Programmiertechnik 1

Unit 7: Programmiersprache C - Kontrollfluss

Ablauf

- Anweisungen und Blöcke
- if-else- else-if- switch
- Schleifen: while and for, do-while
- break and continue
- goto and labels

Anweisungen und Blöcke

- Kontrollfluss gibt Reihenfolge von Berechnungen an
- Ausdruck wird zu Anweisung durch nachfolgendes Semikolon (;)
  - x = 0; i++; printf(...);
  - Semikolon ist Trennzeichen
    - Zwischen Anweisungen
    - ...anders als Pascal, wo Semikolon Anweisung abschliesst
- Geschweifte Klammern ({, }) (Blöcke)
  - Gruppieren Deklarationen und Anweisungen zu Verbundanweisungen
  - Verbundanweisungen sind syntaktisch äquivalent zu einfachen Anweisungen
  - Kein Semikolon am Ende eines Blockes
  - Variablen können in jedem Block deklariert werden → Sichtbarkeit
- Einfaches Semikolon → leere Anweisung

scope.c

if-else-Anweisung

- Drückt Entscheidungen aus (Verzweigungen)

```
if ( ausdruck )
    anweisung1
else
    anweisung2
```

  - else-Teil ist optional
  - Ausdruck wird evaluiert
    - Wahr (Wert != 0) → $anweisung_1$ ausführen
    - Falsch (Wert == 0) → $anweisung_2$ ausführen
- Kurzfassung
    - if (ausdruck != 0)... Kann ausgedrückt werden als if (ausdruck) ...
- Geschachtelte else-Zweige sind mehrdeutig

```
if (n > 0) if (a > b ) z = a; else z = b;
```

  - else wird immer dem innersten (nächsten) if zugeordnet

Geschachtelte if-else-Anweisungen

- Mehrdeutigkeit – ein Beispiel

```
if ( n > 0 )
  for ( i = 0; i < n; i++ )
      if (s[i] > 0 ) {
          printf("...");
          return i;
      }
else                    /* Fehler, irreführende Einrückung */
  printf("Fehler: n ist negativ\n");
```

- Lösung:
  - Geschweifte Klammern zum Gruppieren von Anweisungen bei geschachtelten if-else-Anweisungen verwenden

else-if

- Häufiger Spezialfall:

```
if ( ausdruck ) anweisung
else if ( ausdruck ) anweisung
else if ( ausdruck ) anweisung
....
else anweisung
```

- Mehr-Wege-Entscheidung
  - Ausdrücke werden der Reihe nach ausgewertet
  - Erster wahrer Ausdruck bewirkt Ausführen der zugehörigen Anweisung, weitere Abfrage terminiert
  - Bsp: binäre Suche,
    - Finde x in v[0]....v[n-1], Schleife

```
if ( x < v[mid] ) high = mid-1;
else if ( x > v[mid] ) low = mid+1;
else return mid; /* match */
```

switch-Anweisung

- Mehr-Wege-Entscheidung
  - Test, ob ein Ausdruck auf einen von mehreren konstanten Integern passt

```
switch ( ausdruck ) {
    case konstanter-ausdruck: anweisungen
    case konstanter-ausdruck: anweisungen
    default: anweisungen
}
```

  - Fallunterscheidung, alle Fälle müssen verschieden sein
  - default-Fall wird ausgeführt wenn nichts anderes passt
    - default: kann fehlen
    - Reihenfolge der Fälle ist egal
  - break-Anweisung
    - Führt zum sofortigen Verlassen der switch-Anweisung
    - „Durchfallen“: fehlt am Ende eines case-Ausdrucks ein break, so wird einfach mit den Anweisungen hinter dem nächsten case weitergemacht
  - Defensive Programmierung:
    - Break am Ende von default → vermeidet Fehler bei Erweitern der Fallunterscheidung

switch.c

break-Anweisung

- Abbruch der Abarbeitung einer Anweisung
  - switch, auch bei while, for, do-Schleifen
- „Durchfallen“ bei case-Ausdrücken
  - Flexibel; mehrere Fälle können mit einer einzigen Aktion behandelt werden
  - Gefährlich; kann bei Erweiterungen leicht übersehen werden
- Robuste Programmierung
  - Abschließen jeder case-Alternative mit break
  - break hinter default-Alternative
- Anderer Weg des Abbruchs
  - return-Anweisung
  - Springt aus der umgebenden Funktion heraus
  - Bricht dann auch Abarbeitung von switch ab

Einschub: Codegenerierung

- if-else-Anweisung
  - Codegenerierung:
    - if (E) S1 else S2 wird übersetzt nach:

      <code to evaluate E to t1>

      if_false t1 goto L1

      <code for S1>

      goto L2

      label L1: <code for S2>

      label L2:
  - Bei mehreren Alternativen ist Laufzeit nicht konstant
  - Maximale Laufzeit = O(#Alternativen)

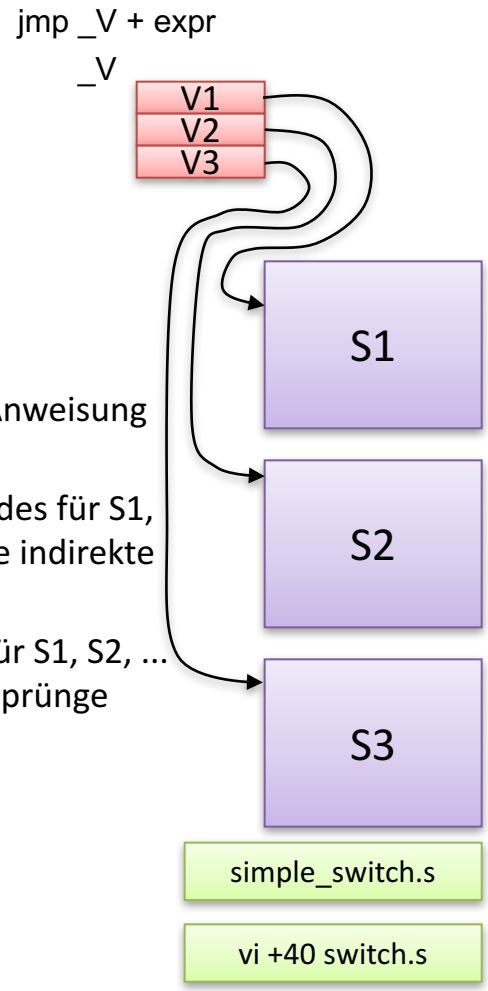
simple_if.s

Einschub: Codegenerierung

- switch-Anweisung: drei Alternativen zur Codegenerierung

```
switch (E) {
        case V1: S1
        case V2: S2
        ...
        default: Sn
}
```

  - (a) ist die Zahl der case-Ausdrücke V1, V2, ...klein so generiere code für if-Anweisung und teste case-Ausdrücke sequentiell
  - (b) ist der Wertebereich von V1, V2, ... klein, so speichere Adressen des Codes für S1, S2, ... in einem Feld (array) das durch V1, V2, ... indiziert wird und generiere indirekte Sprünge durch dieses Feld
  - (c) ist die Zahl der Labal V1, V2, ... groß, so speichere die Code-Adressesn für S1, S2, ... in einer hash-Tabelle mit den Labels als Schlüssel und generiere indirekte Sprünge durch diese Tabelle



- Laufzeit:
  - Bei (b) ist die Laufzeit konstant! O(1)
  - Bei (c) ist die Laufzeit << O(#Alternativen)

Schleifen – while and for

- Auswerten eines Ausdrucks, Abarbeitung der Schleife solange Wert != 0

  while (*ausdruck*)
  anweisung

  – Wird Ausdruck == 0, so wird mit nächster Anweisung fortgesetzt
  – Endlosschleife: while (1) { ... } – Abbruch mit break, return
- for-Schleife

  for ($ausdruck_1$; $ausdruck_2$; $ausdruck_3$)
  anweisung

  – Ist äquivalent zu (bis auf Verhalten der continue-Anweisung in der Schleife)

  $ausdruck_1$;
  while ($ausdruck_2$) {
  anweisung
  $ausdruck_3$;
  }

  – Typisch: $ausdruck_1$, $ausdruck_3$ *: Zuweisungen*
  – $ausdruck_2$ typischerweise Vergleich
  – Endlosschleife: for (; ;) { ... } – Abbruch mit break, return

Verwendung

- while
  - wenn keine Initialisierung nötig ist

```
/* skip white space */
while (( c = getchar()) == ' ' || c == '\n' || c == '\t' ) ;
```

- for
  - Wenn einfache Initialisierung und Inkrement-Operation pro Durchlauf
  - Iteration über ein Feld (array)

```
for ( i = 0; i < n; i++ ) ...
```

  - Vergleichbar mit Fortran DO-Schleife oder Pascal for
  - Aber:
    - Index und Limit können innerhalb der C for-Schleife verändert werden
    - Index-Variable behält ihren Wert nach Beendigung der Schleifenabarbeitung

Komma ”,”-Operator

- Zusammenfassen von Ausdrücken
  - Auswertung von links nach rechts
  - Type und Wert entsprechen dem rechten Ausdruck
  - Typischerweise in for-Schleifen verwendet

```
# include <string.h>

/* reverse a string in place */
void reverse( char s[] ) {
    int c, i, j;
    for (i = 0, j = strlen(s)-1; i < j; i++, j--) {
        c = s[i]; s[i] = s[j]; s[j] = c;
    }
}
```

  - Trennzeichen bei Funktionsargumenten ist kein Komma-Operator!
    → Auswertungsreihenfolge von Funktionsargumenten undefiniert
  - Selten verwenden!

reverse.c

do-while-Schleife

- Überprüfen der Abbruchbedingung am Schleifenende
  - Wird zumindest einmal ausgeführt

```
do
    anweisung
while ( ausdruck );
```

  - Schleife wird abgebrochen falls Ausdruck == 0 evaluiert wird
  - Seltener verwendet als while und for
  - Vergleichbar mit Pascal Repeat-Until-Schleife
    - Allerdings funktioniert dort der Test auf Schleifenabbruch umgekehrt

```
/* Zahl in Zeichenkette umwandeln – umgekehrte Reihenfolge */
i = 0;
do {
    s[i++] = n % 10 + '0';
} while ((n /= 10) > 0);
```

itob.c

break und continue

- Schleife abbrechen
- break:
  - In switch, for, while, do
  - nächste Anweisung nach der Anweisung (Schleife) wird ausgeführt
  - Bei verschachtelten Schleifen wird nächste Iteration der umgebenden Schleife begonnen

```
/* nachfolgende Leerzeichen/Tabs/Zeilenumbrüche entfernen */
int n; char s[] =”abc def g      ”;
for ( n = strlen(s) -1; n >= 0; n-- )
    if (s[n] != ’ ’ && s[n] != ’\t’ && s[n] != ’\n’) break;
s[n+1] = ’\0’;
```

- continue:
  - Nur in for, while, do
  - nächste Schleifeniteration wird begonnen

goto und Marken (labels)

Edsger Dijkstra (March 1968). "Go To Statement Considered Harmful“ Communications of the ACM 11 (3): 147–148. doi:10.1145/362929.362947.
"The unbridled use of the go to statement has as an immediate consequence that it becomes terribly hard to find a meaningful set of coordinates in which to describe the process progress. ... The go to statement as it stands is just too primitive, it is too much an invitation to make a mess of one's program."

- Sprunganweisung
  - Unterbricht sequentiellen Kontrollfluss
    - Programmstruktur wird sehr unübersichtlich
  - Beliebig missbrauchbar
  - Anwendung mitunter zur Fehlerbehandlung
    - Rücksprung aus großer Verschachtelungstiefe
  - Werkzeuge können mit Hilfe von goto effizienten Code generieren
    tabellen-gesteuerter Parser, yacc
  - goto kann immer durch andere Kontrollstrukturen ausgedrückt werden

```
for ( ... )
        for ( ... ) {
        ...
        if ( disaster )
                goto error;
        }
...


error:
        cleanup();
```

Lint (Splint)

- Splint (Secure Programming Lint)
  - Software für statische Quellcode-Analysen der Programmiersprache C
  - Splint ist eine indirekte Weiterentwicklung von Lint
  - analysiert C-Quellcode und weist auf wahrscheinliche Programmierfehler hin
    - typische C-Konstrukte in unüblicher Art und Weise verwendet
    - Wahrscheinlich Verwechslung ähnlicher Operatoren
    - Vergessen eines Klammerpaars, etc.
  - Interpretiert Annotationen um zwischen absichtlicher und versehentlicher Verwendung eines Konstrukts zu unterscheiden
- Beispiele:
  - Variable c wird mit 'x' verglichen, obwohl ihr kein Wert zugewiesen wurde
  - Der Rückgabewert der Funktion getchar ist vom Typ int, wird aber einer Variablen vom Typ char zugewiesen
  - C erlaubt Zuweisungen innerhalb von Ausdrücken. Das ist oft unbeabsichtigt.
  - In einer switch-Anweisung sollte jeder Zweig explizit mit einem break abgeschlossen werden, ansonsten wird der Code des nächsten Zweiges mit ausgeführt.
- http://www.splint.org/download.html

Die 10 Gebote für C Programmierer

Henry Spencer

1. Du sollst lint regelmäßig anwenden; und studiere seine Verkündigungen mit Bedacht, denn oft übertreffen die Wahrhaftigkeit seiner Erkenntnisse und Urteile die eigenen.
2. Du sollst keinem NULL Zeiger folgen, denn Chaos und Wahnsinn erwarten dich am Ende dieses Weges.
3. Du sollst die Art aller Funktionsargumente an die erwartete anpassen, falls sie das nicht schon sind; tue dies auch, solltest du der Meinung sein, es sei nicht von Nöten; denn sie nähmen grausame Rache, wenn du es am wenigsten erwartest.
4. Falls missratene Header Dateien nicht den Rückgabetyp seiner Bibliotheksfunktionen ausrufen, so rufe diese, mit akribischer Genauigkeit, selbst aus, oder schmerzliche Pein möge dein Programm befallen.
5. Du sollst die Grenzen aller Strings, nein aller Arrays, prüfen, denn wo du 'foo' eingibst, wird eines Tages jemand 'supercalifragilisticexpialidocious‘ übergeben.

Die 10 Gebote für C Programmierer

Henry Spencer

6. Falls eine Funktion verkündet sie setze einen Fehlercode, sollte sie von Schwierigkeiten ereilt werden, dann sollst du diesen Fehlercode prüfen; Ja, prüfe diesen auch, sollten die Tests die Grösse deine Codes verdreifachen, denn die Göttern sollen jede für ihre Arroganz strafen, die denken 'derlei Probleme könnten mich nie ereilen'.

7. Du sollst Bibliotheken studieren und nicht dem Neuerfinden dieser, ohne triftigen Grundes, ergeben. Möge dein Code kurz und lesbar sein, und mögen die Tage freundlich und produktiv werden.

8. Du sollst den Zweck und die Struktur deines Programmes für deinen Nächsten klar gestalten indem du den 'einzig wahren Klammerstil' nutzen sollst. Und mögest du den Stil auch nicht mögen, so nutze ihn doch, denn Kreativität sollte nur zum Lösen von Problemen verwendet, und nicht zur Schaffung neuer, hübscher Hindernisse, für das Verständnis des Codes, verschwendet werden.

9. Die externen Bezeichner sollen einzigartig in ihren ersten sechs Buchstaben sein; und sei dieses raue Benehmen auch verdrießlich, und mögen sich auch die Jahre auch scheinbar ohne Ende dahinziehen, in denen du darauf wartest seine Wahrhaftigkeit zu erkennen. So mögest du dem Wahnsinn entgehen, an jenem einen schicksalshaften Tage, an welchem du danach verlangest dein Programm auf einem alten System laufen zu lassen.

10. Schwöre ab, verleugne und entsage der abscheulichen Ketzerei, welche folgendene ungeheuerliche Behauptung vertritt: 'Alles in der Welt ist eine VAX!'; Und habe keinen Umgang mit den gottlosen Heiden, die jenem barbarischen Irrglauben anhängen. So mögen die Tage des Programms lang sein, auch wenn die Tage der Maschine kurz sind.

original

Obfuscated C Contest

www.ioccc.org

GOALS OF THE CONTEST:

- To write the most Obscure/Obfuscated C program under the rules below.
- To show the importance of programming style, in an ironic way.
- To stress C compilers with unusual code.
- To illustrate some of the subtleties of the C language.
- To provide a safe forum for poor C code. :-)

Detailed Rules...

1. Your entry must be a complete program.
2. The size of your program source must be <= 4096 bytes in length.
   The number of characters excluding whitespace (tab, space, newline, formfeed, return), and excluding any ; { or } immediately followed by whitespace or end of file, must be <= 2048.
3. ...

Beispiel aus 2011:
akari.c


/*
[>i>n[t
*/ #include<stdio.h>
/*2w0,1m2,]_<n+a m+o>r>i>=>(['0n1'0)1;
*/int/**/main(int/**/n,char**m){FILE*p,*q;int A,k,a,r,i/*
#uinndcelfu_dset<rsitcdti_oa.nhs>i/_*/;char*d="P%" "d\n%d\40%d"/**/
"\n%d\n\00wb+",b[1024],y[]="yuriyurararararayuruyuri*daijiken**akkari~n**"
"/y*u*k/riin<ty(uyr)g,aur,arr[a1r2a82*y2*/u*r{uyu}riOcyurhiyua**rrar+*arayra*="
"yuruyurwiyuriyurara'rariayuruyuriyuriyu>rarararayuruy9uriyu3riyurar_aBrMaPrOaWy^?"
"*]/f]`;hvroai<dp/f*i*s/<ii(f)a{tpguat<cahfaurh(+uf)a;f}vivn+tf/g*`*w/jmaa+i`ni("/**
*/"i+k[>+b+i>++b++>1[rb";int/**/u;for(i=0;i<101;i++)y[i*2]^="~hktrvg~dmG*eoa+%squ#12"
":(wn\"11))v?wM353{/Y;lgcGp`vedllwudvOK`cct~[|ju {stkjalor(stwvne\"gt\"yogYURUYURI"[
i]^y[i*2+1]^4;/*!*/p=(n>1&&(m[1][0]-'-'||m[1][1] !='\0'))?fopen(m[1],y+298):stdin;
/*y/riynrt~(^w^)],]c+h+a+r+*+*[n>)+{>f+o<r<(-m] =<2<5<64;}-]-(m+;yry[rm*])/[*
/q=(n<3||!(m[2][0]-'-'||m[2][1]))?stdout /]{ }[*/:fopen(m[2],d+14);if(!p||/*
"]<<*-]>y++>u>>+r >+u+++y>--u---r>++i+++" <)< ;[>-m-.>a-.-i.++n.>[(w)*/!q/**/)
return+printf("Can " "not\x20open\40%s\40" "" "for\40%sing\n",m[!p?1:2],!p?/*
o=82]5<<+(+3+1+&.(+ m +-+1.)<)<|<|.6>4>-+(> m- &-1.9-2-)-|-|.28>-w-?-m.:>([28+
/"read":"writ");for (a=k=u= 0;y[u]; u=2 +u){y[k++]=y[u];}if((a=fread(b,1,1024/
,mY/R*Y"R*/,p/*U*/)/* R*/)>/*U{ */ 2&& b/*Y*/[0]/*U*/=='P' &&4==/*"y*r/y)r\}
/sscanf(b,d,&k,& A,& i, &r)&& ! (k-6&&k -5)&&r==255){u=A;if(n>3){/
]&<1<6<?<m.-+1>3> +:+ .1>3+++ . -m-) -;.u+=++.1<0< <; f<o<r<(.;<([m(=)/8*/
u++;i++;}fprintf (q, d,k, u >>1,i>>1,r);u = k-5?8:4;k=3;}else
/*]>*/{(u)=/*{ p> >u >t>-]s >++(.yryr*/+(n+14>17)?8/4:8*5/
4;}for(r=i=0 ; ;){u*=6;u+= (n>3?1:0);if (y[u]&01)fputc(/*
<g-e<t.c>h.a r -(-).)8+<1. >;+i.(<)< <)+{+i.f>([180*/1*
(r),q);if(y[u]&16)k=A;if (y[u]&2)k--;if(i/*
("^w^NAMORI; { I*/==a/*")*/){/**/i=a(u)*11
&255;if(1&&0>= (a= fread(b,1,1024,p))&&
")]i>(w)-;} { /i-f-(-m--M1-0.)<{"
[8]==59/* */)break;i=0;}r=b[i++]
;u+=(/**>> *..</<<<)<[[;]**/+8&*
(y+u))?(10- r?4:2):(y[u] &4)?(k?2:4):2;u=y[u/*
49;7i\(w)/;} y}ru\=*ri[,mc]o;n}trientuu ren (
*/]-(int)'`';} fclose(p);k= +fclose(q);
/*] <*.na/m*o{ri{ d;^w^;} }^_^}}
" */ return k- -1+ /*\' '-`*/
(-/*}/ */0x01); {;{ }}
; /*^w^*/ ;}

Selbstreferentielle Programme

- Ein Quine ist ein Computerprogramm, das seine Kopie ausgibt
  - üblicherweise seines Quelltextes
  - eine Form der Selbstbezüglichkeit
  - Benannt nach dem Logiker und Philosophen Willard Van Orman Quine
- C-Programme werden übersetzt
  - die Laufzeitversion des Programms liegt in Maschinensprache vor
  - ursprüngliche Repräsention ist ein ASCII-codierter Quelltext
- Idee: legen Quelltext als Daten ab
- Ken Thompson: „Reflections on trusting Trust“

hello.c

quine.c

Zusammenfassung

- Anweisungen und Blöcke
  - ; trennt Anweisungen
  - {} eröffnen neuen Sichtbarkeitsbereich
- if-else- else-if- switch
  - Einfache / Mehrwege-Verzweigungen
  - Switch ist effizienter bei vielen disjunkten Fällen
- Schleifen: while and for, do-while
  - Mächtige Konstrukte, flexible Initialisierung und Iteration
  - Keinen „fremden Code“ in Schleifeninitialisierung unterbringen
- break and continue
  - Abarbeitung der aktuellen Anweisung abbrechen
- goto and labels
  - Beliebige Sprünge durch das Programm; gefährlich!