Programmieren mit C++

Zeiger (pointer) und Text (string)

Inhalt

Zeiger auf Adressen im Hauptspeicher

Jede Variable belegt einen Bereich des Hauptspeichers im RAM. Der Variablenname dient stellvertretend für die Adresse dieses Speicherplatzes und der Wert der Variablen steht in dem Speicherbereich an dieser Adresse.

Nun kann man auch die Adressen selbst speichern. Sie erhalten einen Variablennamen und belegen 4 Bytes (Typ long integer). Diese Adressvariable nennt man **Zeiger** (engl. Pointer).

Datentyp und Variablenname	Adresse	Wert
`int i`	0012FE6C	10
`float f`	0012FE54	0.5
`double d`	0012FE38	123457
`int * pi`	0012FE78	0012FE6C
`float * pf`	0012FE60	0012FE54
`double * pd`	0012FE48	0012FE38

Der Verweis-Operator

Die Deklaration eines Zeigers enthält hinter dem Typ den **Verweisoperator * :**

Deklaration: `int * pi;`

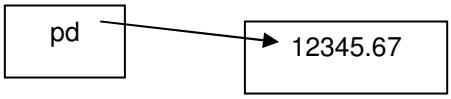
Der Datentyp int * ist ein Zeiger auf eine int-Variable. Das heißt, die Variable pi kann die Adresse einer int-Variablen speichern.

Verwendung: int i = 10; pi = &i; // Jetzt enthält pi die Adresse von i.

Der Adress-Operator

Mit dem **Adressoperator &** wird die Adresse einer **zugewiesenen** Variablen gewonnen:

```
double d = 12345.67;
double * pd = &d;  // pd enthält die Adresse von d
```



Man sagt: pd zeigt auf d

Übung

Programm	Ergänze die Ausgaben
`int x = 12;`	
`int * px ;`	
`cout << x << endl;`	[redacted]
`cout << &x << endl;`	0012FE8C
`px = &++x;`	
`cout << x << endl;`	[redacted]
`cout << px << endl;`	[redacted]
`cout << x-- << endl;`	[redacted]
`cout << *px << endl;`	[redacted]
`cout << x << endl;`	12

Funktionen

Unterprogramme heißen in C grundsätzlich "Funktionen". Sie werden verwendet

- um mehrmals wiederkehrende Programmteile nur einmal schreiben zu müssen,
- um einen inhaltlich abgeschlossenen, logisch zusammengehörenden Programmteil deutlich vom Rest des Programmes abzusetzen.

Beim Aufruf einer Funktion kann das aufrufende Programm der Funktion Werte zum Bearbeiten mitgeben. Diese Werte werden vor Beginn der Funktion in den **Variablenbereich der Funktion** kopiert, so dass die eigentlichen Originalwerte des aufrufenden Programms unangetastet bleiben.

Call by reference

Es soll eine Funktion erstellt werden, die **2 Variablenwerte vertauscht** (engl. swap).
Vertauschen kann man mit einer Hilfsvariablen, die einen der Werte zwischenspeichern kann:

```
    int h;
    h = a;
    a = b;
    b = h;
```

Welche Werte haben am Ende a und b, wenn zuerst

a=3 und b=7

ist? Welchen Wert speichert h? h= ___ a=____ b= _____

```
void swap (int a, int b)
{
    int h;
    h = a;
    a = b;
    b = h;
}
```

Diese Funktion vertauscht die übergebenen Variablenwerte aber nicht!
Der Grund:
Bei Aufruf der Funktion wurden die Werte aus dem Hauptprogramm nach a und b kopiert, a und b wurden zwar vertauscht, aber im Hauptprogramm bleibt alles beim Alten.

C++ Funktionen arbeiten **grundsätzlich mit Kopien der Übergabeparameter** (**call by value**).

Will man in einer Funktion Werte des Hauptprogramms verändern, so müssen der Funktion die Referenzen auf die Werte, nämlich die Adressen der Variablen übergeben werden. Diesen Vorgang nennt man **Call-by-reference.**

Das vorangegangene Beispiel muß also richtig so lauten:

```
void swap (int &a, int &b)
{
    int h;
    h = a;
    a = b;
    b = h;
}
```

Die Funktion „swap“ bekommt nun die Adresse der Variablen übergeben und greift über diese Adresse auf die Variablen im Hauptprogramm selbst zu.

Funktionsaufruf im Hauptprogramm

swap (x,y);

```
void swap (int* a, int* b)
{
    int h;
    h = *a;
    *a = *b;
    *b = h;
}
```

Hinweis:
Man kann die Funktion „swap“ auch so programmieren, dass als Übergabeparameter zwei Pointer mit dem * Operator verwendet werden. Jetzt sind die Werte natürlich *a und *b und beim Funktionsaufruf im Hauptprogramm muss man explizit die Adressen übergeben:

swap (&x,&y);

Vertauschungen werden z.B. in Sortierprogrammen eingesetzt.

Übung

```
int value(int x)
{
    x = x*x;
    return x+5;
}

int  valref(int & x)
{
      x = x*x;
    return x+5;
}

int main( ) {
        int a = 2;
        int b = value(a);
        cout << a << endl;
        cout << b << endl;

        int c = 2;
        int d = valref(c);
        cout << c << endl;
        cout << d << endl;
        return 0;
}
```

Wo ist der Unterschied bei den Funktionen value und valref?

Ergänze die Ausgaben

```
void fpower(double x, int e) {
      double h=x;
      for(int i=1;i<e;i++){
            h *= x;
      }
      x=h;
}
void power(double &x, int e) {
      double h=x;
      for(int i=1;i<e;i++){
            h *= x;
      }
      x=h;
}
void testpower(){
   double a,b;
   a=2.0;
   b=5.2;
   fpower(a,5);
   cout << a << endl;
   power(a,5);
   cout << a << endl;
   fpower(b,3);
   cout << b << endl;
   power(b,3);
   cout << b << endl;
}
```

Die Funktionen fpower und power unterscheiden sich nur bei den Parametern.

Welche Ausgaben hat das folgende Testprogramm?

Quiz

In einem Programm wurden mehrere Variable zugewiesen:

```
        int a, b;  // 4 Byte Variable
        double d, e;
        a=65; b= 3;
        d=65.0; e=3.0;
```

Welche Ausgabe hat

```
cout << a%b << '\t' << a/b << endl;
cout << (int)e/b << '\t' << (double)a/b << endl;
cout << (char)a << '\t' << a/e << endl;

cout << (a++)+b << '\t' << --b << '\t' << a << endl;
cout << a << endl;
cout << 3*++b << '\t' << 10+--a << endl;
cout << 'B'+2 << '\t' << 'a'-'A' << endl;
```

```
        int * pa;
        double * pd;
        pa = &a;
        pd=&d;
```

Welche Ausgabe hat

```
        cout << sizeof(a) << endl;;
        cout << sizeof(d) << endl;;
        cout << sizeof(pa) << endl;
        cout << sizeof(pd) << endl;
```

```
        int c = 0xff;
```

Welche Ausgabe hat

```
        cout << c << '\t' << c-0x0f << endl;
        cout << c+0xab << '\t' << c-0x23 << endl;
```

Zeichenketten, Texte, Strings

Deklaration: `char * n;`

Der Datentyp char * ist ein Zeiger auf **1** char-Variable. Das heißt, die Variable n kann die Adresse einer char-Variablen speichern.

Textvariable

Zeichenketten, genannt **Strings**, können als eine Folge von einzelnen ASCII-Zeichen definiert werden:

```
char * text;
```

deklariert die Variable „text“, weist ihr aber noch **keinen** Speicherbereich zu.

```
char text [30];
```

```
char * text = new char[30];
```

definiert die Variable „text“ und es werden **30 aufeinanderfolgende Bytes im RAM** reserviert.

```
char * text = "XYZ";
char text[] = "XYZ";
```

bewirkt eine **Initialisierung** bereits bei der Definition.

Speicherzellen	Adressen	
'X'	text	Nach Eingabe von "XYZ" werden 4 Bytes belegt.
'Y'	text+1	
'Z'	text+2	
'\0'	text+3	Als letztes Zeichen wird ein NULL-Byte angehängt.
…		
...		
...		

Das NULL-Byte `'\0'` am Ende des Strings wird vom System angehängt, es muss aber bei der Dimensionierung berücksichtigt werden. Beachten Sie, dass ein String nachträglich nicht mehr vergrößert werden kann.

Verwendung:
Zeiger sind nützlich, wenn auf eine Folge von Speicherzellen verwiesen werden soll. Das kommt zum Beispiel bei der Verarbeitung von Zeichenketten vor. Eine Zeichenkette muss immer mit \0 beendet werden, damit beim Durchlaufen aller Zeichen eine Abbruchbedingung existiert.

```
char * ort  = "Biberach";
for(int i=0; ort[i]!='\0' ; i++)
{
   cout << ort[i] << endl;    //……..
   cout << ort+i << endl;     //……..
}
cout << ort << endl;     //……..
```

Falls das Input-Stream-Objekt cin aus der Standardbibliothek für die Eingabe verwendet wird, muss ein genügend großer String definiert werden, welcher den Datenstrom aufnehmen kann. Siehe dazu das folgende Beispiel.

Eingabe und Ausgabe von Texten

C++ Quellcode	**Erklärung**

`char text[50]; // Definition`
 	
`// Eingabe`
`cout << "Geben Sie einen Text ein: " ;`
`cin >> text; getchar();`
 	
`// Ausgabe als String`
`cout  << text << endl;`
 	
`// Ausgabe zeichenweise`
`int i;`
`for (i=0; text[i]!='\0'; i++)`
`{`
`  cout << text[i] << endl;`
`}`
 	
`// Ausgabe der Anfangsadresse`
`int * p;`
`p = (int *)&text[0];`
`cout<<"Speicheradresse:"<< p <<endl;`
	Definition eines 50 Byte großen Strings
	
	Eingabeaufforderung
	cin liest die Eingabe in den reservierten Speicherbereich, getchar() löscht den Tastaturbuffer.
	
	Ausgabe bis zum Nullbyte
	
	Zeichenweise Ausgabe bis zum Nullbyte, wobei jedes Zeichen in eine neue Zeile geschrieben wird.
	
	Deklaration eines Zeigers
	Zuweisung der Adresse des 1. Zeichens des Strings

Eingabe in einer Funktion

Der Programmierer muss dafür sorgen, dass genügend Speicherplatz reserviert wird.

`char *  eingabe1()`
`{`
`  char text[100];`
`  cout << "Eingabe: ";`
`  cin >> text;`
`  return text;`
`}`
 	
`void main() {`
`  char * s;`
`  do`
`  {`
`  s = eingabe1();`
`  } while ( s[0] != 'e')  ;`
`}`
	So nicht!
	Temporäre Definition eines Strings, der mit dem return – Aufruf wieder gelöscht wird.
	
	Das Hauptprogramm übernimmt den Datenstrom, kann ihn aber nicht weiter verarbeiten.

`void  eingabe2(char * text)`
`{`
`  cout << "Eingabe: ";`
`  cin >> text;`
 	
`}`
 	
`void main() {`
`  char s[100];`
`  do`
`  {`
`  eingabe2(s);`
`  cout << s << endl;`
`  } while ( s[0] != 'e')  ;`
`}`
	Besser!
	
	Besser ist es, wenn das Hauptprogramm einen genügend großen String zur Verfügung stellt.
	
	Dieser wird dann in der Funktion gefüllt.

Messen der Länge eines Strings

In einer for-Schleife werden alle Zeichen durchlaufen und gezählt.

```
int length(char * text)
{
  int i;
  for(i=0; text[i]!='\0'; i++);
  return i;
}
```

Warum befindet sich nach der for-Schleife ein „;“ ?

Verschieben eines Strings im ASCII-Code

Der C++ Datentyp char definiert eigentlich nur die Zahlen des ASCII Codes, die erst bei der Ausgabe als Zeichen angezeigt werden. Deshalb ist es immer möglich auch mit den char-Variablen zu rechnen. Da sich die Grossbuchstaben im ASCII Code 32 Plätze tiefer als die Kleinbuchstaben befinden, kann durch subtrahieren von 32 ein Kleinbuchstabe in einen Großbuchstaben verwandelt werden.

```
char *  upper(char * text)
{
   for(int i=0; text[i]!='\0'; i++)
   {
     text[i] -= 32;
   }
   return text;
}
```

Welche Ausgabe hat

char * t1 = „schule“;

cout << upper(t1)

Einfügen eines Zeichens

Das Einfügen von Zeichen in einen String macht diesen länger. Deshalb kann man nicht dieselbe Stringvariable für das Ergebnis verwenden. Man braucht einen zweiten, größeren String für das Ergebnis.

```
void umlauteersetzen(char* alt,char* neu)
{
   int i, k=0;
   for( i=0; alt[i]!='\0'; i++)
   {
         if(alt[i]=='Ä' ){
                  neu[k] = 'A';
                  neu[k+1]= 'E';
                  k+=2;
         } else if(alt[i]=='Ö' ){
                  neu[k] = 'O';
                  neu[k+1]= 'E';
                  k+=2;
         } else if(alt[i]=='Ü' ){
                  neu[k] = 'U';
                  neu[k+1]= 'E';
                  k+=2;
         } else {
                  neu[k] = alt[i];
                  k++;
         }
   }
   neu[k]='\0';

 }
```

Alle Ä, Ö, Ü werden durch AE, OE, UE ersetzt.
Der Funktion werden Referenzen des alten und neuen Strings übergeben.

Wenn
alt = ÄöÜ

was ergibt sich dann für

neu = ?

Aufgabe 1

Programmieren Sie

1. eine Funktion bool istGleich(int, int) , die 2 Zahlen vergleicht

2. eine Funktion void verdoppeln(int *)

3. eine Funktion double fahrenheit(double), die zum Celsiuswert den Fahrenheitwert berechnet.
   fahrenheit = 5*celsius/9+32

4. eine Funktion bool istGleich(char *, char *) , die 2 Texte vergleicht

5. eine Funktion, die in einem Text jedes 3. Zeichen durch ein 'x' ersetzt.

6. eine Funktion
   void bindestriche(char *, char *),
   die in einem (als Parameter übergebenen) Text nach jedem Buchstaben ein Zeichen ‚-' einfügt.
   Z. B. „Eichhoernchen" wird zu „E-i-c-h-h-o-e-r-n-c-h-e-n-"

Aufgabe 2

In einem Programm sind die Variablen s und c definiert:

```
char s[4], c;
s[3]='\0';
```

Die Variable c wird als Ziffer eingeben:
Schreibe ein Programm, das den String s am Ende wie folgt in Abhängigkeit von c ausgibt.

char c							
'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'
s="___"	s="__x"	s="_w_"	s="_wx"	s="r__"	s="r_x"	s="rw_"	s="rwx"

Aufgabe 3

```
switch(option)
{
  case 0: cout << "A";break;
  case 1: cout << "B";
  case 2: cout << "C";
  case 3: cout << "D";break;
  default: cout << "X";
}
```

Ausgabe, wenn option = 0 _____

Ausgabe, wenn option = 1 _____

Ausgabe, wenn option = 2 _____

Ausgabe, wenn option = 3 _____

Ausgabe, wenn option = 4 _____

13 Quiz-Fragen

Frage 1

```
      char s[3];
      s[0]='a';s[1]='b';s[2]='\0';
      cout << s << endl;
```

A	ab0	B	ab
C	s	D	AB

Frage 2

```
      char s[3];
      s[0]='A'+32;s[1]='B'+32;s[2]='\0';
      cout << s << endl;
```

A	ab0	B	ab
C	s	D	AB

Frage 3

```
      int a=10;
      int * pa=&a;
      cout << &(*pa) << endl;
```

A	0012FE8C	B	&10
C	0x12FF	D	Fehlermeldung

Frage 4

```
      char s[6];
      s[0]='4';s[1]='b';s[2]='\0';s[3]='d';s[4]=65;s[5]='\0';
      for(int i=0;i<6;i++){
           cout << s[i];
      }
```

A	4bdA	B	4b0A0
C	4b dA	D	e2fs

Frage 5

```
char * ersetzen(char * text) {
   for(int i=0;text[i]!='\0';i++){
      if(i%3==0) {
           text[i]='-';
      }
   }
   return "Biberach";
}

// Welche Ausgabe hat
cout << ersetzen("Karl-Arnold-Schule");
```

A	-ar—A-no-d—ch-le	B	xarlxArnxld-xchuxe
C	kxrlxarxolx-sxhuxe	D	Biberach

Frage 6

```
      char s[5];
      s[0]='e';s[1]='2';s[2]='f';s[3]='s';s[4]='\0';
      for(int i=0;i<5;i++){
           cout << s[i];
      }
      cout << s;
```

A	e2fs e2fs	B	e2fs0e
C	e2fs	D	e2fs e

Frage 7

```
int a=10;
int * pa=&a;
cout << 2*(*pa) << endl;
```

A	10	B	0012FE8C
C	20	D	8

Frage 8

```
char s[5];
s[0]='e';s[1]='2';s[2]='f';s[3]='s';s[4]=0x00;
for(int i=0;i<5;i++){
     cout << s[i];
}
cout << s[0];
```

A	e2fs e2fs	B	e2fs0e
C	e2fs	D	e2fs e

Frage 9

```
int a=10;
int * pa=&a;
cout << 7/(double)a*2 << endl;
```

A	1	B	0
C	1.4	D	0.35

Frage 10

```
char * ersetzen(char * text) {
   for(int i=0;text[i]!='\0';i++){
      if(i%3==1) {
           text[i]='x';
      }
   }
   return text;
}
// Welche Ausgabe hat
cout << ersetzen("Karl-Arnold-Schule");
```

A	KxrlxArxolx-Sxhuxe	B	xarlxArnxld-xchuxe
C	kxrlxarxolx-sxhuxe	D	Biberach

Frage 11

```
int a=10;
int * pa=&a;
cout << --(++a) << endl;
```

A	10	B	11
C	9	D	12

Frage 12

```
int a=10;
int * pa=&a;
cout << (++a)-3 << endl;
```

A	10	B	8
C	9	D	7

Frage 13

```
int a=10;
int * pa=&a;
cout << 7/a*2 << endl;
```

A	1	B	0
C	1.4	D	0.35