Aufgabe 1: Typen und Werte (Basistypen) (6P)

Geben Sie für jeden der Ausdrücke auf der nächsten Seite jeweils C++Typ und Wert an. Wenn der Wert nicht bestimmt werden kann, schreiben Sie "undefiniert".
Die verwendeten Variablen sind wie folgt deklariert / initialisiert.

```
int i = 10;
double d = 2;
int m = 7;
int pre = 1;
int post = 1;
unsigned int u = 0;
```

(a) `1 / 4 * d` /1P

Typ / *Type* | Wert / *Value*

(b) `m % m` /1P

Typ / *Type* | Wert / *Value*

(c) `i = 10` /1P

Typ / *Type* | Wert / *Value*

(d) `i += i` /1P

Typ / *Type* | Wert / *Value*

(e) `++pre / post--` /1P

Typ / *Type* | Wert / *Value*

(f) `u-5 < 6` /1P

Typ / *Type* | Wert / *Value*

Aufgabe 2: Konstrukte (8P)

Geben Sie zu folgenden Codestücken jeweils die erzeugte Ausgabe an.

/2P (a)

```
int a[] = {1,2,3};
int* b = a;
std::cout << *(b+*(b+1));
```

Ausgabe / *Output*:

/2P (b)

```
int i = 3;
int& j = i;
++j;
std::cout << j + i;
```

Ausgabe / *Output*:

/2P (c)

```
char s[] = "DTI";
char* n = s;
int t[] = {1,0,-1};
int* i = t;
*(n++) += *(i++);
*(++n) += *(++i);
std::cout << s;
```

Ausgabe / *Output*:

/2P (d)

```
struct S{
    S* n;
    int v;
    S(int V, S* N): n(N), v(V) {};
};

S s = S(10, new S(20, &s));
std::cout << (s.n->n->v);
```

Ausgabe / *Output*:

Aufgabe 3: Zahlendarstellungen (8P)

(a) Geben Sie ein möglichst knappes normalisiertes Fliesskommazahlensystem an, mit welchem sich die folgenden dezimalen Werte gerade noch genau darstellen lassen: jede Verkleinerung von p, $e_{\max}$ oder $-e_{\min}$ muss dazu führen, dass mindestens eine Zahl nicht mehr dargestellt werden kann.
Hinweis: p zählt auch die führende Ziffer.
Tipp: Schreiben Sie sich die normalisierte Binärzahldarstellung der Werte auf, wenn sie für Sie nicht offensichtlich ist.

/3P

Werte / *Values*: 2.25, $\frac{1}{8}$, 0.5, 16.5, 2^3

$F^*(\beta, p, e_{\min}, e_{\max})$ mit / *with*

$\beta = 2$, $p =$, $e_{\min} =$, $e_{\max} =$.

(b) Markieren die Werte, die eine exakte Darstellung in einem beliebigen (endlichen) normalisierten binären Fliesskommazahlensystem besitzen.

/3P

◯ 123.4 ◯ 0.025 ◯ 1/10 ◯ 1000.5 ◯ 1/16 ◯ 1.5/32

(c) Die Zahl in der linken Spalte der nachfolgenden Tabelle ist jeweils als Literal der Sprache C++zu verstehen. Berechnen Sie, was jeweils verlangt ist.

/2P

0x7FF	Dezimalzahl / *decimal number*	=	
-16	Binärzahl mit 6 Bits im Zweierkomplement / *Binary number with 6 bits in two's complement*	=	

Aufgabe 4: EBNF I (6P)

Die folgende EBNF definiert erlaubte Anweisungen einer vereinfachten Programmiersprache.
Beantworten Sie die Fragen auf der rechten Seite.
Anmerkung: Leerschläge sind im Rahmen der EBNF bedeutungslos.

```
Statement      = Expression ';'.
Expression     = Simple [':' Simple].
Simple         = Designator | Integer.
Designator     = Identifier {'.' Identifier | '(' [ExpressionList] ')'}.
ExpressionList = Expression { ',' Expression }.

Integer        = Digit {Digit}.
Digit          = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' .

Identifier     = Letter {Letter}.
Letter         = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' .
```

(a) Folgende Zeichenkette entspricht einer gültigen Anweisung (statement) gemäss der EBNF: /1P

```
a(2:5);
```

◯ wahr / *true* ◯ falsch / *false*

(b) Folgende Zeichenkette entspricht einer gültigen Anweisung (statement) gemäss der EBNF: /1P

```
a(b).c:d;
```

◯ wahr / *true* ◯ falsch / *false*

(c) Folgende Zeichenkette entspricht einer gültigen Anweisung (statement) gemäss der EBNF: /1P

```
a(3).3;
```

◯ wahr / *true* ◯ falsch / *false*

(d) Folgende Zeichenkette entspricht einer gültigen Anweisung (statement) gemäss der EBNF: /1P

```
a(3):3;
```

◯ wahr / *true* ◯ falsch / *false*

(e) Ändern Sie genau eine Produktionsregel der EBNF ab, so dass folgende Anweisung (statement) gültig wird. /2P

```
a3(c3):a4(c4);
```

Geänderte Zeile / *Modified line*:

Aufgabe 5: EBNF II (8P)

Auf der nächsten Seite ist Code abgebildet, mit welchem identifiziert werden soll, ob eine Zeichenkette eine im Sinne obiger EBNF gültige Anweisung (Statement) darstellt. Folgender Code, welcher nur als Funktionsdeklaration vorliegt, soll als korrekt implementiert vorausgesetzt werden.

```
// POST: when the next available non-whitespace character equals c,
//       it is consumed and the function returns true,
//       otherwise the function returns false.
bool has(std::istream& is, char c);

// Integer = Digit {Digit}.
bool Integer (std::istream& is);

// Identifier = Letter {Letter}.
bool Identifier (std::istream& is);
```

/2P (a) Die Funktion `Expression` wird im Code anscheinend zweimal deklariert. Erklären Sie warum.

/2P (b) Ergänzen Sie die Funktion `Designator`, so dass Sie die entsprechende EBNF-Zeile korrekt implementiert.

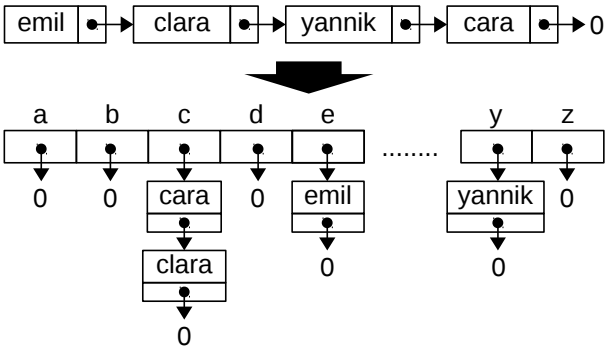
/2P (c) Ergänzen Sie die Funktion `ExpressionList`, so dass Sie die entsprechende EBNF-Zeile korrekt implementiert.

/2P (d) Ergänzen Sie die Funktion `Expression`, so dass Sie die entsprechende EBNF-Zeile korrekt implementiert.

```
bool Expression (std::istream& is);
// ExpressionList = Expression { ',' Expression }.
bool ExpressionList(std::istream& is){
    if (!Expression(is)) return false;
    
    return true;
}
// Designator = Identifier {'.' Identifier | '(' [ExpressionList] ')'}.
bool Designator(std::istream& is){
    if (!Identifier(is)) return false;
    while(true){
        if (has(is, '.')){
            
            ;
        } else if (has(is, '(')){
            bool ignore = ExpressionList(is);
            
            ;
        } else
            return true;
    }
}
// Simple = Designator | Integer.
bool Simple(std::istream& is){
    return Integer(is) || Designator(is);
}
// Expression = Simple [':' Simple].
bool Expression(std::istream& is){
    if (!Simple(is)) return false;
    return 
    ;
}
// Statement = Expression ';'.
bool Statement(std::istream& is){
    return Expression(is) && has(is, ';');
}
```

Aufgabe 6: Partial Bucketsort (8P)

Betrachten Sie das untenstehende Programm zum Teilsortieren einer Liste von Namen anhand ihres Anfangsbuchstabens. Die Namen bestehen ausschliesslich aus Kleinbuchstaben (`'a'-'z'`). Dabei werden die Elemente der Eingabe in Buckets (Gefässe) die ihrem Anfangsbuchstaben entsprechend einsortiert. Ein Array von verketteten Listen bildet die Buckets, wobei a=0, b=1, c=2, ... z=25. Die Eingabe erfolgt in der Form einer verketteten Liste L. Über jedes Element der Liste L wird von vorne her iteriert, dieses Element wird aus der Liste L entfernt und in die Liste des Zielbuckets zuvorderst einfügt.



/2P (a) Vervollständigen Sie die Funktion `add_first` entsprechend den angegebenen Vor- und Nachbedingung.

/3P (b) Vervollständigen Sie die Funktion `partial_bucketsort` mittels den Funktionen `remove_first` und `add_first`.

/3P (c) Geben Sie die Ausgabe des Programms an (betrachten Sie die Funktionen `main` und `print`).

```
#include<iostream>
#include<string>

struct Node {
    std::string name;
    Node* next;
    Node(std::string _name, Node* _next) : name(_name), next(_next) {}
};
```

```
struct List {
    Node* first;
    List() : first(0) {}
    // PRE: non-empty list
    // POST: Returns and removes first element of list
    Node* remove_first() {
        Node* node = first;
        first = node->next;
        return node;
    }
    // PRE: valid list
    // POST: Adds node as first element of the list
    void add_first(Node* node) {
        [                                                    ];
        [                                                    ];
    }
};
void partial_bucketsort(List buckets[], List& unsorted) {
    while(unsorted.first != 0) {
        Node* node = [                                       ];
        int bucket = [                                       ];
        [                                                ];
    }
}
void print(List buckets[]) {
    for (List* list = buckets; list != buckets + 26; ++list) {
        if (list->first != 0) {
            std::cout << list->first->name[0] << ": ";
            for(Node* node = list->first; node != 0; node = node->next)
                std::cout << node->name << " ";
        }
    }
}
int main() {
    List buckets[26]; // 'a' - 'z'
    List unsorted;
    unsorted.first = new Node("lara", new Node("lena",
                     new Node("albert", new Node("hubert", 0))));
    partial_bucketsort(buckets, unsorted);
    print(buckets);
    return 0;
}
```

Aufgabe 7: Referenztypen und Pointer (7P)

Das Programm auf der rechten Seite definiert eine Datenstruktur für Datumsangaben und stellt weitere Operationen auf dieser Struktur bereit. Beantworten Sie für dieses Programm untenstehende Fragen.

/4P (a) In der untenstehenden Tabelle sehen Sie zwei verschiedene Ausgaben des Programms, sowie Spalten mit Markierungen, die jenigen in der Klasse `DateList` entsprechen. Füllen Sie die Tabelle mit Typdeklarationen oder Ausdrücken, so dass das Programm die entsprechende Ausgabe erzeugt (pro Zeile).

Ausgabe: / *Output:*	**A**	**B**	**C**	**D**
`1.4.2001` `31.7.2013`				
`23.8.2017` `31.7.2013`				

/2P (b) Betrachten Sie die Funktion `createDate`, welche eine Datumstruktur deklariert, initialisiert und zurückgibt. Beantworten Sie folgende Frage (je ein Satz): (i) Welchen Fehler hat diese Funktion und (ii) wie könnte man die Funktion korrigieren?

(i)

(ii)

/1P (c) Betrachten Sie die Funktionen `f1`, `f2`, `f3`, welche jeweils eine Datumsstruktur als Parameter übernehmen. Der Parameter ist ein Referenztyp oder ein Pointer mit unterschiedlicher const-Eigenschaft. Markieren Sie alle Funktionen die korrekt kompiliern.

◯ `f1` ◯ `f2` ◯ `f3`

```
#include <iostream>
#include <vector>

struct Date {
  int day; int month; int year;

  void set(int d, int m, int y) { day = d; month = m; year = y; }
  void print() {
    std::cout << day << "." << month << "." << year << "\n";
  }
};

class DateList {
  std::vector< A > dates;
public:
  void add( B date) { dates.push_back( C ); }
  void print() {
    for(std::vector< A >::iterator it=dates.begin(); it!=dates.end(); ++it) {
      D .print();
    }
  }
};

int main() {
  DateList list; Date d1; Date d2;

  d1.set(1, 4, 2001);
  list.add(d1);
  d2.set(31,7, 2013);
  list.add(d2);
  d1.set(23, 8, 2017);

  list.print();
  return 0;
}

Date* createDate(int day, int month, int year) {
    Date d;
    d.set(day, month, year);
    return &d;
}

void f1(const Date& date) { date.year = 0; }

void f2(const Date* date) { date->year = 0; }

void f3(Date* const date) { date->year = 0; }
```