2. *Variablen*

2.1.Verschiedene Typen von Variablen

Das sind veränderbare Daten. Sie werden allegemein so definiert:

datenTyp <variablenName>;

z.B.

Beispiel 1

```
float dasIstEinFloat;
```

Man kann aber der Variablen noch bei ihrer Definition einen Anfangswert geben:

Beispiel 2

```
float dasIstEinFloat = 1.35 ;
```

Daher sind diese zwei Abschnitte gleich und man sollte das erste benutzen (wenn dies zu keiner Unklarheit und Unlesbarkeit führt):

Beispiel 3

`float var = 1.34;`	`float var;` `var = 1.34;`

Achtung! In C++ unterscheidet man streng zwischen Klein- und Grossschreibung!

Man kann die Definition von mehreren Variablen in einen Ausdruck zusammenfassen:

Beispiel 4

```
float var1, var2;
float var3 = 0.34, var4 = 4.87;
```

Oder man kann die Initialisierung der Daten mit einem Ausdruck machen:

Beispiel 5

```
float var1 = 1.0/3.0 + 6.2;
```

2.1.1. Ganzzahltypen

Das sind die ganzzahligen Typen, die C++ kennt mit ihren typischen Wertebereichen:

Tabelle 1

signed char	-128...127
short int	-32768...32767
long int	-2147483648...2147483647
unsigned char	0...255
unsigned short	0...65535
unsigned long	0...4294967295

Die verschiedenen Typen unterscheiden sich nicht nur in ihren Wertebereichen, sondern auch in der Darstellung intern (benötigte Anzahl von Bytes für eine Variable).

Unsigned Variablen nehmen nur positive Werte (einschliesslich der Null) an, *signed* dagegen auch negative Werte. Bei den *short int* und *long int* kann man bei der Definition auf *int* verzichten und stattdessen nur *long* und *short* schreiben.

Achtung! Bei den Typen *char* wird die Unterscheidung zwischen *signed* und *unsigned* zwingend vorgeschrieben! Das Datentyp char ist aus der Sicht des Compilers ein anderer Typ.

Welche Typen man für eine Variable wählt hängt davon ab, welche Wertebereiche benötigt werden.

2.1.2. Fliesskommatypen

Die Fliesskomma – Typen sind nur drei:

Tabelle 2

float	$\pm 3.4*10^{38}$ (32 Bits)
double	$\pm 1.7*10^{308}$(64 Bits)
long double	$\pm 1.7*10^{4932}$ (80 Bits)

Beim Eingeben vom Komma benutzt man einen Punkt:

Beispiel 6

```
float var = 1.3;
```

2.1.3. Zeichentypen

Der Zeichentyp ist nur eins: *char*. Er speichert alle Zeichen der ASCII-Tabelle und belegt genau 1 Byte.

Achtung! Der *char* Datentyp ist nicht mit *[un]signed char* zu verwechseln!

2.1.4. void

void ist ein unvollständiger Datentyp, der nie alleine auftrenen sollte. Er wird hauptsächlich für Funktionen oder Zeigern benutzt (*main* Methode z.B. von Beispiel 1.1).

2.1.5. bool

Bool ist ein sehr einfacher Datentyp, der nur zwei Werte (Zustände) annehmen kann: true und false. Er wird folgendermassen definiert und initialisiert:

Beispiel 7

```
bool allDone = true;
```

2.2.Konstanten

Man kann alle Variablen auch als Konstanten definieren: dazu benutzt man *const* vor der Definition:

Beispiel 8

```
const char dasIstEineKonstante = ´a´;
```

Die Konstante wird dann nicht mehr verändert.

2.3.Variablen und Zeiger

Jede Variable wird intern im Computer irgendwo im Speicher angelegt. Bei der Definition reserviert der Compiler Platz für sie und merkt sich ihre Adresse im Speicher, damit er sie wiederfinden kann. Diese Adressen kann man aber auch für ein paar nützliche Sachen benutzen, und nicht nur intern für den Compiler. Daher gibt C++ (wie fast alle Programmiersprachen auch) die Möglichkeit, mit Zeigern zu arbeiten.

Ein Zeiger (Pointer) soll auf keinen Fall mit dem Wert der Variable verwechselt werden. Z.B. für die Variable *variable* könnte gelten:

Adresse von *variable*: 145 000

Wert von *variable*: 3.6

Das sind also zwei verschiedene Sachen. Einmal wird die absolute Adresse der Variablen im Speicher ermittelt und einmal der Wert, der an dieser Adresse gespeichert ist.

Zeiger zu einer Variablen werden folgendermassen definiert:

Beispiel 9

```
char* zeigerAufChar;
```

Dann gilt:

- zeigerAufChar – ist die Adresse einer Variable des Typs char
- *zeigerAufChar – ist der gespeicherte Wert

Und bei der Definition einer Variable:

Beispiel 10

```
char variableChar;
```

gilt:

- variableChar – ist der Wert der Variable
- &variableChar – ist ihre Adresse

Hier ist ein Beispiel, das die Definitionen und ihre Wirkungen anschaulicher macht:

Beispiel 11

```
#include <iostream.h>

void main()
{
   int a=5;// int a ist 5
   cout<<a<<endl;// Ausgabe: 5

   int *pa=&a;// int Zeiger pa ist Adresse von a
   *pa=6;// Inhalt von pa ist 6
   cout<<a<<endl;// Ausgabe: 6
```

```
   a=7;// a ist 7
   cout<<*pa<<endl;// Ausgabe: 7

   int b=8;// int b ist 8
   int *pb=pa;// int Zeiger pb ist pa, pb zeigt jetzt auch auf a
   cout<<*pb<<endl;// Ausgabe: 7

   int **ppa=&pa;// int Zeiger Zeiger ppa=Adresse von pa;
   cout<<**ppa<<endl;// Ausgabe: 7

   ppa=&pb;// Inhalt von ppa ist Adresse von pb;
   cout<<**ppa<<endl;// Ausgabe: 7

   pb=&b;// pb ist Adresse von b
   cout<<**ppa<<endl;// Ausgabe: 8
};
```

2.4.Arrays, Arrays von Arrays

Arrays sind auch als Datenfelder bekannt. Ein Array oder ein Datenfeld ist einfach eine Variable, die genaugenommen aus mehreren Variablen vom gleichen Typ besteht. Anstatt nun aber jede einzelne Variable über den Variablennamen anzusprechen, wird das gesamte Feld über einen Variablennamen angesprochen und die einzelnen Variablen darin über einen Index.

Die Definition und die Benutzung von einem Array sind im folgenden Beispiel anschaulich gemacht:

Beispiel 12

```
include <iostream>
void main() {
   int i[3];

   i[0] = 10;
   i[1] = 15;
   i[2] = 20;

   std::cout << i[0] << „ „ << i[1] << „ „ << i[2] << „\n“;
}
```

Wichtig ist zu merken, dass man bei der Definition in den eckigen Klammern die Anzahl der Felder angibt (absolute Anzahl), aber dass die Indices von 0 bis n-1 gehen.

Interessant sind noch die *char*-Felder: die Vorgänger von *strings*. Sie enthalten, wie auch alle andere Felder, *chars*, enden aber mit einer binärischen Null, die als Zeichen für das Ende des Strings dient.

Bemerkung: die binärische und die ASCII-Null sind verschieden: ihre Nummern in der ASCII Tabelle unterscheiden sich. Die binärische Null wird mit \0 gekennzeichnet.

Hier ein Beispiel für einen char-Feld:

Beispiel 13

```
#include <iostream>

int main()
{
  char c[6];
  c[0] = 'H';
```

```
  c[1] = 'a';
  c[2] = 'l';
  c[3] = 'l';
  c[4] = 'o';
  c[5] = '\0';
  std::cout << c << std::endl;
}
```

2.5.Strings

Die C++ Standard Bibliothek stellt eine Klasse *string* zur Verfügung, mit der man Strings verwalten kann. Ein paar interessante Tatsachen:

- Es wird automatisch genügend Speicher reserviert um eine vorgegebene Zeichenfolge abzulegen. Und allein dies ist schon ausreichend Grund genug die *string* Klasse einzusetzen.
- Es werden diverse Operatoren überladen um Strings auf einfache Art und Weise verarbeiten zu können.
- Es stehen Konvertierungen zur Verfügung um ein *string* Objekt in einen *char** und umgekehrt zu konvertieren

Achtung! Es gibt Unterschied zwischen einem Char-Feld, der als **char* definiert wird und mit einer binären Null endet und einem *string*. *string* ist eine Klasse.

2.5.1. Definition von *string*

Damit man die Klasse string in seinem Code benutzen kann, muss man die dazugehörige Header-Datei einfügen:

Beispiel 14

```
#include <string>
```

Hier sind ein paar Beispiele für string-Definitionen:

Beispiel 15

```
// Leeren String definieren
std::string empty;

// String mit C-String initialisieren
std::string text1 = "Ein String";

// String Feld definieren
std::string array[10];

// und ersten String initialisieren
array[0] = "String-Element 0";

// String mit einem anderen String init.
std::string text2(text1);
```

Man kann natürlich auch Arrays (Felder) von *strings* definieren, man muss aber dabei beachten, dass ein Feld von n strings Strings mit unterschiedlicher Länge annimmt.

Bemerkung: *string* gehört, genauso wie *cout*, zu dem Namesbereich *std*. Das heisst, sie müssen entweder *std::string* benutzen oder am Anfang des Codes (nach den includes) folgende Anweisung schreiben:

Beispiel 16

```
using namespace std;
```

2.5.2. Ein- und Ausgabe von strings

Die Ausgabe von strings erfolgt wie bei den anderen Variablen mit *cout*:

Beispiel 17

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string text = "Ein string-Objekt";
    cout << text << endl;
}
```

Die Eingabe erfolgt auf die bekannte Art mit *cin*:

Beispiel 18

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string word1, word2;
    cin >> word1 >> word2
}
```

2.5.3. Operationen mit Strings

Den einfachsten Operator haben wir schon gesehen: die Zuweisung:

Beispiel 19

```
// Zwei Strings definieren
string s1, s2;
// C-String-Literal einem String zuweisen
s1 = "Dies ist String1";
// String einem anderen String zuweisen
s2 = s1;
```

Man kann auch sehr einfach zwei Strings „addieren" – nacheinanderhängen:

Beispiel 20

```
// Zwei Strings definieren
string s1("Text1"), s2;
// C-String-Literal addieren
s1 += " erweitert";
// Strings addieren
s2 = s1 + "in s2"
```

Die anderen sehen wir später bei den Operatoren.

2.6.Casting, Austausch von Information zwischen Variablen

Man kann natürlich Information auch zwischen verschiedenen Typen austauschen. Hier werden wir nur ein paar einfache Beispiele sehen, die man oft in der Praxis braucht:

Beispiel 21

```
short var1 = 10;
short var2 = 4;
float result1, result2;

result1 = (float)var1 / (float)var2;
result2 = (float)(var1 / var2);
```

2.7.sizeof

Man kann den durch eine Variable belegten Speicher (in Bytes) jederzeit mittels dem Operator *sizeof(datentyp)* ermitteln. Hier ein Beispiel:

Beispiel 22

```
#include <iostream>

void main() {

   float var_1;
   int var_2;
   char var_3;

   std::cout << "Ein float hat : " << sizeof(var_1) << " Bytes \n";
   std::cout << "Ein int hat : " << sizeof(var_2) << " Bytes \n";
   std::cout << "Ein char hat : " << sizeof(var_3) << " Bytes \n";

   std::cout << "Ein long int hat : " << sizeof(long) << " Bytes \n";
   std::cout << "Ein short hat : " << sizeof(short) << " Bytes \n";
   std::cout << "Ein double hat : " << sizeof(double) << " Bytes \n";
}
```

2.8.Aufgaben

2.8.1. Definieren Sie Variablen aus allen ganzzahligen Typen. Geben Sie sie einzeln auf dem Bildschirm aus und geben Sie zu jeder Variable die Grösse des belegten Speichers mit Hilfe von sizeof().

2.8.2. Definieren Sie Variablen aus allen fliesskommazahligen Typen. Geben Sie sie einzeln auf dem Bildschirm aus und geben Sie zu jeder Variable die Grösse des belegten Speichers mit Hilfe von sizeof().

2.8.3. Definieren Sie einen Array aus Zeigern zu den in Aufgabe 2.8.1 definierten Variablen. Definieren Sie noch einen Array, der die Adressen dieser Zeiger beinhaltet. Geben Sie beide Arrays aus und die Variablenwerte zuerst mit Hilfe des einen Arrays, dann mit Hilfe des anderen.

2.8.4. Schreiben Sie ein Program, dass 5 Wörter aus der Standardeingabe einliest, zusammenfügt und als Satz ausgibt (vergessen Sie die Leerzeichen zwischen den Wörtern nicht!)