Informatik 1 - Zusammenfassung

Herbstsemester 2011
Giuseppe Accaputo
Rechnergestützte Wissenschaften
ETH Zürich

Übersicht Datentypen

Type	Typical Bit Width	Typical Range
char	1byte	-127 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	Range	0 to 65,535
signed short int	Range	-32768 to 32767
long int	4bytes	-2,147,483,647 to 2,147,483,647
signed long int	4bytes	same as long int
unsigned long int	4bytes	0 to 4,294,967,295
float	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	2 or 4 bytes	1 wide character

Pointer und Referenzen

```
int a = 10;
int &b = a; // Referenz
int *c = &a; // Pointer
```

Statische Arrays

Deklaration

```
int arr[1];
```

Initialisierung

```
int arr[2] = {1,2};
```

Partielle Initialisierung

```
int brr[3] = {1};
```

Initialisierung Array von `structs`

```
struct mytype{
      int one;
      double two;
      char* three;
};

mytype a[2] = {{1,2.0,"three"},{4,5.0,"six"}};
```

Array zu Array Zuweisung verboten

```
int a[3];
int b[3];

a = b; //Verboten
```

Workaround

```
int *a;
int b[3];

a = b; //Funktioniert
a = &b[0]; //Dasselbe
```

Dynamische Arrays

Deklaration, Initialisierung

```
int nrelements = 2;
int *arr = new int[nrelements];
```

Deletion

```
int *arr = new int[3];
//...
delete[] arr;
```

Array von Pointern / Array von Arrays

Initialisierung 1, NICHT DYNAMISCHwie

```
int * numbers[5];
```

Initialisierung 2

```
int length = 10;

int **a = new int*[10];

for(int i = 0; i < length; i++){
      a[i] = new int[length];
}
```

Deletion

```
for(int i = 0; i < length; i++){
      delete [] a[i];
}

delete[] a;
```

Pointer-Arithmetik für Arrays

Allgemein

Für `int a[10];` oder `int *a = new int[10];` gilt folgendes:

```
a+i == &a[i]

*(a+i) == a[i]
```

Pointer auf statisches Array

```
int *a;
int b[3];

a = b;
```

Pointer auf dynamisches Array

```
int *b;
b = new int[3];

int *a = b;
```

Zuweisungen via Pointer

Sei folgendes dynamische Array gegeben:

```
int *arr = new int[10];
```

Dann macht dieser Code dasselbe...	...wie dieser Code
`*a = 10;`	`a[0] = 10;`
`*(a+3) = 100;`	`a[3] = 100;`
`*a = *(a+3);`	`a[0] = a[3];`

Visualisierungen

Initialisierung

```
int *a = new int[4];
```

*a	*(a+1)	*(a+2)	*(a+3)

Wert-Zuweisung

```
*(a+1) = 10; // a[1] = 10;
```

*a	*(a+1)	*(a+2)	*(a+3)
	10		

Neuer Pointer auf Array

```
int *b = a+1;
```

	*b	*(b+1)	*(b+2)
*a	*(a+1)	*(a+2)	*(a+3)
	10		

Konsolenausgabe des aktuellen Zellenwerts

```
cout << *b << endl; //Prints: 10
```

Zugriff mittels negativem Index

```
*(b-1) = 20;
```

*(b-1)	*b	*(b+1)	*(b+2)
*a	*(a+1)	*(a+2)	*(a+3)
20	10		

Zellenadresse in neue Zelle schreiben

Schreibe die Adresse der ersten Zelle von a in die Zelle `*(b+1)`:

```
*(b+1) = (int)a;
```

34	35	36	37
*(b-1)	*b	*(b+1)	*(b+2)
*a	*(a+1)	*(a+2)	*(a+3)
20	10	34	

Stolperstein: `*(b++)`

```
a[0] = *(b++);
```

b wird erst auf der nächsten Zeile inkrementiert, d.h. in die Zelle `a[1]` wird der Wert von `b[0]` reingeschrieben:

*(b-1)	*b	*(b+1)	*(b+2)
*a	*(a+1)	*(a+2)	*(a+3)
10	10	34	

C-Strings

Initialisierungsvarianten

```
char *b = "Hello, World!";

char c[7] = {'H','e','l','l','o','!','\0'};

char d[4] = "Hey";

char e[] = "Hello"; //Länge: 6
```

Nullterminierung

Weil ein C-String Nullterminiert wird (\0 als letztes Zeichen des C-String), muss bei der Deklarierung eines char Arrays immer Platz reserviert werden für das Nullterminierungs-Zeichen \0:

```
char i[4] = "Hey"; //Funktioniert
char j[3] = "Hey"; //Fehler; zu wenig Platz reserviert
```

Durch C-String iterieren

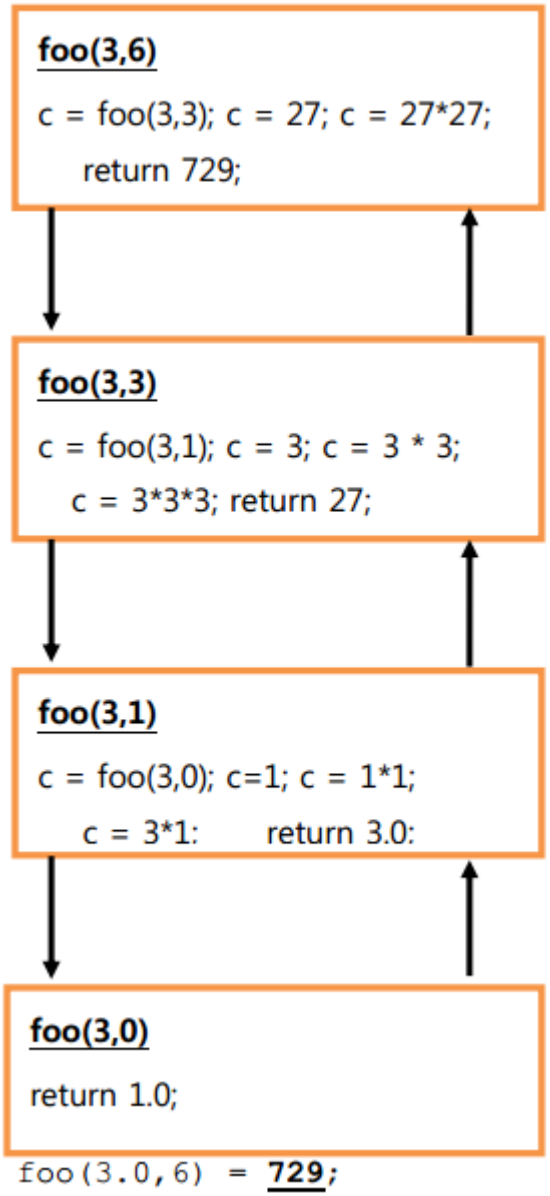
```
char *e = "Iteration";
for(int i = 0; e[i]; i++){
      cout << e[i];
}
```

Rekursion

Gute Übersicht

```
double foo(double a, unsigned int b)
{
      if (b == 0) return 1.0;
      double c = foo(a,b/2);
      c = c*c;
      if (b % 2 != 0) c = a*c;
      return c;
}
```

Aufruf von foo(3,6)



foo(3.0,6) = **729**;

Beispiele

Umgekehrte Polnische Notation

```
int evaluate_(char* argv[], int& ci){
	int c1= 0; int c2 = 0;
	c1 = ci+1; c2 = ci +2;

	if(strcmp(argv[ci], "+") == 0){
		return evaluate_(argv, c1) + evaluate_(argv,
c2);

	}
	else if(strcmp(argv[ci], "*") == 0){
		return evaluate_(argv, c1) * evaluate_(argv,
c2);

	}
	else{
		return atoi(argv[ci]);
	}

}
```

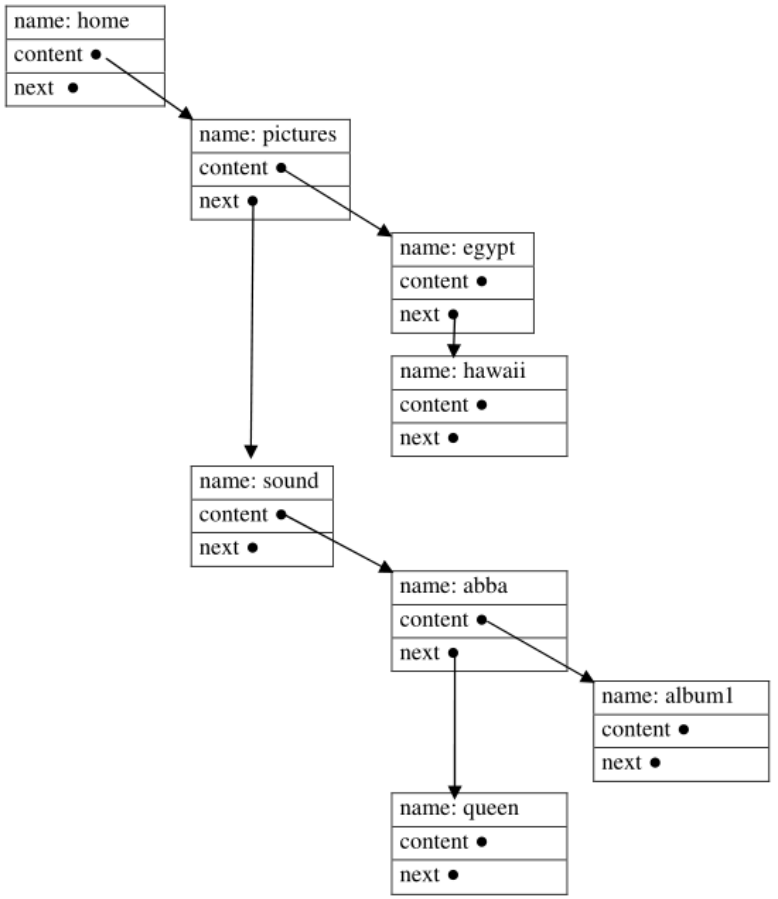
Aufruf

```
char * upn [5]= {"*", "2", "+", "3", "5"};

int counter = 0;
cout << evaluate_(upn, counter);
```

Ordnerstruktur löschen

Ordnerstruktur

```
struct folder{
	string name;
	folder* content;
	folder* next;
};
```



Deletion

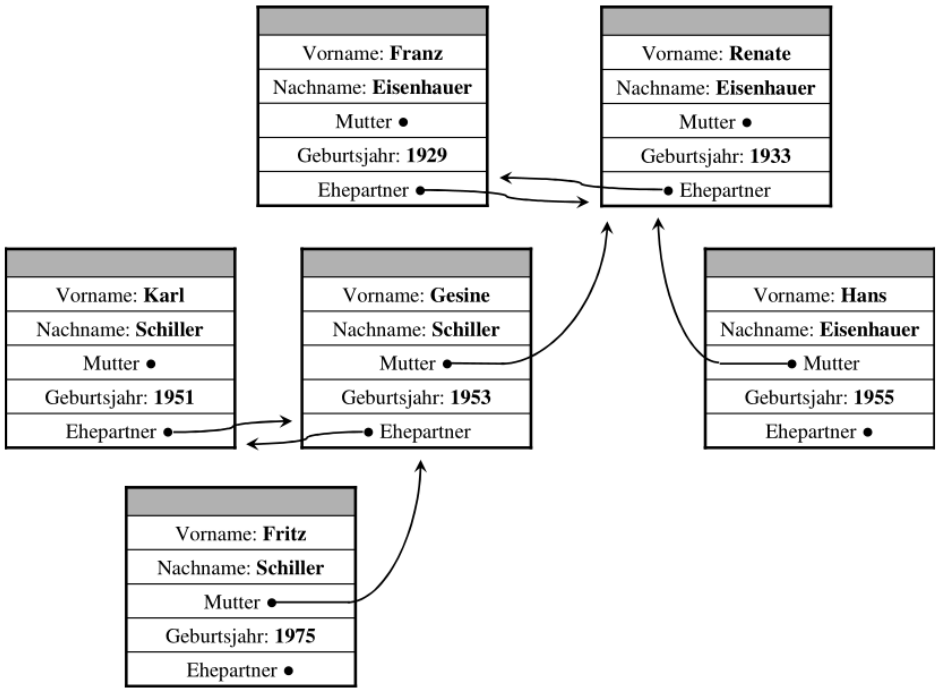
```
void clear(folder* f){
	if(f==NULL)
		return;
	else{
		clear(f->content);
		clear(f->next);
		delete f->content;
		f->content = NULL;
```

```
            delete f->next;
            f->next = NULL;
      }
}
```

Männliche Vorfahren ausgeben

Familienbaum

```
struct person{
    string Vorname;
    string Nachname;
    int Geburtsjahr;
    person* Mutter;
    person* Ehepartner;
};
```



Männliche Vorfahren ausgeben

```
void vorfahre(person * p, bool isMother = false){
      if(p==NULL){
            return;
      }
      if(p->Ehepartner!=NULL && isMother){
            cout << p->Ehepartner->Nachname << endl;
      }
      vorfahre(p->Mutter, true);

      if(isMother)
            vorfahre(p->Ehepartner);
}
```

Operator Überladung

operator<< (Chaining)

```
CharCounter& operator<<(CharCounter& c, const char* str){
      c.count(str);
      return c;
}
```

Der folgende Code ermöglicht nun die folgende Verkettung solcher Aufrufe:

```
CharCounter CC;
CC << "Hello" << "World";
//operator<<(operator<<(CC,"hello"),"test");
```

operator*(+,-,/)

Innerhalb der Klasse

```
T T::operator*(const T& b) const;
```

Ausserhalb der Klasse

```
T operator*(const T &a, const T &b);
```

Beispiel

```
Time Time::operator+(const Time & t) const
{
      Time sum;
      sum.mii t nutes = mii t + nutes + tt i t .minutes;
      sum.hours = hours + t.hours + sum.minutes / 60;
      sum.minutes %= 60;
      return sum;
}
```

Operatoren Priorität und Assoziativität

Operator level	Associativity
`::`	left to right
`( ) [ ] -> .`	left to right
`! ~ ++ -- +` (unary) `-` (unary) `*` (deference) `&` (address of) `new delete (data type)`	right to left
`* / %`	left to right
`+ -`	left to right
`<< >>`	left to right
`> <= > >=`	left to right
`== !=`	left to right
`&`	left to right
`|`	left to right
`&&`	left to right
`||`	left to right
`= += -= *= /= %=`	right to left

abnehmende Priorität

Funktionen

Funktionen und Arrays

Arrays als Parameter einer Funktion werden immer by Reference übergeben, d.h. es wird nur die Anfangsadresse des Arrays übergeben.

Dies bedeutet, dass folgende Funktionsdefinition dieselbe ist wie...

```
void f(int a[]){//...
```

folgende Funktionsdefinition:

```
void f(int* a){//...
```

Funktionenpointer

```
void run(double k, double (*fun)(double d)){
      //...
      fun(k);
}

double do(double d){//...

run(123, do);
```

const

Pointer

```
const int x = 5;
x = 6;//ILLEGAL

const int * y; //Read-only Pointer
*y = 3;//ILLEGAL
y = &x;

int * const z;//Konstanter Pointer auf Variable
*z = 4;
z = &y;//ILLEGAL

const int * const a;
*a = 5;//ILLEGAL
a = &z;//ILLEGAL
```

Funktion

```
void myclass::f() const{//...
```

`f` verändert das `myclass` Objekt auf welchem die Funktion aufgerufen wurde nicht. `f` ist nur eine Read-Only Funktion.

Funktionsparameter

```
void f(const double a[]){//...
```

`f` kann nur lesend auf die Daten (hier ein Array) zugreifen.

Weiter kann `f` konstante und nicht-konstante Daten verarbeiten, d.h. Originaldaten müssen nicht `const` sein.

Rückgabewert

```
const char *f(){
      return "Hello, World!";
}
```

Folgendes ist dann nicht mehr möglich:

```
f()[1] = 'a'; //Fehler
```

Filehandling

Dateien lesen

Wichtiges!

Der `ifstream`-Konstruktor akzeptiert nur C-Strings, d.h. falls man eine `string`-Variable dem `ifstream` Konstruktor übergeben sollte, muss man die `c_str()`-Funktion aufrufen:

```
ifstream input(s.c_str());
```

Zahlen auslesen

numbers.txt

```
1234  3.14159
1235  4.12413
1236  5.23141
```

Code

```
ifstream input("input.txt");

if(!output.is_open()){ return 1; }

int c = 0;
double d = 0;

while(!input.eof()){
	input >> c >> d;

	cout << c << ": " << d << endl;
}

input.close();
```

Ausgabe

```
1234: 3.14159
1235: 4.12413
1236: 5.23141
```

Zeilenweise Text auslesen

text.txt

```
This is some text,
on a new line
even
```

Code

```
ifstream input("text.txt");

if(!output.is_open()){ return 1; }

string s;

getline(input, s);

while(input){
	cout << s << endl;
	getline(input, s);
}

input.close();
```

Ausgabe

```
This is some text,
on a new line
even
```

Dateien lesen: Beispiele

Votes zählen

```
void countVotes(string f, double votes[], int len){
	ifstream input(f.c_str());
	int kreis = 0;
	int kand = 0;
	int stimmen = 0;
```

```
	int summe = 0;
	double votestmp[4] = {0,0,0,0};
	while(!input.eof()){
		input >> kreis >> kand >> stimmen;
		votestmp[kand]+= stimmen;
		summe += stimmen;
	}

	for(int i = 0; i < len; i++){
		votes[i] = votestmp[i]*100 / summe;
	}

	input.close();

}
```

Binäre Datei lesen

```
WAVEHEADER *readWaveHeader(ifstream *fin) {

	WAVEHEADER *hdr = NULL;

	//benutze die read Funktion um binaere Daten zu lesen
	hdr = new WAVEHEADER;

	fin->read((char*)hdr, sizeof(WAVEHEADER));

	if(hdr->Subchunk1Size==16 && hdr->SampleRate==44100 &&
hdr->BitsPerSample==16 && hdr->NumChannels==1){
		printWaveHeaderInfos(hdr);
		return hdr;
	}
	else{
		cerr << "Fehler: Falsche Eigenschaften!" <<
endl;
		return NULL;
	}


}
```

Dateien schreiben

```
ofstream output("text.txt");

if(!output.is_open()){
	cout << "ERROR FILE OPEN" << endl;
	return 1;
}

string s;

output << s;

output.close();
```

Binäre Datei schreiben

```
bool writeWaveHeader(ofstream *fout, WAVEHEADER *hdr) {

	if(NULL == hdr)
		return false;

	fout->write((char*)hdr, sizeof(WAVEHEADER));

	if(fout->eof() || fout->bad() || fout->fail())
		return false;

	return true;
}
```

Klassen

Header- und cpp-Datei

QueueElement.h:

```
#ifndef QUEUEELEMENT_H_
#define QUEUEELEMENT_H_

#include <string>

using namespace std;

class QueueElement{
      private:
            string value;
            QueueElement* next;
      public:
            QueueElement(string v);
            string GetValue();
            QueueElement * GetNext();
            void SetNext(QueueElement * n);
};

#endif
```

QueueElement.cpp:

```
#include <string>
#include "QueueElement.h"

using namespace std;

QueueElement::QueueElement(string v){
      this->value = v;
      this->next = NULL;
}

string QueueElement::GetValue(){
      return this->value;
}

QueueElement* QueueElement::GetNext(){
      return this->next;
}

void QueueElement::SetNext(QueueElement* n){
      this->next = n;
}
```

Kompilierung, Linken, etc.

Nur kompilieren, ohne assemblen:

```
g++ -c file.cpp
```

Objektdateien linken (produziert ein exe):

```
g++ -o progname obj1.o obj2.o ...
```

Sonstiges

Angepasster Insertionsort

```
void sortPages(page * pages[], int len){
      for(int i = 0; i < len; i++) {
        int j = i;
        page *v = pages[i];
        while(j > 0 && v->score <= pages[j-1]->score) {
            if(v->score == pages[j-1]->score){
                  if(v->nlinks < pages[j-1]->nlinks)
                        pages[j] = pages[j-1];
            }
            else{
                  pages[j] = pages[j-1];
            }
          j--;
        }
        pages[j] = v;
```

```
	}
}
```

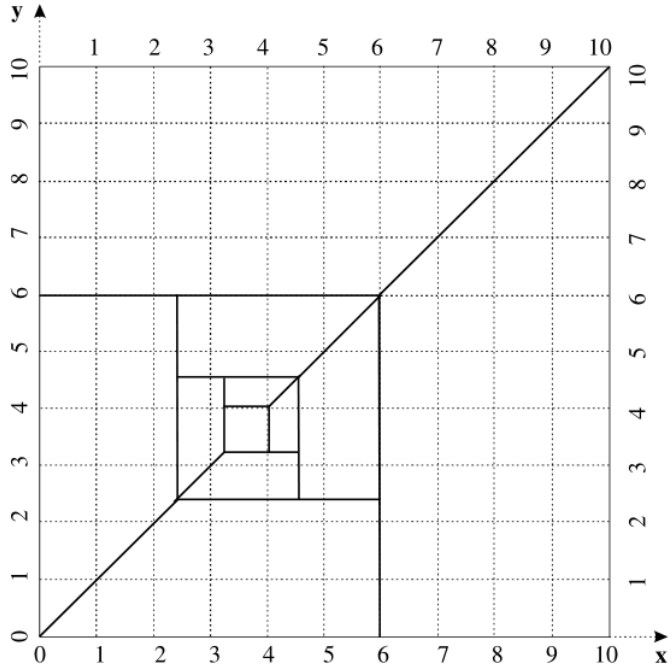
Binomialkoeffizienten: Iterativ

```
int binom_iterative(int n, int k){
	int nz = 1;
	for(int i = 0; i < (n-k); i++){
		nz *= (n-i);
	}

	int kz = 1;
	for(int i = (n-k); i > 0; i--){
		kz*= i;
	}
	return nz / kz;
}
```

Binomialkoeffizienten: Rekursiv

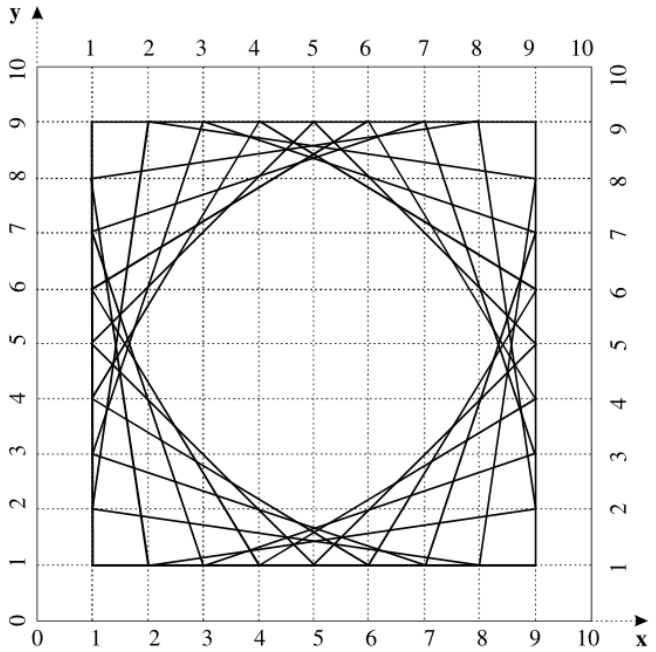
```
int binom_recursive(int n, int k){
	if(n == 0 && k == 0 || k == n || k == 0)
		return 1;

	return binom_recursive(n-1, k-1) + binom_recursive(n-
1, k);
}
```

Fraktal




```
void figure(double x, double y, double size)
{
	if(abs(size)<1){return;}
	double newsize=0.6*size;
	line(x,y+newsize,x+newsize,y+newsize);
	line(x+newsize,y,x+newsize,y+newsize);
	line(x+newsize,y+newsize,x+size,y+size);
	figure (x+newsize,y+newsize,-newsize);
}
```

Geometrisches




```
for(int i = 1; i <10;i++){
      line(i,1,9,i);
      line(i,1,1,10-i);
      line(i,9,9,10-i);
      line(i,9,1,i);
}
```

Zeichen zählen

```
void CharCounter::count(const char* str){
      for(int i=0;str[i]; i++){
            if(str[i]>= first &&str[i] <=last){
                  counts[str[i]-first] = counts[str[i]-
first] + 1;
            }
      }
}
```