aufgabe ist einfach, wenn man eine Skizzen anfertigt

E11:T4: ”Quick Sort”

- nehmt euch Zeit
- eigentlich muss man nicht wissen, wie der Algorithmus eigentlich funktioniert, also lasst euch davon nicht verwirren

Allgemeine Fragen?

Bis zum nächsten Mal

- ☐ macht eure Hausaufgaben
- ☐ bleibt gesund