Aufgabe 1: Typen und Werte (Basistypen) (12P)

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”.

In manchen Teilaufgaben finden Sie den Ausdruck unter einer dünnen Linie. Darueber finden Sie eine oder mehrere Anweisungen, welche direkt vor dem Ausdruck ausgefuehrt wurden und welche relevant sind zur Bestimmung des Typs und Wertes des gefragten Ausdrucks.

(a) `4 <= 5.f` /2P

Typ / *Type* | Wert / *Value*

(b) `2 / 4.0f + 7 / 2.0` /2P

Typ / *Type* | Wert / *Value*

(c) /2P

```
int a = 4;
int d = 5;
```

```
a / d--
```

Typ / *Type* | Wert / *Value*

(d) `std::vector<float> f = { 1, 2, 3, 4, };` /2P

```
f[2] + (int)f[3]
```

Typ / *Type* | Wert / *Value*

(e) /2P

```
int b = 3;
int k = b++;
```

```
++k
```

Typ / *Type* | Wert / *Value*

(f) `(2u - 8 % 5) < 0` /2P

Typ / *Type* | Wert / *Value*

Aufgabe 2: Konstrukte (16P)

Geben Sie zu folgenden Codestücken jeweils die erzeugte Ausgabe an.

/4P (a)

```
int a[6] = {1, 3, 5, 7, 9, 11};
int* x = a;
int& k = *x;
x++;
std::cout << *(x+k);
```

Ausgabe / *Output*:

/4P (b)

```
float p[6] = {0.5, 1.0, 1.5, 2.0, 0.5, 0.0};
float* s = &p[4];
float* r = &p[2];
std::cout << (r[2] + *(s+1))
```

Ausgabe / *Output*:

/4P (c)

```
void my_function(int* m) {
    *m = 2;
}
int values[] = {1, 3, 5};
int* v = values;
my_function(v++);
std::cout << v[0];
```

Ausgabe / *Output*:

/4P (d)

```
struct Node {
   Node* next;
   int value;
   Node (int v, Node* n) : next(n), value(v) {};
};

Node s = Node(10, nullptr);
Node t = Node(20, &s);
s.next = &t;
std::cout << (s.next->next->value);
```

Ausgabe / *Output*:

Aufgabe 3: Zahlendarstellungen (9P)

(a) Markieren Sie die Werte, die eine exakte Darstellung in einem beliebigen (endlichen) normalisierten binären Fliesskommazahlensystem besitzen. /3P

◯ 0.150 ◯ 1.625 ◯ 0.4

(b) Die Zahl in der linken Spalte der nachfolgenden Tabelle ist jeweils als Literal der Sprache C++zu verstehen. Berechnen Sie, was jeweils verlangt ist. /3P

0x2A	als Dezimalzahl /	=
110101	die binäre Zahl (6 Bits, Zweierkomplement) als Dezimalzahl /	=
-17	die Dezimalzahl als binäre Zahl (7 Bits, Zweierkomplement) /	=

(c) 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. /3P
Hinweis: p zählt auch die führende Ziffer.
Tipp: Schreiben Sie sich die normalisierte Binärzahldarstellung der Werte auf, wenn sie für Sie nicht offensichtlich ist.

Werte / *Values*: 10, 1.625

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

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

Aufgabe 4: EBNF: Ein kleiner Parser (12P)

Die folgende EBNF definiert erlaubte Anweisungen einer vereinfachten Programmiersprache. Sie können annehmen, dass die ersten drei gegebenen Funktionsdeklarationen korrekt implementiert sind. Vervollständigen Sie die Funktionen in den Teilaufgaben.

Anmerkung: Leerschläge sind im Rahmen dieser EBNF bedeutungslos.

```
Expression      = Unit {'.' Unit}.
Unit            = Identifier {'[' Range ']' }.
Range           = Expression | Integer ':' Integer.

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

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

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

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

// Consumes next Identifier = Letter {Letter | Digit}
bool Identifier (std::istream& is);
```

/4P (a) Vervollständigen Sie die Funktion Expression gemäss der gegebenen EBNF.

```
// Expression  = Unit { '.' Unit }.
bool Expression(std::istream& is){
   if (!Unit(is)) return false;
```

```
   return true;
}
```

(b) Vervollständigen Sie die Funktion `Unit` gemäss der gegebenen EBNF. /4P

```
// Consumes next Unit = Identifier {'[' Range ']'}.
bool Unit(std::istream& is) {
    if (!Identifier(is)) return false;
    while(true) {
        if (consume(is, '[')) {
            
        } else
            return true;
    }
}
```

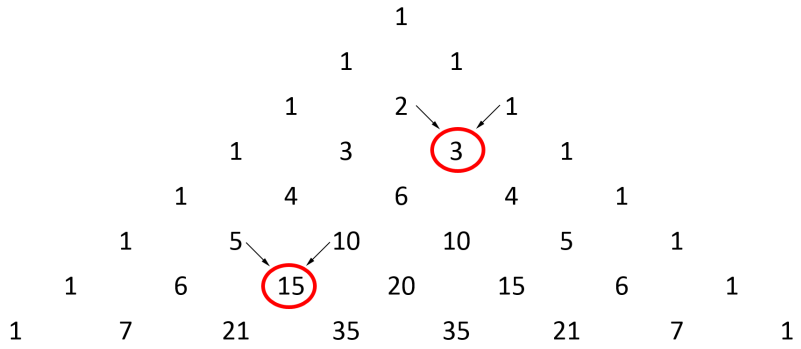
(c) Vervollständigen Sie die Funktion `Range` gemäss der gegebenen EBNF. /4P

```
//Consumes next Range = Expression | Integer ':' Integer
bool Range(std::istream& is) {
    if (Expression(is)) return 
    else if (Integer(is)) {
        
    }
    return false;
}
```

Aufgabe 5: Rekursion: Pascalsches Dreieck (13P)

Das Ziel dieser Aufgabe ist, die Zahl einer gegebene Position im Pascalschen Dreieck zu berechnen (siehe Bild). Verfolständigen Sie das gegebene Programm entsprechend.
Eine Zahl an einer gegebenen Position innerhalb des Dreiecks entspricht der Summe der beiden Zahlen links und rechts in der darüberliegenden Zeile. Alle Zahlen am linken und rechten Rand des Dreiecks haben den Wert 1 gesetzt.
Beispiel: Die eingekreiste 15 im untenstehenden Dreieck steht in der $7ten$ Zeile, an $3te$ Stelle. Gleichermassen: Die eingekreiste 3 steht in der $4ten$ Zeile, an $3ter$ Stelle.

Das Programm soll den Nutzer auffordern, eine Zeile und die Stelle in dieser Zeile einzugeben. Dabei wird **mit 1 angefangen** zu zählen. Ausserdem soll das Programm vor der Berechnung überprüfen, ob die eingegebene Positionskombination gültig ist.



/3P (a) Vervollständigen Sie die Funktion `main`.

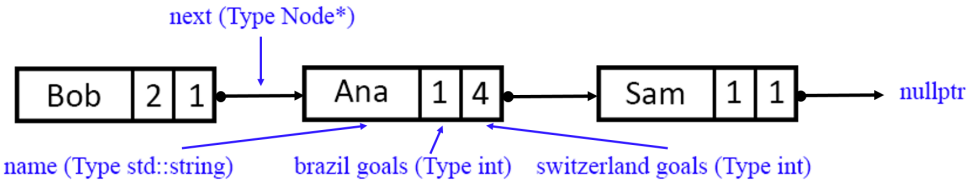
/5P (b) Vervollständigen Sie die Funktion `check_input_validity`. Sie können annehmen, dass die eingegebenen Variablen immer vom richtigen Datentyp sind.

/5P (c) Vervollständigen Sie die Funktion `compute_pascal` mit Hilfe von **Rekursion**.

```
#include <iostream>
#include <cassert>

                check_input_validity(                                ){
    if (                                                          ){
        std::cout << "Invalid: no element at the given row-position pair.";
                                                                  
    }
                                                                  
}

int compute_pascal(int row, int pos){
    if (pos == 1){
        return 1;
    }
                                                                  
}

void main(){

    //Input
    std:: cout<< "Please input a row and a position along the row: ";
                                                                  

    //Check whether the input integers correspond to a valid entry
     assert (check_input_validity(row, pos));

    //Display the result
    std::cout << "The value at row " << row << " and position " << pos << " is "
              << compute_pascal(row, pos);
}
```

Aufgabe 6: Datenstrukturen: WM Tipp-Spiel (15P)

Die Weltmeisterschaft ist vorbei und Sie und Ihre Freunde wollen die Wettresultate zum spannenden Spiel zwischen Brasilien und der Schweiz auswerten. Die abgegebenen Wetten sind in einer verketteten Liste abgelegt, implementiert mittels den untenstehenden `struct Node` und `struct List`.
Ziel des folgenden Programmes ist es, die erste Person in der Liste zu finden, die das Resultat richtig getippt hat.



Tasks

/5P (a) Vervollständigen Sie die Funktion `add_friend`, welche Wetten zur Liste hinzufügt.

/5P (b) Vervollständigen Sie die Funktion `find_winner`, welche durch die Liste geht und das passende Resultat sucht. Als Argumente wird das korrekte Resultat mitgeliefert.

/5P (c) Vervollständigen Sie die Initialisierung der Liste in der Funktion `main` so, dass sie dem Bild oben entspricht und geben Sie die Programmausgabe an in der Box unter dem Code.

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

struct Node {
    std::string name;
    int brazil; int switzerland;
    Node* next;
    Node(std::string _name, int b, int s) : name(_name), brazil(b),
        switzerland(s), next(nullptr) {}
};

struct List{
    Node* first;
    Node* last;
    List() : first(nullptr), last(nullptr) {}
```

```
    // PRE: Argument: a new bet (not null)
    // POST: Adds the argument as last element of the list
    void add_friend(Node* newBet){
        if(                                    ){
                                               ;
        }else{
                                               ;
        }
                                           ;
    }

    // PRE: The correct score in b (for brasil) and s (for switzerland)
    // POST: Returns the first node with the given scores, or nullptr otherwise
    Node* find_winner(int b, int s) {
        Node* curr = first;
        while(                                 ){
            if(                                                    ){
                return curr;
            }
                                               ;
        }
        return nullptr;
    }
};

int main () {
    List friends;
    friends.add_friend(                        );
    friends.add_friend(                        );
    friends.add_friend(                        );
    Node* winner = friends.find_winner(1,1);
    if(winner != nullptr){
        std::cout << winner->name << " won with match result: "
                << winner->brazil << "-" << winner->switzerland;
    }else{
        std::cout << "Nobody guessed it right!";
    }
    return 0;
}
```

Program output:

Aufgabe 7: Objektorientierung: Reiche Studenten (13P)

Bearbeiten Sie die folgenden Teilaufgaben.

/5P (a) Gegeben ist die folgende Klasse `Student`. Schreiben Sie eine Unterklasse `EconomyStudent`, welche die Methode `checkFinances()` überschreibt und die folgende Ausgabe auf der Konsole ausgibt: "I am making a lot of money."

```
class Student {
public:
    virtual void checkFinances() {
        std::cout << "I have no money.\n";
    }
};
```

/4P (b) Kompiliert der untenstehende Code? Falls ja, was ist die Ausgabe auf der Konsole wenn er ausgeführt wird? Falls nein, warum nicht? Die Klassen `Student` und `EconomyStudent` sind wie in der Teilaufgabe a) definiert.

```
EconomyStudent economyStudent;
economyStudent.checkFinances();
Student* student = &economyStudent;
student->checkFinances();
```

(c) Kompiliert der untenstehende Code? Falls ja, was ist die Ausgabe auf der Konsole wenn er ausgeführt wird? Falls nein, warum nicht? Die Klassen `Student` und `EconomyStudent` sind wie in der Teilaufgabe a) definiert. /4P

```
Student student;
student.checkFinances();
EconomyStudent* economyStudent = &student;
economyStudent->checkFinances();
```