1 Structs, Konstruktoren und New *(5 Punkte)*

Unten finden Sie ein Programm. Ihre Aufgabe ist es, die Ausgaben der angegebenen fünf Zeilen zu bestimmen. Tragen Sie die Antworten direkt in die vorgegebenen Lücken ein! Geben Sie zudem in der grösseren Box die Anweisungen zum Aufräumen des dynamische allozierten Speichers an.

Below you find a program. Your task is to determine the outputs of the five indicated lines. Fill your answers directly into the provided gaps! Moreover, provide in the larger box the statements for cleaning up the dynamically allocated memory.

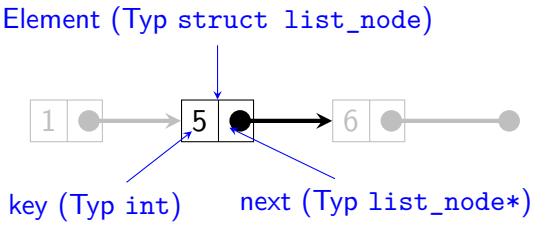
```
#include<iostream>

struct S {
  int v;
  S* n;
  S () : v(5), n(0) {}

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

int main(){
  S s;
  S* ps = new S (1, &s);
  S* pt = new S (9, new S (6, ps));
  std::cout << s.v;       // outputs 5
  std::cout << ps->v;     // outputs 1
  std::cout << pt->v;     // outputs 9
  std::cout << ps->n->v; // outputs 5
  std::cout << pt->n->v; // outputs 6
  // cleanup; delete dynamically allocated nodes:
  delete ps; delete pt->n; delete
  pt;

  return 0;
}
```

Diese Aufgaben zum weiteren Üben haben es wegen der Fülle nicht auf das Selfassessment geschafft. Viel Spass beim Lösen!

2 Dynamische Datentypen *(11 Punkte)*

Auf der nächsten Seite finden Sie das Skelett einer Klasse `list`, die eine Folge ganzer Zahlen verwaltet. Jedes Element ist durch ein Objekt vom Typ `list_node` repräsentiert, das den Wert (`key`) sowie einen Zeiger auf das nächste Element (`next`) speichert (0 im Fall des letzten Elements). Eine Liste ist durch einen Zeiger auf ihr erstes Element repräsentiert (0 im Fall einer leeren Liste). Ergänzen Sie die Lücken so, dass sich korrekte und speicherleckfreie Definitionen der zwei Memberfunktionen `last` und `remove_second` ergeben! Die Assertions sollen dabei genau die angegebenen Vorbedingungen prüfen.

```
struct list_node {
    int key;              // value of the node
    list_node* next;      // pointer to next node
    ..
};
```



On the next page you find a skeleton of a class `list` for maintaining a sequence of integers. Each element is represented by an object of type `list_node` that stores the value (`key`) and a pointer to the next element (`next`) which is 0 in case of the last element. A list is represented by a pointer to its first element (0 in case of an empty list). Fill the gaps such that you obtain correct and memory-leak-free definitions of the two member functions `last` and `remove_second`! The assertions shall exactly check the given preconditions.

```
class list {
private:
  list_node* first_node;
public:
  ...
  // PRE: *this is not empty
  // POST: the key of the last element in *this is returned
  int last() const
  {
    assert(first_node != 0);
    const list_node* p = first_node;
    while (p->next != 0)
      p = p->next;
    return p->key;
  }

  // PRE: *this contains at least two elements
  // POST: the second element is removed from *this
  void remove_second()
  {
      assert(first_node != 0);
      assert(first_node->next != 0);
      list_node* p = first_node->next;
      first_node->next =
          first_node->next->next;
      delete p;
  }
};
```