Podstawy Programowania 2

Jednokierunkowa i dwukierunkowa lista liniowa

Arkadiusz Chrobot

Katedra Systemów Informatycznych

24 kwietnia 2024

Plan

Wprowadzenie

Na poprzednim wykładzie omówiliśmy dwie struktury dynamiczne, stos i kolejkę. Są one szczególnymi przypadkami bardziej ogólnych struktur, jakimi są listy liniowe. To uogólnienie dotyczy operacji usuwania pojedynczego węzła i dodawania nowego. W przypadku listy mogą one być przeprowadzone w dowolnym jej miejscu. Z kolei przymiotnik „liniowa" oznacza, że każdy węzeł tej struktury może mieć co najwyżej jednego następnika lub poprzednika.
W ramach tego wykładu zajmiemy się dwiema odmianami list. Będą to, odpowiednio, jednokierunkowa lista liniowa (ang. *singly-linked list*) i dwukierunkowa lista liniowa (ang. *doubly-linked list*). Aby objaśnić ich budowę i działanie posłużymy się dwoma programy używającymi tych list do przechowywania w nich liczby naturalnych (jeden węzeł — jedna liczba), tak aby te liczby były uporządkowane w sposób niemalejący.

Jednokierunkowa lista liniowa

Typ danych węzła

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  struct list_node
5  {
6      int data;
7      struct list_node *next;
8  } *list_pointer;
```

Jednokierunkowa lista liniowa

Typ danych węzła

Pierwszy ze wspomnianych programów demonstruje użycie jednokierunkowej listy liniowej. W zamieszczonym na poprzednim slajdzie fragmencie jego kodu są włączane dwa pliki nagłówkowe `stdio.h` i `stdlib.h`. Pierwszy z nich umożliwia korzystanie z funkcji `printf()`, a drugi z funkcji zarządzających stertą. Lista jednokierunkowa zrealizowana w formie dynamicznej struktury danych wymaga określenia typu danych dla jej węzłów. Jego definicja, bardzo podobna do definicji typu danych dla węzłów kolejki i stosu, znajduje się w wierszach 4–8. Pole `data` będzie służyło do przechowywania liczby naturalnej, choć jego typ pozwala na przechowywanie liczb całkowitych, a pole `next` jest wskaźnikiem, dzięki któremu węzeł może być połączony z innym. W wierszu nr 8 zadeklarowany jest również *wskaźnik listy* (`list_pointer`). Ponieważ jest on zmienną globalną, to jego początkową wartością będzie `NULL`. Ta wartość oznacza także, że lista jest początkowo pusta.

Jednokierunkowa lista liniowa

Dwa kolejne slajdy zawierają kody funkcji, które razem są odpowiedzialne za dodanie nowego węzła do listy, tak aby liczby przechowywane w niej były posortowane niemalejąco. Dzięki zastosowaniu w nich parametru będącego wskaźnikiem na wskaźnik mogą one obsłużyć dowolny z czterech możliwych przypadków dodania nowego elementu do jednokierunkowej listy liniowej:

1. dodanie do pustej listy,
2. dodanie na początku listy,
3. dodanie wewnątrz listy,
4. dodanie na końcu listy.

Aby to sprawdzić, oprócz ogólnego opisu budowy tych funkcji, przeanalizujemy ich zachowanie dla każdego z wymienionych przypadków.

Jednokierunkowa lista liniowa

Funkcja `create_and_add_node()`

```
1  int create_and_add_node(struct list_node **node, int
    ↪  number)
2  {
3      struct list_node *new_node = (struct list_node
    ↪  *)malloc(sizeof(struct list_node));
4      if(!new_node)
5          return -1;
6      new_node->data = number;
7      new_node->next = *node;
8      *node = new_node;
9      return 0;
10 }
```

Jednokierunkowa lista liniowa

Funkcja `create_and_add_node()`

Funkcja `create_and_add_node()` odpowiada za utworzenie nowego węzła i dodanie go do listy. Przyjmuje ona dwa argumenty. Pierwszym, przekazywanym przez parametr `node`, jest *adres wskaźnika na węzeł listy*, przed którym należy wstawić nowy. Drugim jest liczba, która ma być przechowywana w tym elemencie. Najpierw funkcja przydziela pamięć na węzeł (wiersz nr 3), a następnie sprawdza, czy ta operacja się powiodła (wiersz nr 4). Jeśli nie, to zwraca wartość `-1` i kończy działanie (wiersz nr 5). W przeciwnym przypadku zapisuje w polu `data` węzła liczbę, która ma być w nim umieszczona (wiersz nr 6), a potem do pola `next` zapisuje adres węzła *przechowywany we wskaźniku wskazywanym przez* `node` (wiersz nr 7). Na koniec, we wskaźniku wskazywanym przez `node` zapisywany jest adres nowego węzła (wiersz nr 8) i funkcja kończy działanie zwracając `0`.

Jednokierunkowa lista liniowa

Funkcja `add_node()`

```
1  int add_node(struct list_node **node, int number)
2  {
3      while(*node != NULL && (*node)->data < number)
4          node = &(*node)->next;
5      return create_and_add_node(node, number);
6  }
```

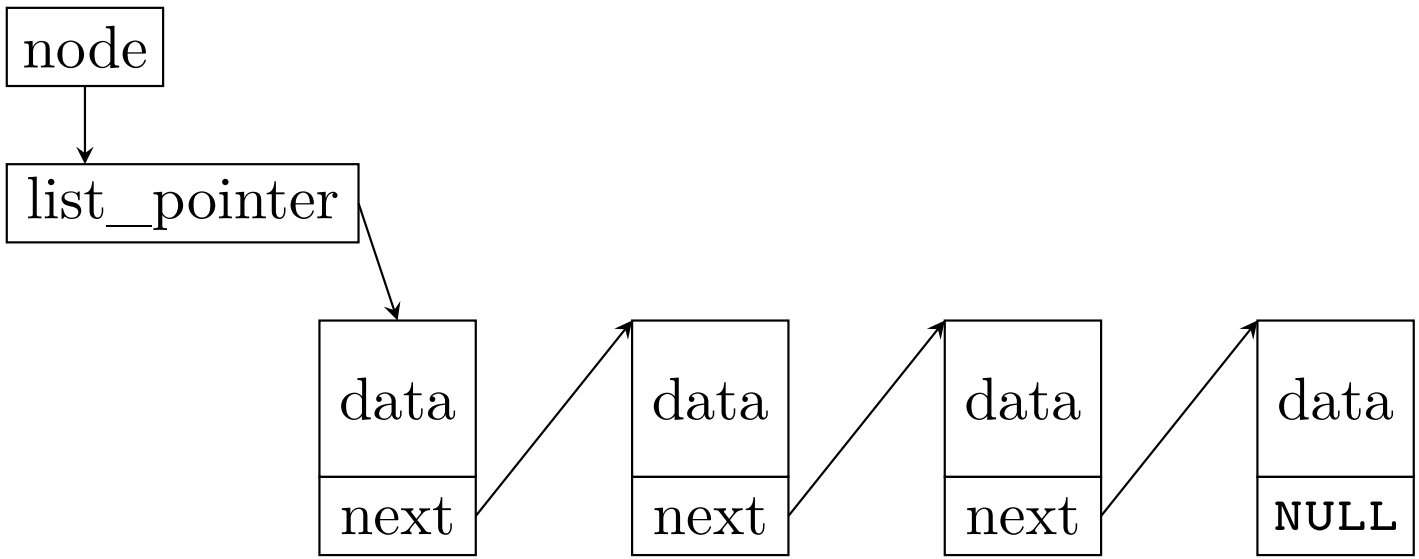
Jednokierunkowa lista liniowa

Funkcja `add_node()`

Funkcja `create_and_add_node()` jest wywoływana w programie tylko z poziomu funkcji `add_node()`. To ta ostatnia powinna być używana w celu dodania nowego węzła do listy. Jako argumenty przyjmuje ona adres wskaźnika listy, który jest przekazywany przez parametr `node` oraz liczbę, przekazywaną przez parametr *number*, która ma być zapisana w nowym węźle. Najpierw wykonuje ona pętlę `while`, celem znalezienia węzła w liście, *przed którym* trzeba wstawić nowy (wiersze 3–4). Ta pętla może się zakończyć wtedy, gdy wskaźnik wskazywany przez `node` będzie pusty lub gdy liczba w węźle wskazywanym przez wskaźnik, którego adres znajduje się w `node` jest większa lub równa tej, którą została przekazana przez parametr `number`. Proszę zwrócić uwagę, że w kolejnych iteracjach `while` do parametru wskaźnikowego `node` zapisywany jest *adres pola* `next` kolejnego węzła w liście. Sposób iterowania po tej liście ilustruje rysunek na kolejnym slajdzie.

Jednokierunkowa lista liniowa

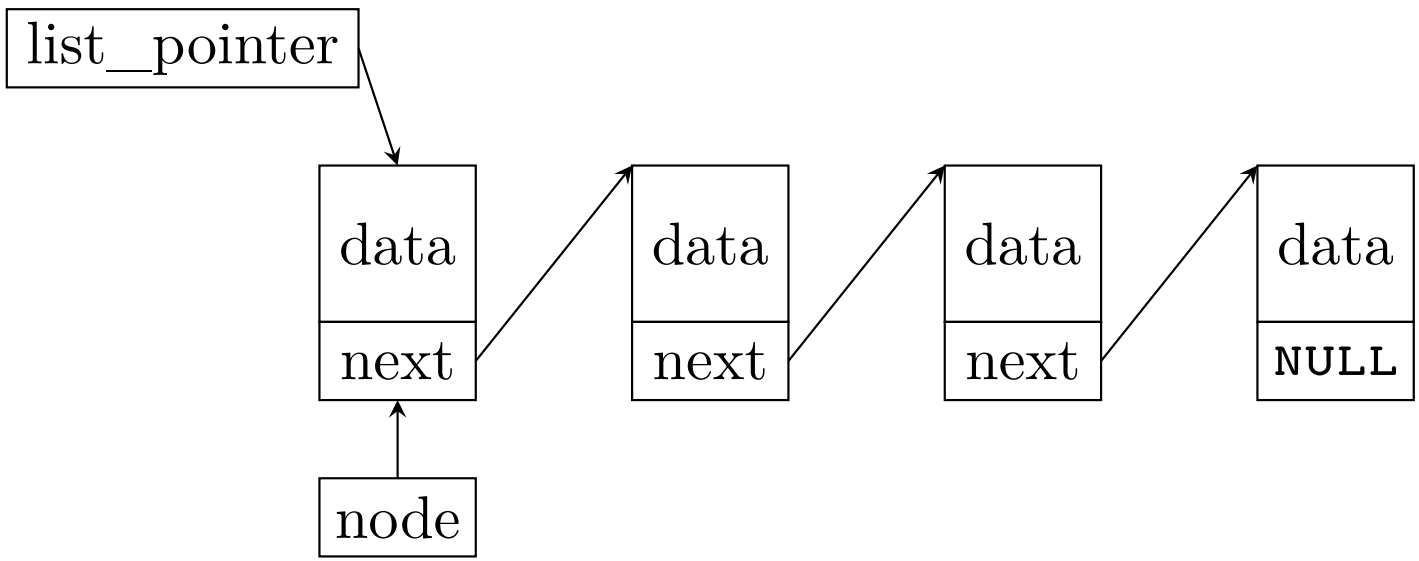
Iterowanie po liście



Iterowanie po jednokierunkowej liście liniowej

Jednokierunkowa lista liniowa

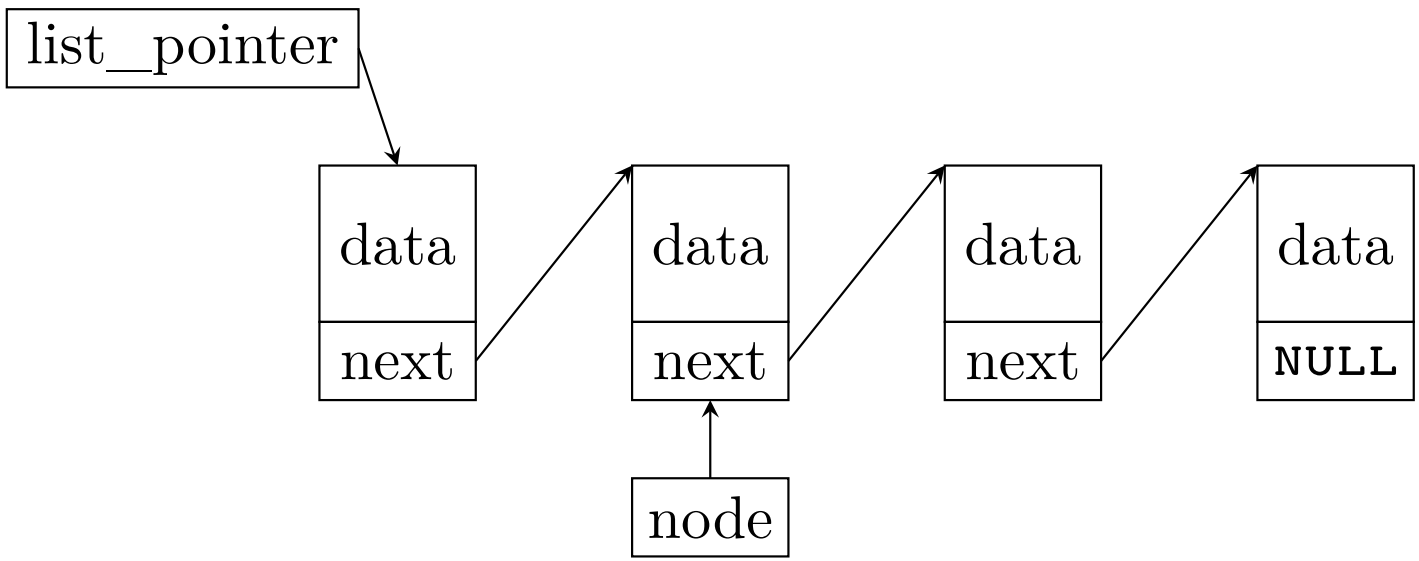
Iterowanie po liście



Iterowanie po jednokierunkowej liście liniowej

Jednokierunkowa lista liniowa

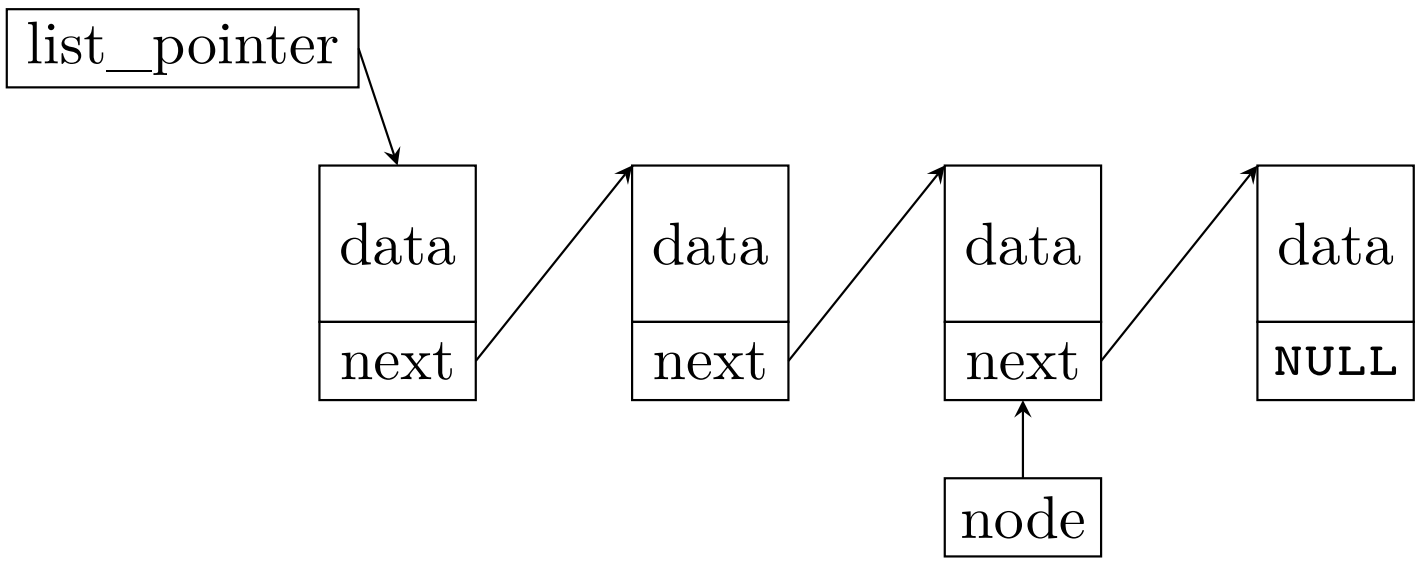
Iterowanie po liście



Iterowanie po jednokierunkowej liście liniowej

Jednokierunkowa lista liniowa

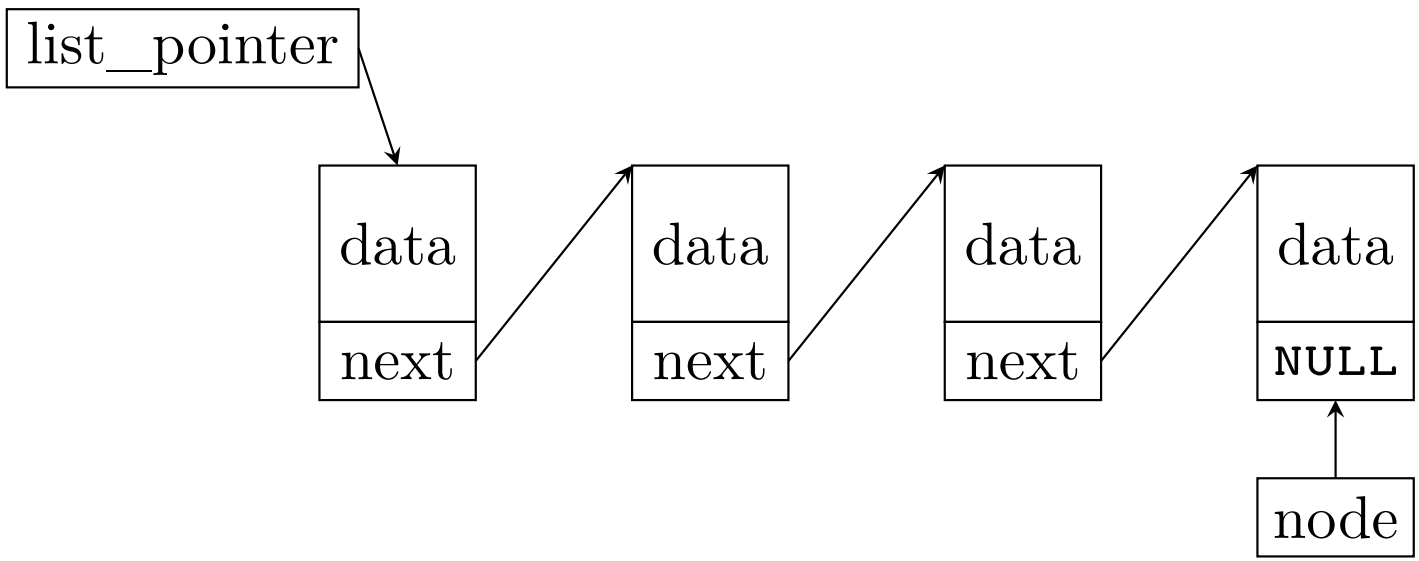
Iterowanie po liście



Iterowanie po jednokierunkowej liście liniowej

Jednokierunkowa lista liniowa

Iterowanie po liście



Iterowanie po jednokierunkowej liście liniowej

Jednokierunkowa lista liniowa

Funkcja `add_node()`

Po zakończeniu pętli `while` funkcja `add_node()` wywołuje funkcję `create_and_add_node()`, zwraca tę samą wartość, co ona i kończy swoje działanie.
Te krótkie opisy nie wyjaśniają jednak w pełni, jak obie funkcje przeprowadzają operację dodawania węzła do listy. Przejdźmy zatem do analizy poszczególnych przypadków, zaczynając od tego, w którym lista jest pusta.

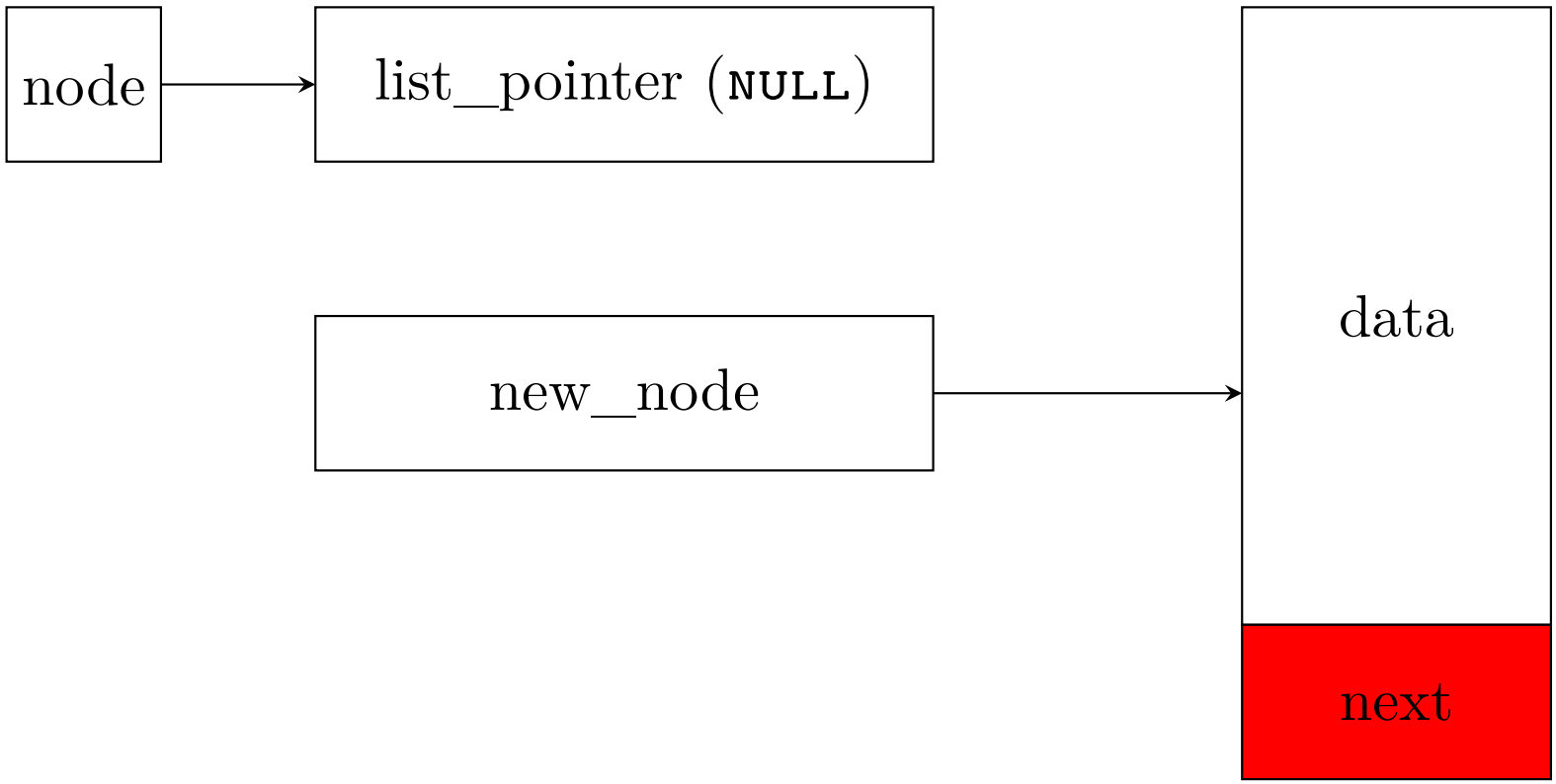
Jednokierunkowa lista liniowa

Dodanie pierwszego węzła

Ponieważ wskaźnik `list_pointer` przekazany do funkcji `add_node()` jest pusty w takim przypadku, to pętla `while` od razu się zakończy — wyrażenie `*node != NULL` jest fałszywe. Następuje wywołanie funkcji `create_and_add_node()`, która przydziela pamięć na nowy węzeł i jeśli ta operacja się powiedzie, to zapisuje w jego polu `data` liczbę przekazaną przez parametr `number` (wiersz nr 6, slajd nr 7). Następnie, w polu `next` zapisuje adres przechowywany we wskaźniku wskazywanym przez `node` (wiersz nr 7). Przypomnijmy, że jest to w tym przypadku wskaźnik listy, który jest pusty. Zatem w tym polu pojawi się wartość `NULL`. Jest ona prawidłowa, ponieważ dodawany węzeł będzie zarazem pierwszym i ostatnim w liście. Z kolei instrukcja przypisania w wierszu nr 8 zapisze do wskaźnika listy adres nowego węzła. Tym samym `list_pointer` zacznie wskazywać na ten węzeł i powstanie prawidłowa lista jednokierunkowa z jednym węzłem. Przebieg tego działania jest zilustrowany na następny slajdzie.

Jednokierunkowa lista liniowa

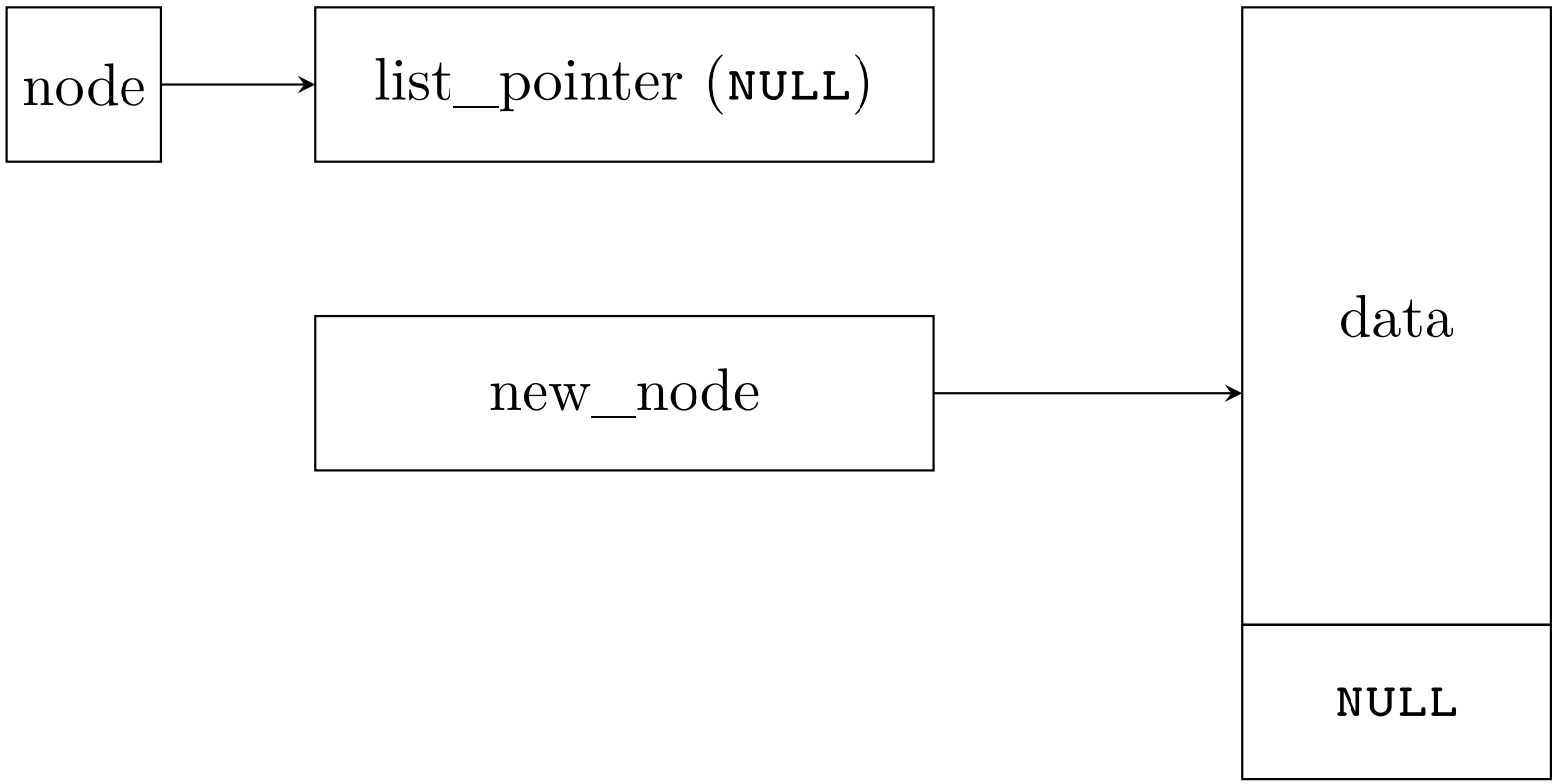
Dodanie pierwszego węzła



Przed wykonaniem wiersza nr 7 funkcji `create_and_add_node()`

Jednokierunkowa lista liniowa

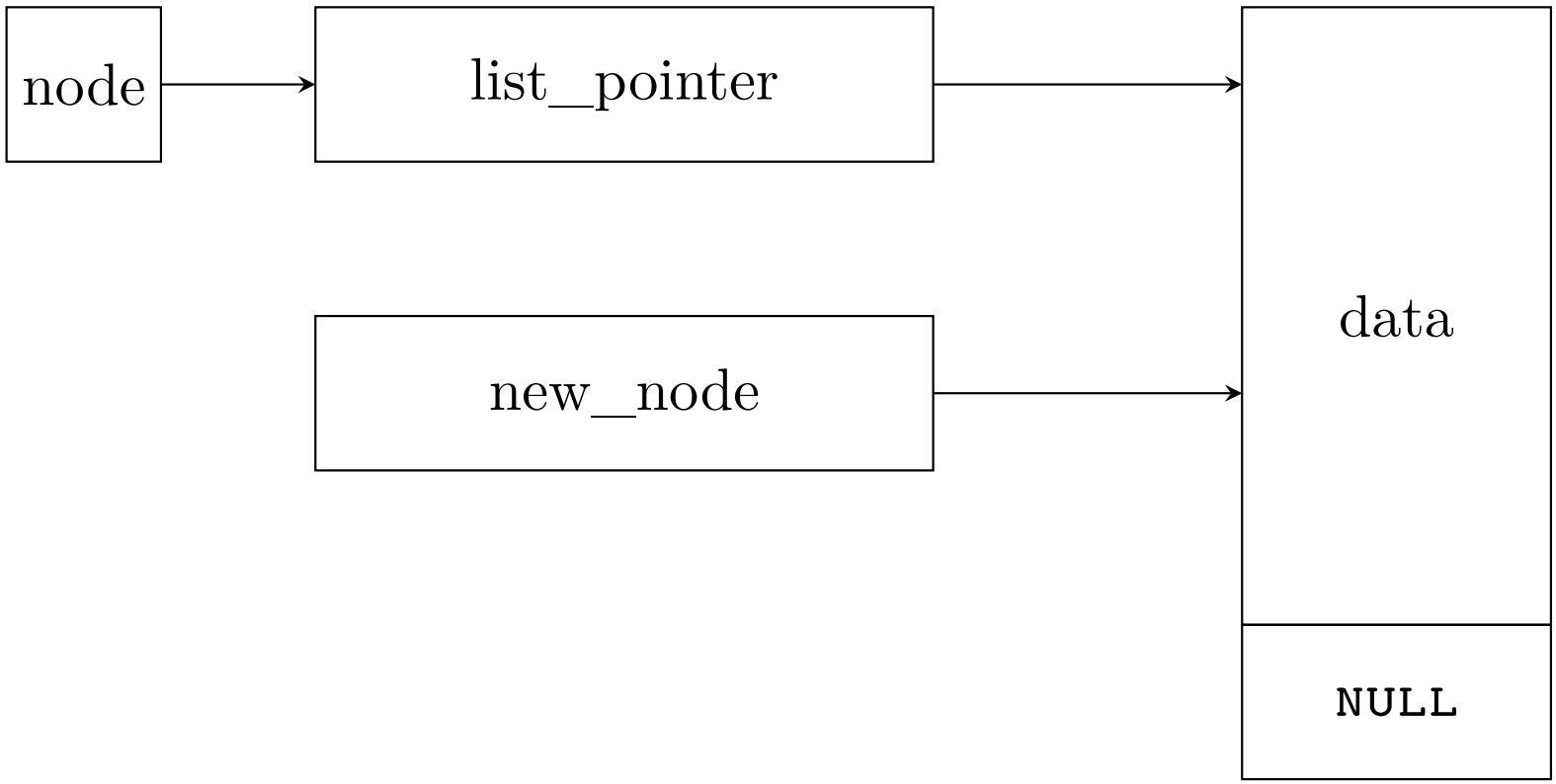
Dodanie pierwszego węzła



Przed wykonaniem wiersza nr 8 funkcji `create_and_add_node()`

Jednokierunkowa lista liniowa

Dodanie pierwszego węzła



Po wykonaniu wiersza nr 8 funkcji `create_and_add_node()`

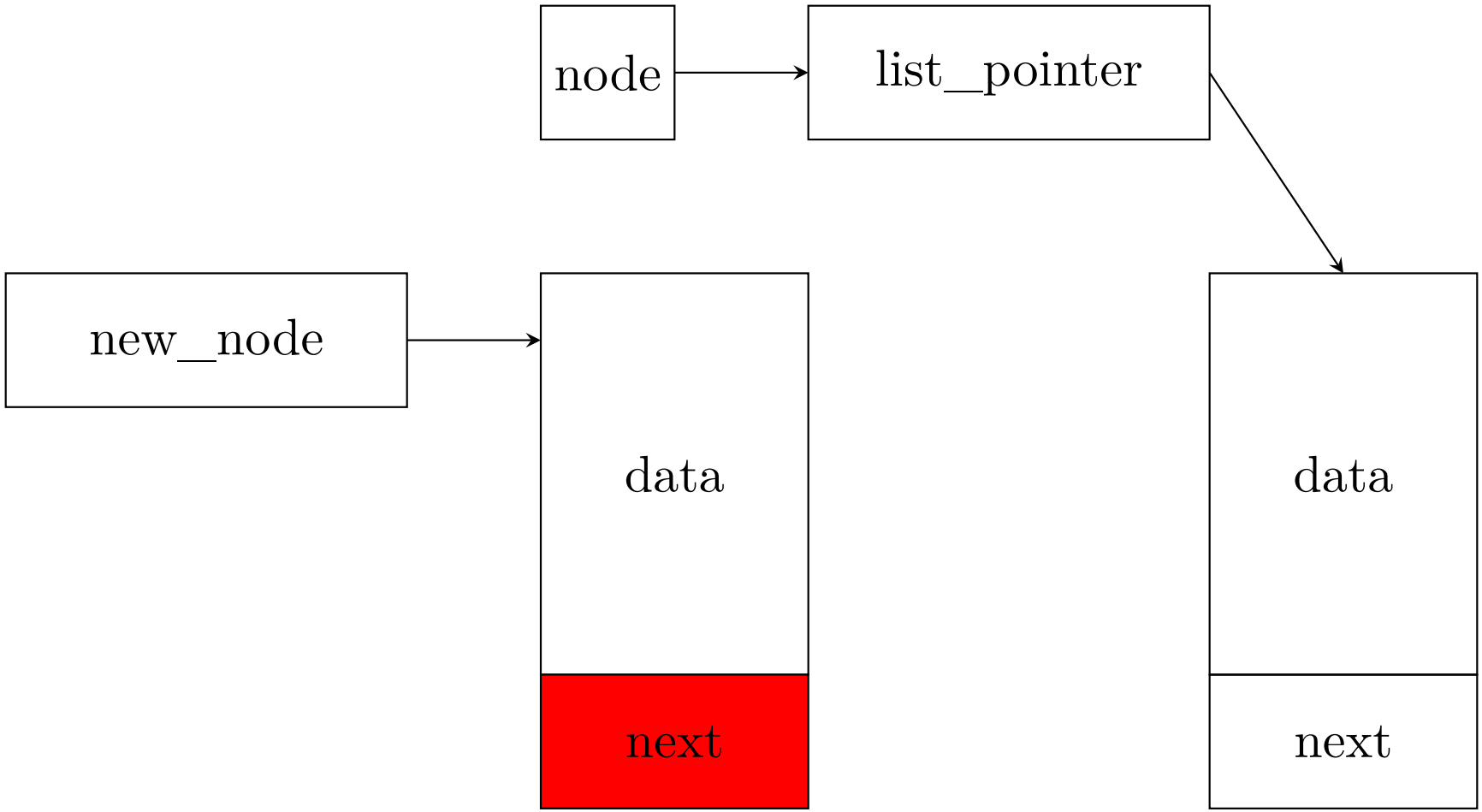
Jednokierunkowa lista liniowa

Dodanie na początku listy

W tym przypadku `list_pointer` zawiera adres pierwszego węzła listy, ale liczba, która w nim się znajduje ma wartość większą lub równą tej, która ma być zapisana w nowym węźle. Zatem i w tym przypadku pętla `while` w funkcji `add_node()` się zakończy od razu, bo fałszywe będzie wyrażenie `(*node)->data < number`, i wywołana zostanie `create_and_add_node()`. Ta funkcja utworzy nowy węzeł i jeśli ta operacja zakończy się sukcesem, to najpierw zapisze w jego polu `data` liczbę przekazaną przez parametr `number`. Następnie w polu `next` tego węzła umieszczony zostanie adres ze wskaźnika wskazywanego przez `node` (wiersz nr 7, slajd nr 7). Jak poprzednio, tym wskaźnikiem jest `list_pointer`, ale tym razem zawiera on adres pierwszego węzła listy, zatem pole `next` zacznie wskazywać właśnie na ten węzeł. W wierszu nr 8 funkcji `create_and_add_node()` we wskaźniku listy zostanie zapisany wskaźnik nowego węzła. Jest to konieczne, gdyż wskaźnik ten powinien zawsze wskazywać na początek listy, a teraz jest nim nowy węzeł.

Jednokierunkowa lista liniowa

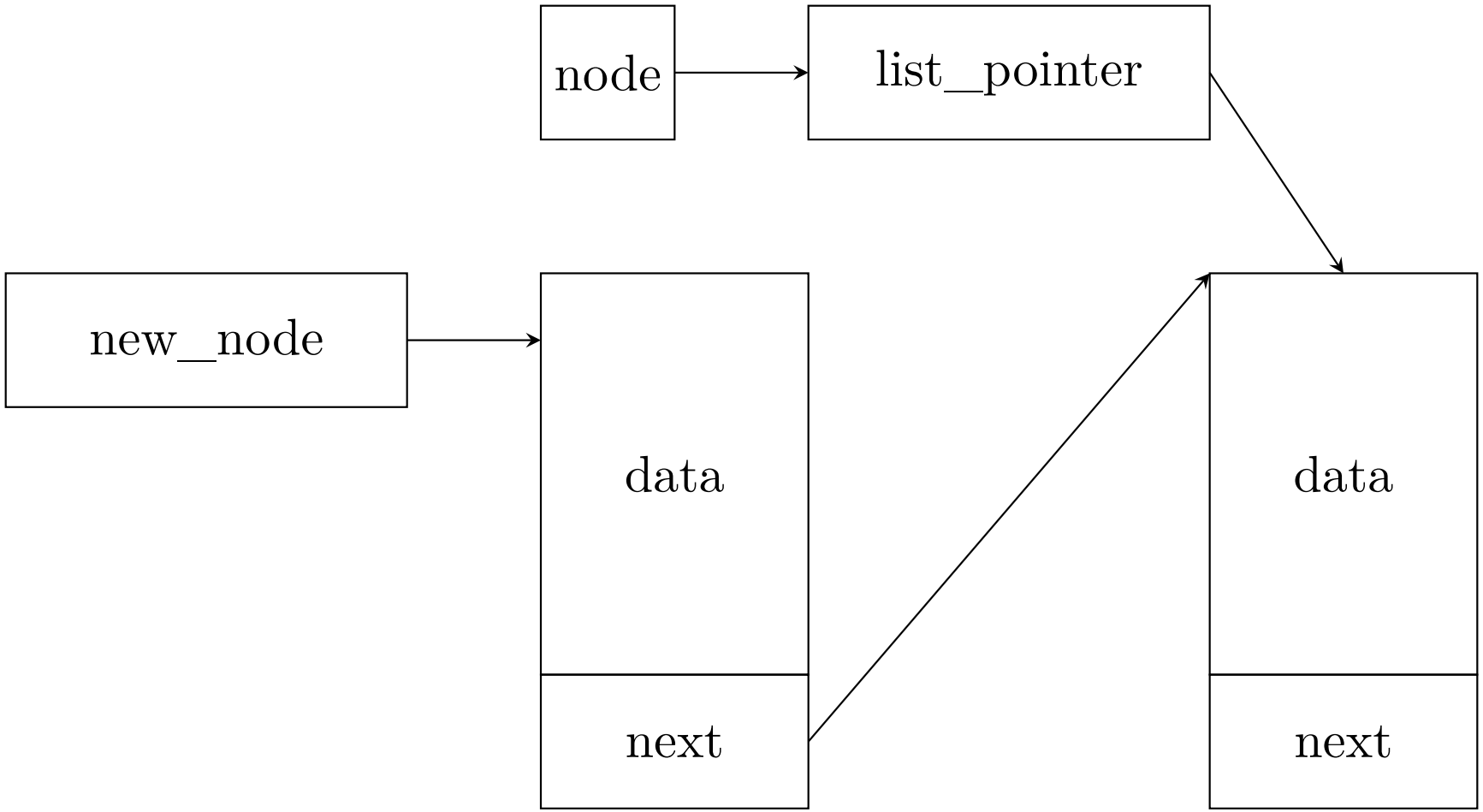
Dodanie na początku listy



Przed wykonaniem wiersza nr 7 funkcji `create_and_add_node()`

Jednokierunkowa lista liniowa

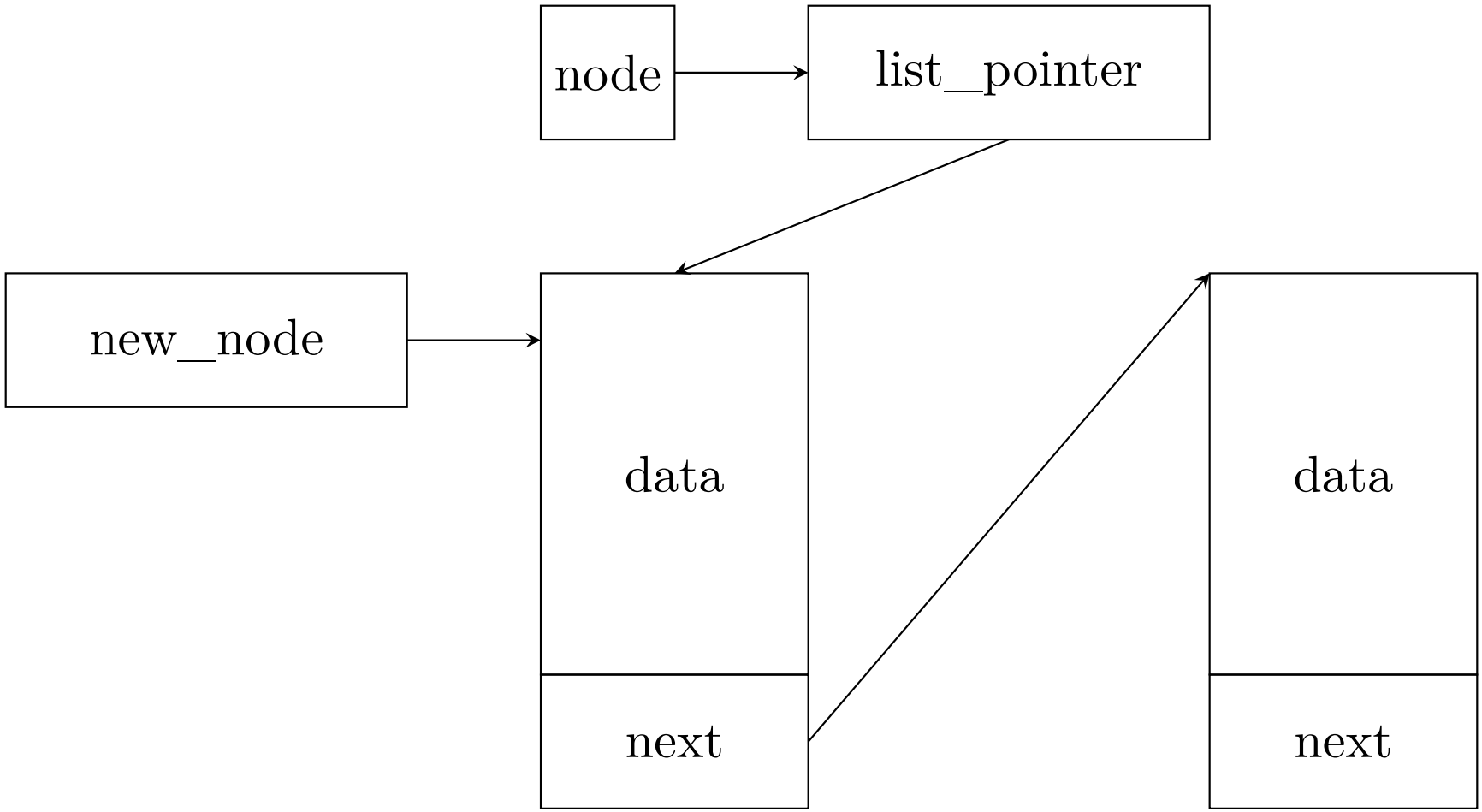
Dodanie na początku listy



Przed wykonaniem wiersza nr 8 funkcji `create_and_add_node()`

Jednokierunkowa lista liniowa

Dodanie na początku listy



Po wykonaniu wiersza nr 8 funkcji `create_and_add_node()`

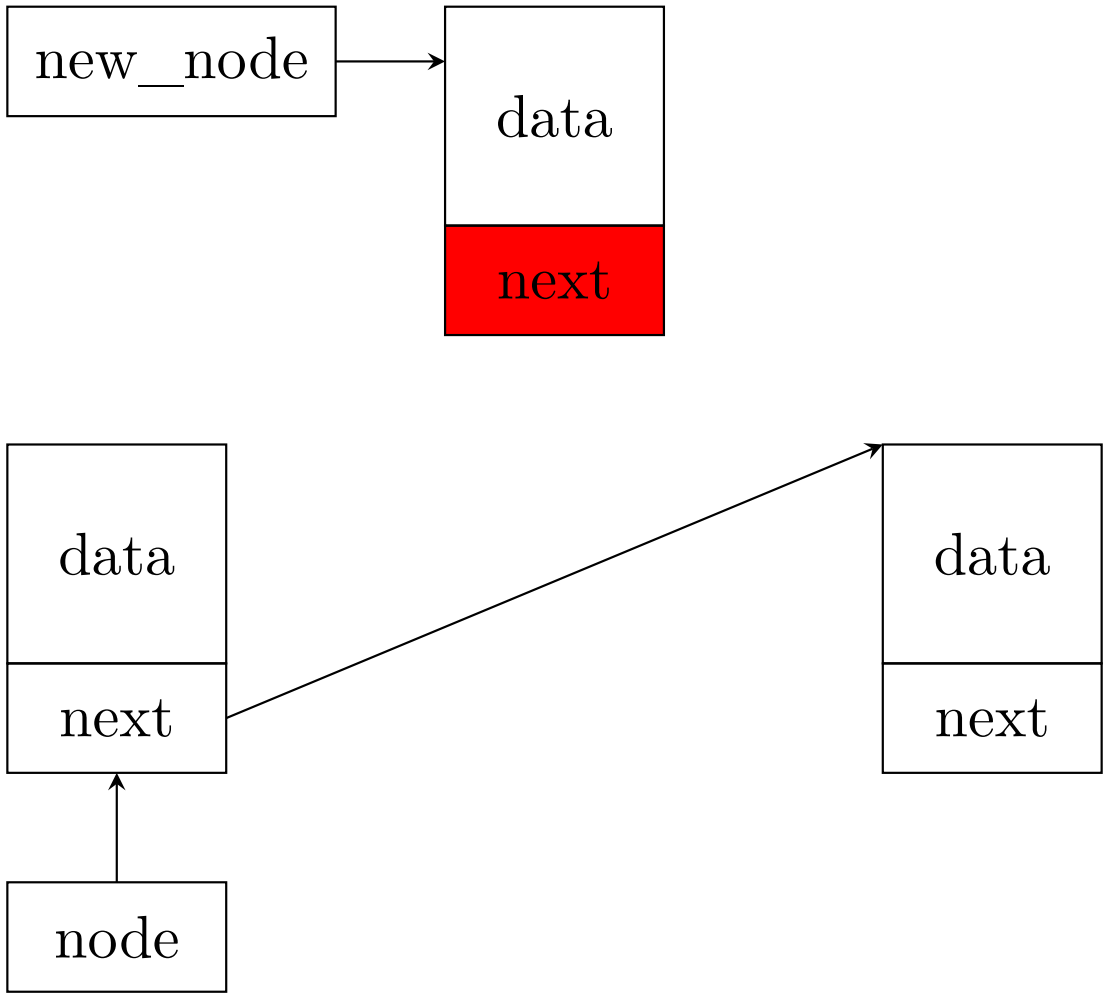
Jednokierunkowa lista liniowa

Dodanie wewnątrz

Kolejnym przypadkiem jest dodawanie nowego węzła wewnątrz listy, czyli *między* węzłami, które już do niej należą. Tym razem pętla `while` w funkcji `add_node()` wykona się pewną liczbę razy, aż we wskaźniku `node` znajdzie się adres pola `next` węzła, które wskazuje na inny (kolejny) węzeł, zawierający liczbę większą lub równą tej, która jest przekazana przez parametr `number`. Zatem, podobnie jak w poprzednim przypadku, wyrażenie `(*node)->data < number` będzie fałszywe i wywołana zostanie funkcja `create_and_add_node()`. Utworzy ona nowy węzeł i jeśli ta operacja zakończy się sukcesem, to zapisze w nim liczbę przekazaną przez `number` (wiersz nr 6), a następnie (wiersz nr 7) umieści w polu `next` tego węzła adres z pola `next` wskazywanego przez parametr `node`. Jest to adres węzła *przed* którym powinien być dodany nowy. Teraz pozostało zmienić wartość pola `next` wskazywanego przez `node`, tak aby wskazywało ono nowy węzeł. Dlatego w wierszu nr 8 zapisywany jest do tego pola adres tego węzła.

Jednokierunkowa lista liniowa

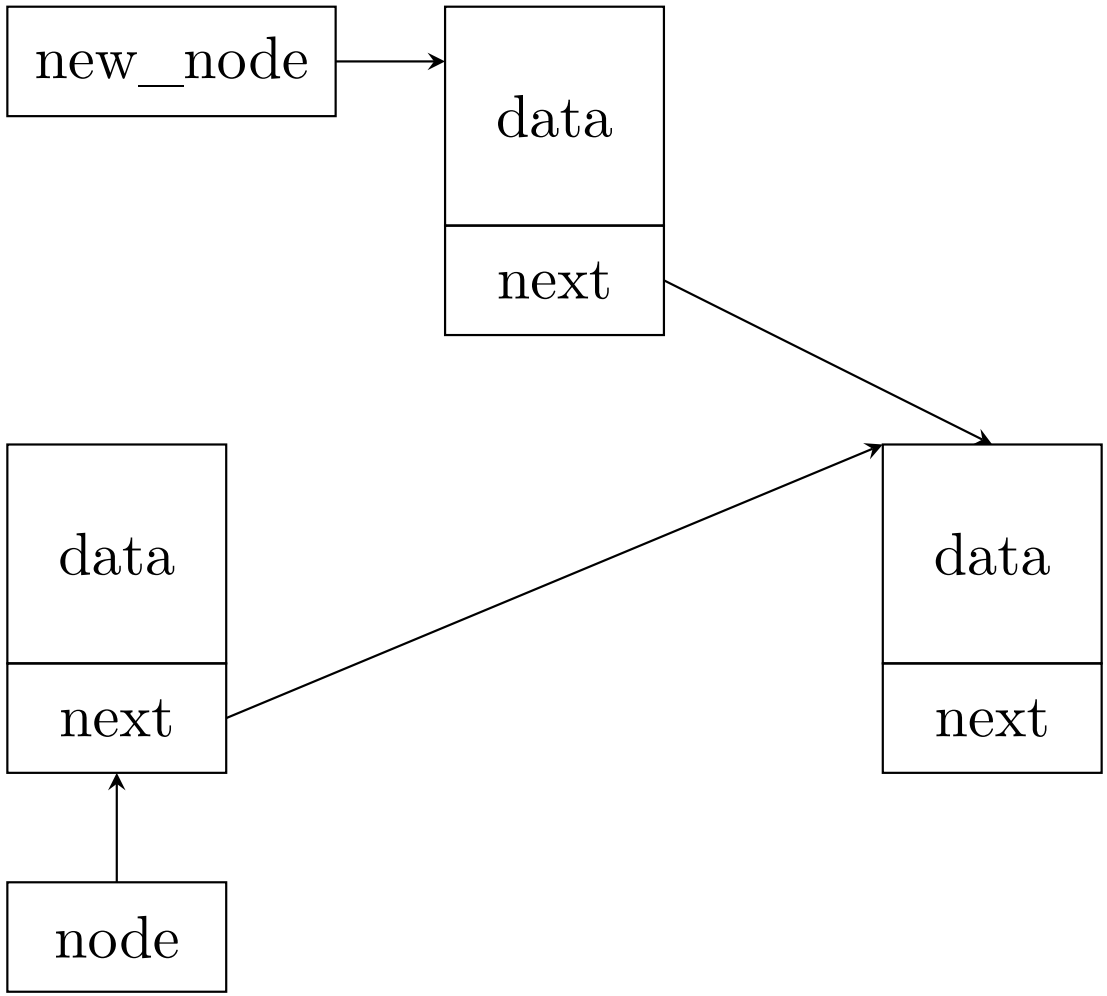
Dodanie wewnątrz



Przed wykonaniem wiersza nr 7 funkcji `create_and_add_node()`

Jednokierunkowa lista liniowa

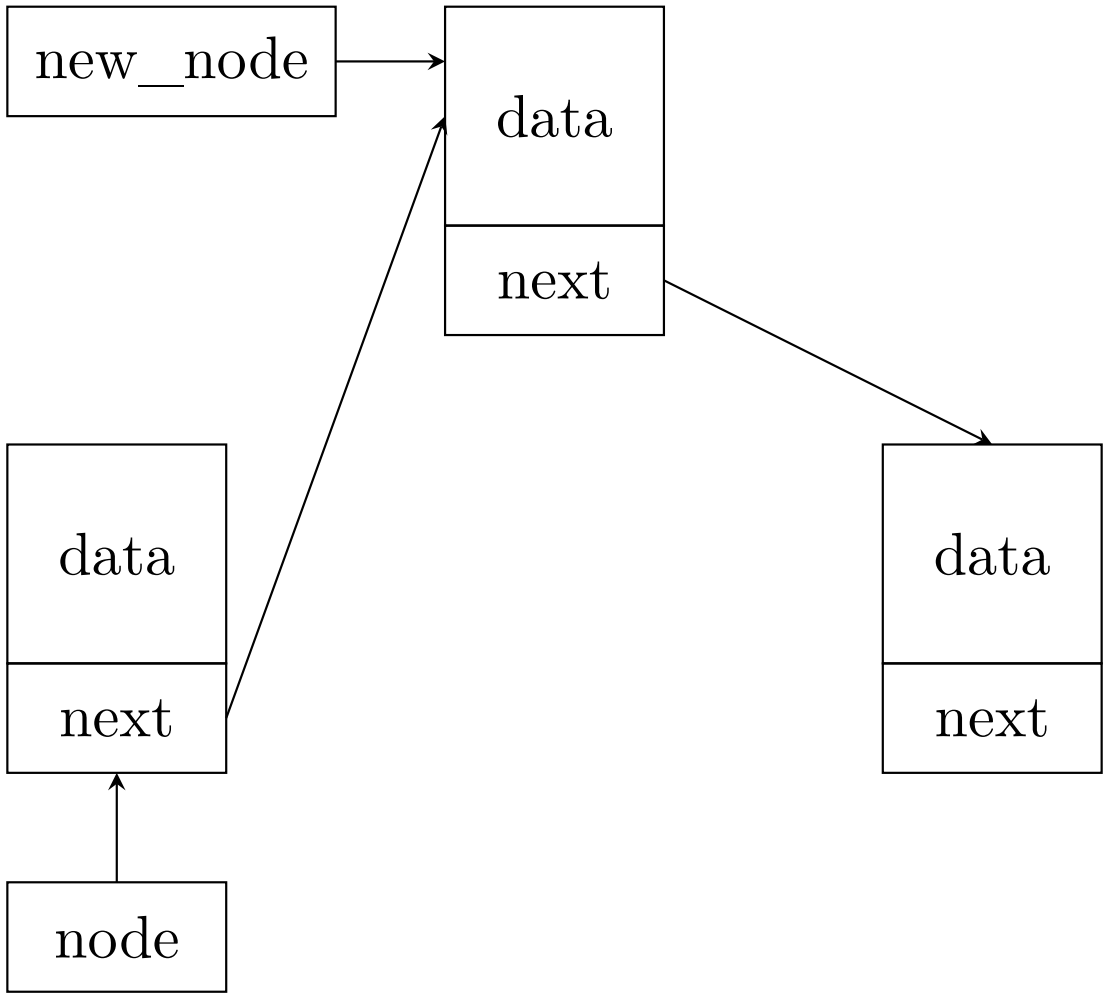
Dodanie wewnątrz



Przed wykonaniem wiersza nr 8 funkcji `create_and_add_node()`

Jednokierunkowa lista liniowa

Dodanie wewnątrz



Po wykonaniu wiersza nr 8 funkcji `create_and_add_node()`

Jednokierunkowa lista liniowa

Dodanie na końcu

Jeśli nowy węzeł listy będzie zawierał liczbę większą od wszystkich tych, które już są zgromadzone na liście, to będzie musiał on być umieszczony na jej końcu. W takim wypadku pęta `while` w funkcji `add_node()` zakończy się, gdy wyrażenie `*node != NULL` będzie fałszywe, czyli wskaźnik `node` będzie zawierał adres pola `next` węzła, które będzie zawierało wartość `NULL`. Będzie to zatem pole w ostatnim węźle na liście. Jego adres, wraz z liczbą do zapisania w nowym węźle zostanie przekazany do wywołanej funkcji `create_and_add_node()`. Ta, podobnie jak poprzednio, spróbuje utworzyć nowy węzeł i jeśli ta operacja się powiedzie, to zapisze w nim przekazaną liczbę (wiersz nr 6, slajd nr 7), a następnie umieści w jego polu `next` wartość ze wskaźnika wskazywanego przez `node`. Jak już ustaliliśmy, będzie to `NULL`, co jest odpowiednią wartością w przypadku węzła, który ma znaleźć się na końcu listy.

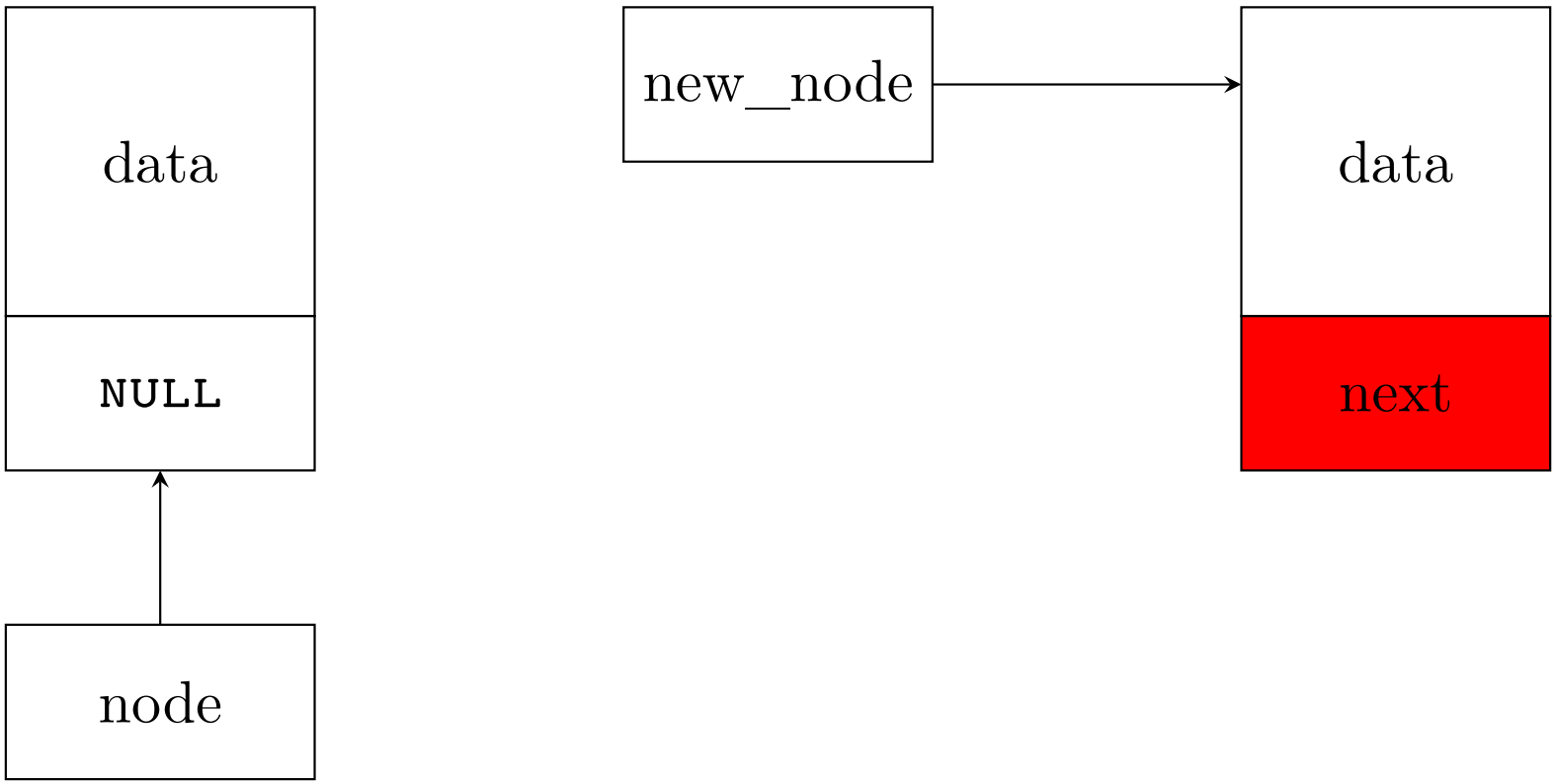
Jednokierunkowa lista liniowa

Dodanie na końcu

W wierszu nr 8 funkcji `create_and_add_node()` do pola `next` wskazywanego przez wskaźnik `node` zostanie zapisany adres nowego węzła. Tym samym dotychczas ostatni węzeł zacznie wskazywać swoim polem `next` na nowy i bieżąco ostatni węzeł listy.

Jednokierunkowa lista liniowa

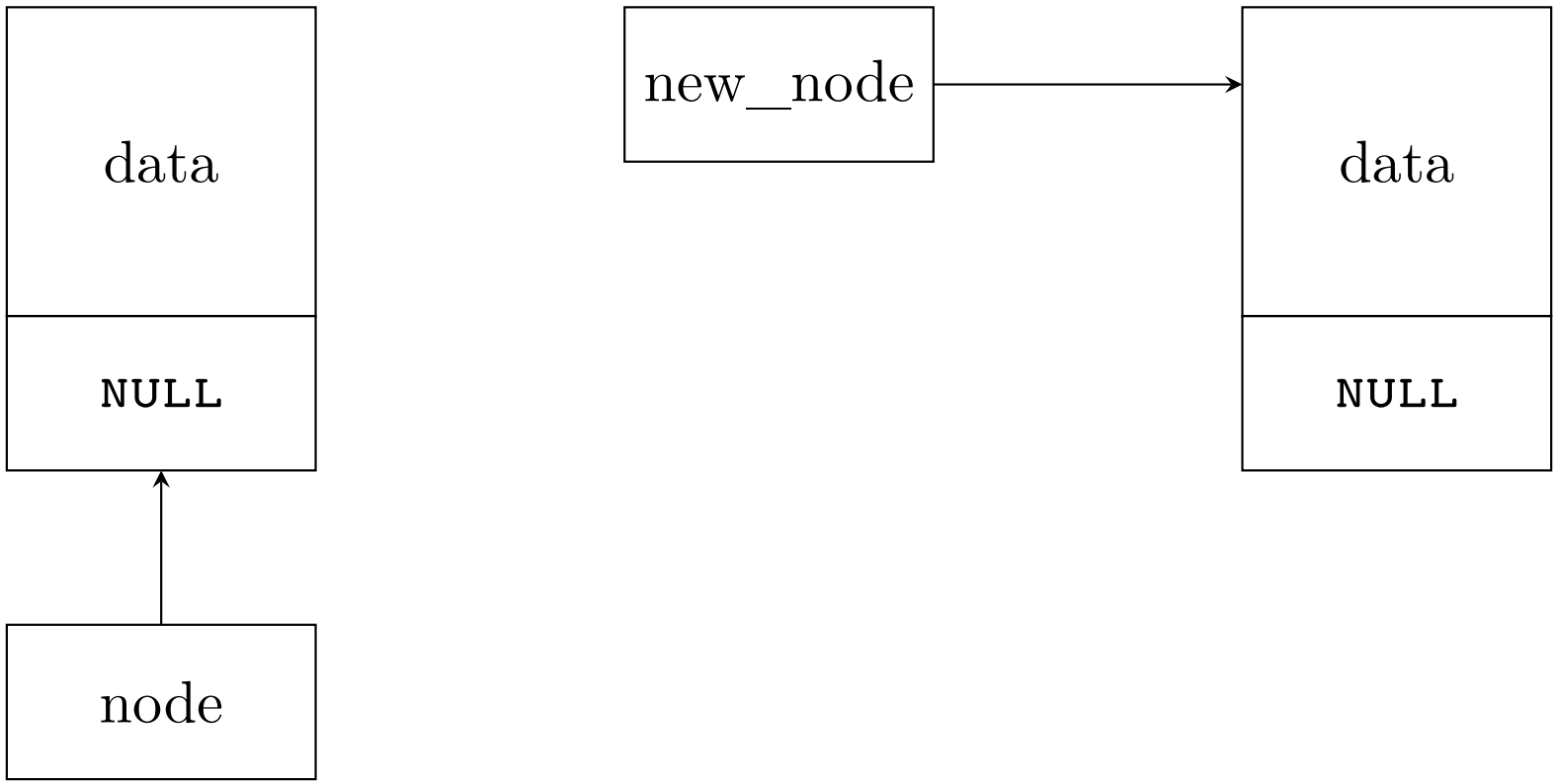
Dodanie na końcu



Przed wykonaniem wiersza nr 7 funkcji `create_and_add_node()`

Jednokierunkowa lista liniowa

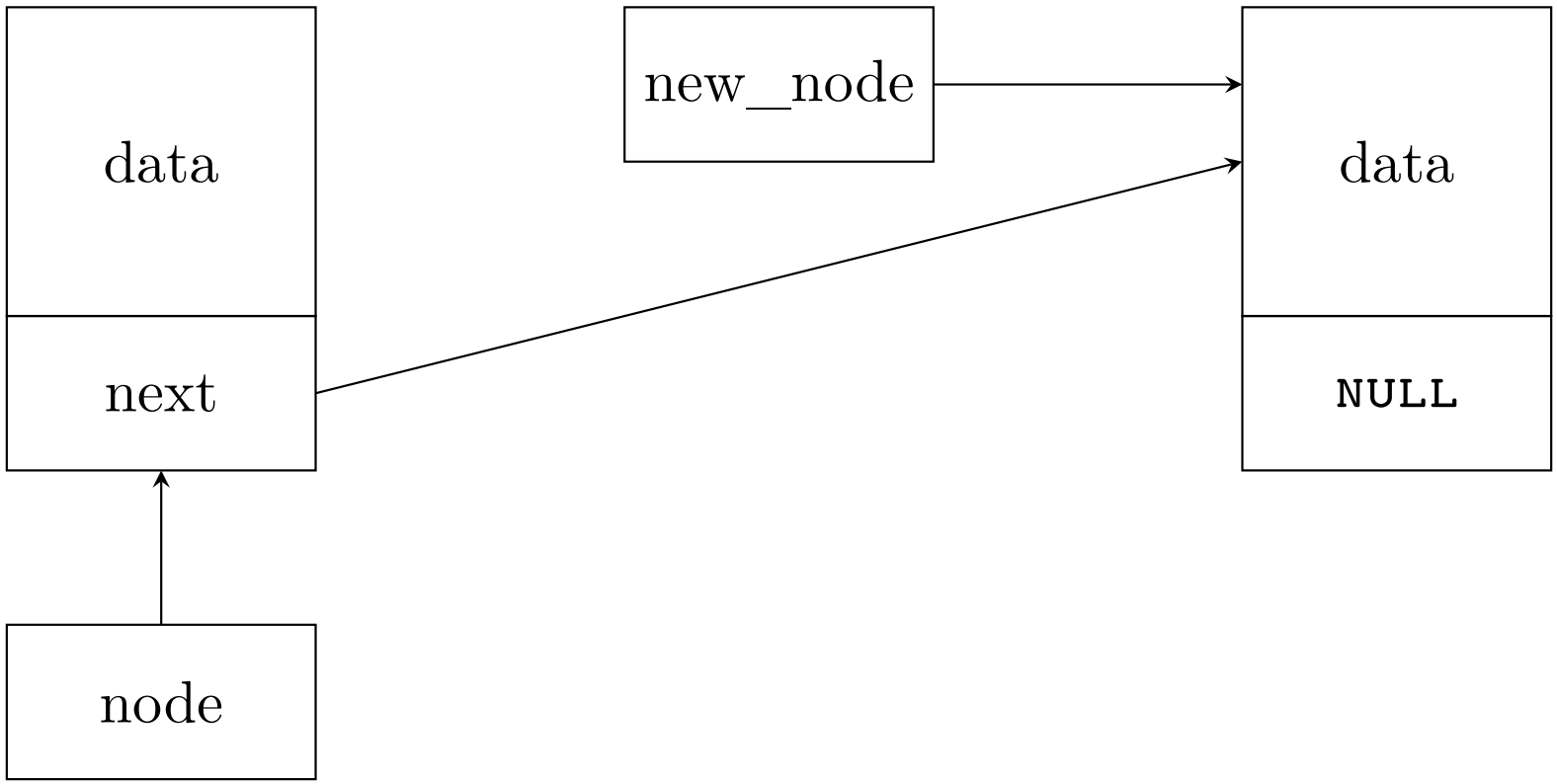
Dodanie na końcu



Przed wykonaniem wiersza nr 8 funkcji `create_and_add_node()`

Jednokierunkowa lista liniowa

Dodanie na końcu



Po wykonaniu wiersza nr 8 funkcji `create_and_add_node()`

Jednokierunkowa lista liniowa

Funkcja `delete_node()`

```
1  void delete_node(struct list_node **node, int number)
2  {
3      while(*node && (*node)->data != number)
4          node = &(*node)->next;
5      if(*node) {
6              struct list_node *temporary =
   ↪   (*node)->next;
7              free(*node);
8              *node = temporary;
9          }
10 }
```

Jednokierunkowa lista liniowa

Funkcja `delete_node()`

Funkcja `delete_node()` odpowiada za usunięcie pojedynczego węzła z listy. Kryterium branym przez nią pod uwagę przy wyszukiwaniu tego węzła jest liczba zapisana w jego polu `number`. Jeśli funkcja nie znajdzie takiego węzła, to zakończy swoje działanie bez usuwania żadnego z węzłów. Jeśli w liście jest więcej niż jeden węzeł przechowujący tę samą liczbę, to usunie z niej pierwszego z nich.
Zapis tej funkcji jest bardziej zwarty, niż w przypadku dwóch opisanych wcześniej. Podobnie jak `add_node()`, funkcja `delete_node()` przyjmuje jako pierwszy argument adres wskaźnika listy. Jako drugi jest przekazywana liczba, którą powinien zawierać węzeł do usunięcia. Funkcja nie zwraca żadnej wartości.

Jednokierunkowa lista liniowa

Funkcja `delete_node()`

Na początku wykonuje ona pętlę **`while`**, która jest bardzo podobna do tej z **`add_node()`** (wiersze 3–4). Jedyna różnica polega na operatorze użytym w drugim wyrażeniu, w jej warunku. Pętla zakończy się, jeżeli wskaźnik wskazywany przez **`node`** będzie pusty, lub gdy będzie on wskazywał na węzeł zawierający liczbę przekazaną przez parametr **`number`**. W pierwszym przypadku będzie to oznaczało, że nie istnieje węzeł w liście, który należałoby usunąć, a w drugim, że taka operacja powinna być przeprowadzona. Aby rozróżnić te dwie sytuacje funkcja sprawdza, czy wskaźnik wskazywany przez **`node`** nie jest pusty (wiersz nr 5). Jeśli ten warunek jest spełniony to zapamiętuje ona w zmiennej **`temporary`** adres przechowywany w polu **`next`** węzła wskazywanego przez wskaźnik, którego adres znajduje się w **`node`** (wiersz nr 6), usuwa ten węzeł (wiersz nr 7) i zapisuje we wskaźniku wskazywanym przez **`node`** adres przechowywany w **`temporary`** (wiersz nr 8).

Jednokierunkowa lista liniowa

Funkcja `delete_node()`

Krótki opis, który znajduje się na poprzednim slajdzie, nie przedstawia szczegółów działania funkcji `delete_node()`. Przeanalizujemy zatem jej zachowanie dla trzech najbardziej interesujących przypadków:

1. usunięcie pierwszego węzła z listy,
2. usunięcie węzła z wnętrza listy,
3. usunięcie ostatniego węzła listy.

Jednokierunkowa lista liniowa

Usunięcie pierwszego węzła

W pierwszym przypadku pętla `while` w funkcji `delete_node()` zakończy się od razu, gdyż wyrażenie `(*node)->data != number` będzie fałszywe. Oznacza to, że poszukiwana w pętli liczba znajduje się w pierwszym węźle i właśnie ten węzeł należy usunąć. Zatem warunek w wierszu nr 5 będzie spełniony. Funkcja `delete_node()` zapamięta w zmiennej `temporary` adres znajdujący się w polu `next` węzła wskazywanego przez wskaźnik, którego adres jest zapisany w `node` (wiersz nr 6). Tym wskaźnikiem jest w tym przypadku wskaźnik listy (`list_pointer`), a tym węzłem pierwszy węzeł listy. Zatem w zmiennej `temporary` znajdzie się *adres drugiego węzła* listy (o ile on istnieje). Następnie `delete_node()` zwolni pamięć przeznaczoną na pierwszy węzeł (wiersz nr 7) i zapisze we wskaźniku wskazywanym przez `node` adres znajdujący się w zmiennej `temporary` (wiersz nr 8).

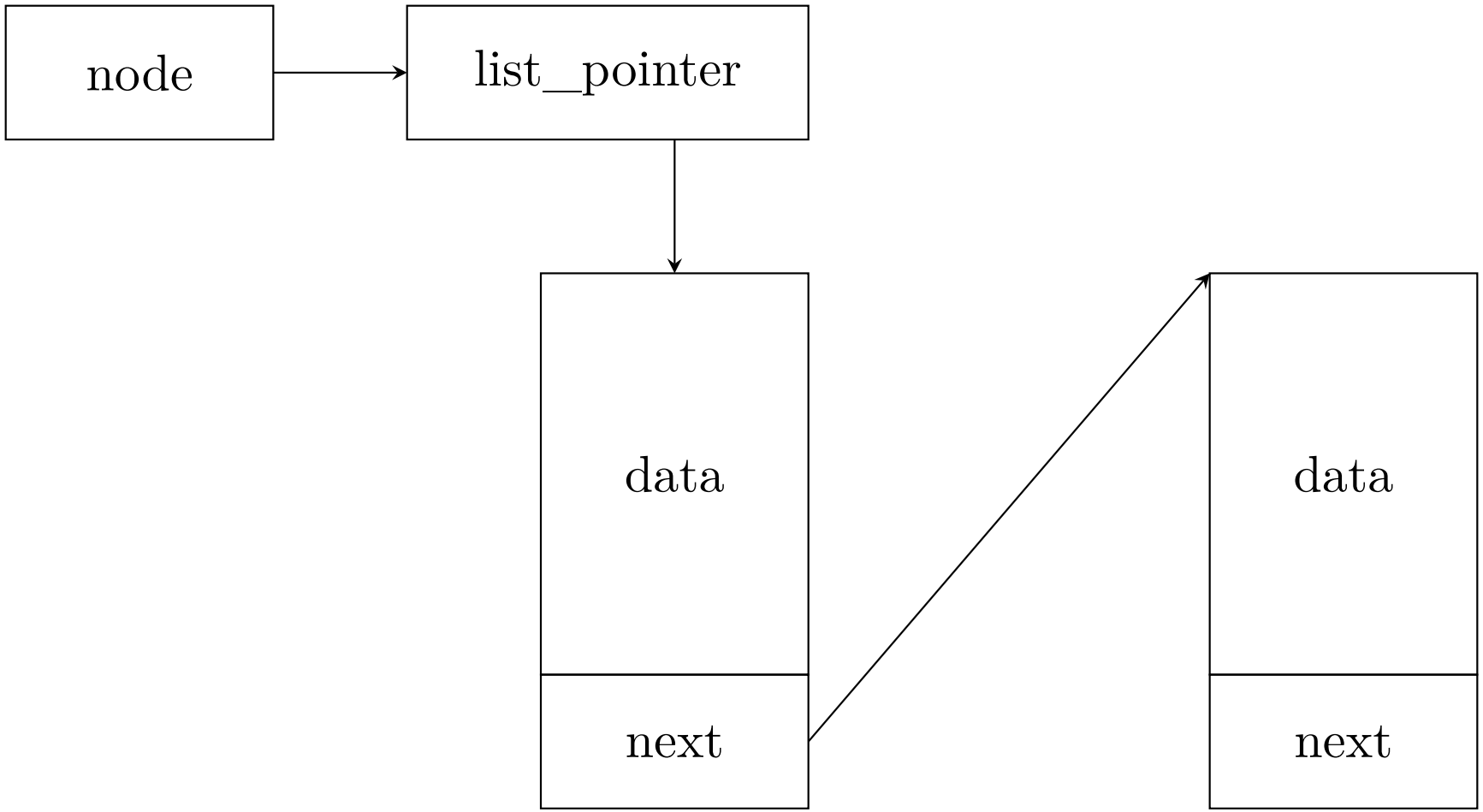
Jednokierunkowa lista liniowa

Usunięcie pierwszego węzła

Proszę zwrócić uwagę, że funkcja zdziała prawidłowo również wtedy, gdy pierwszy węzeł będzie jednocześnie jedynym węzłem w liście. W takim przypadku, w wierszu nr 6 do `temporary` zostanie zapisana wartość `NULL`, która następnie trafi do wskaźnika listy (wiersz nr 8). Jest to działanie oczekiwane, bo po usunięciu jedynego węzła lista staje się pusta i `list_pointer` też powinien stać się wskaźnikiem pustym.
Kolejny slajd ilustruje usunięcie przez `delete_node()` pierwszego węzła z listy, która zawiera ich co najmniej dwa.

Jednokierunkowa lista liniowa

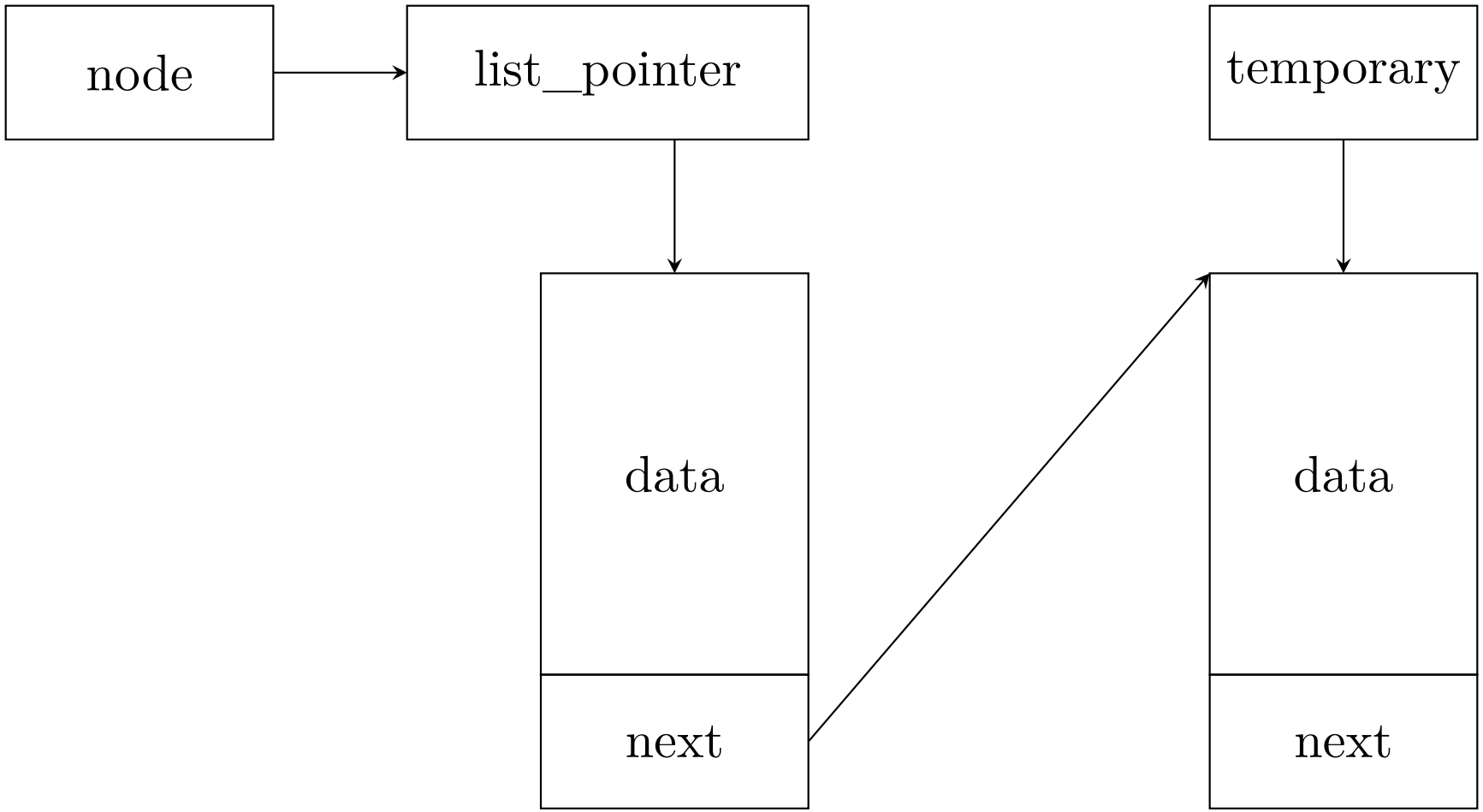
Usunięcie pierwszego węzła



Przed wykonaniem wiersza nr 6 funkcji `delete_node()`

Jednokierunkowa lista liniowa

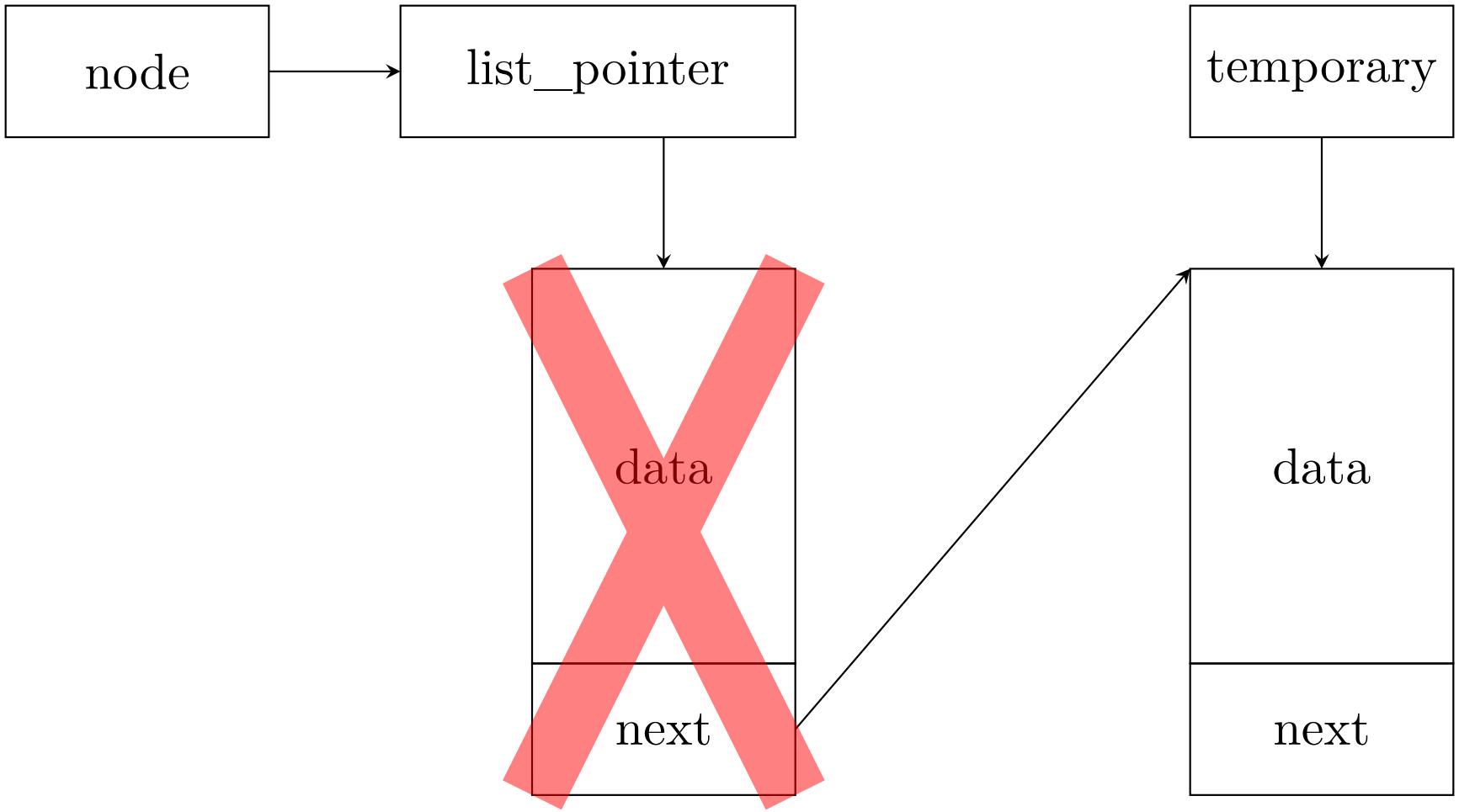
Usunięcie pierwszego węzła



Przed wykonaniem wiersza nr 7 funkcji `delete_node()`

Jednokierunkowa lista liniowa

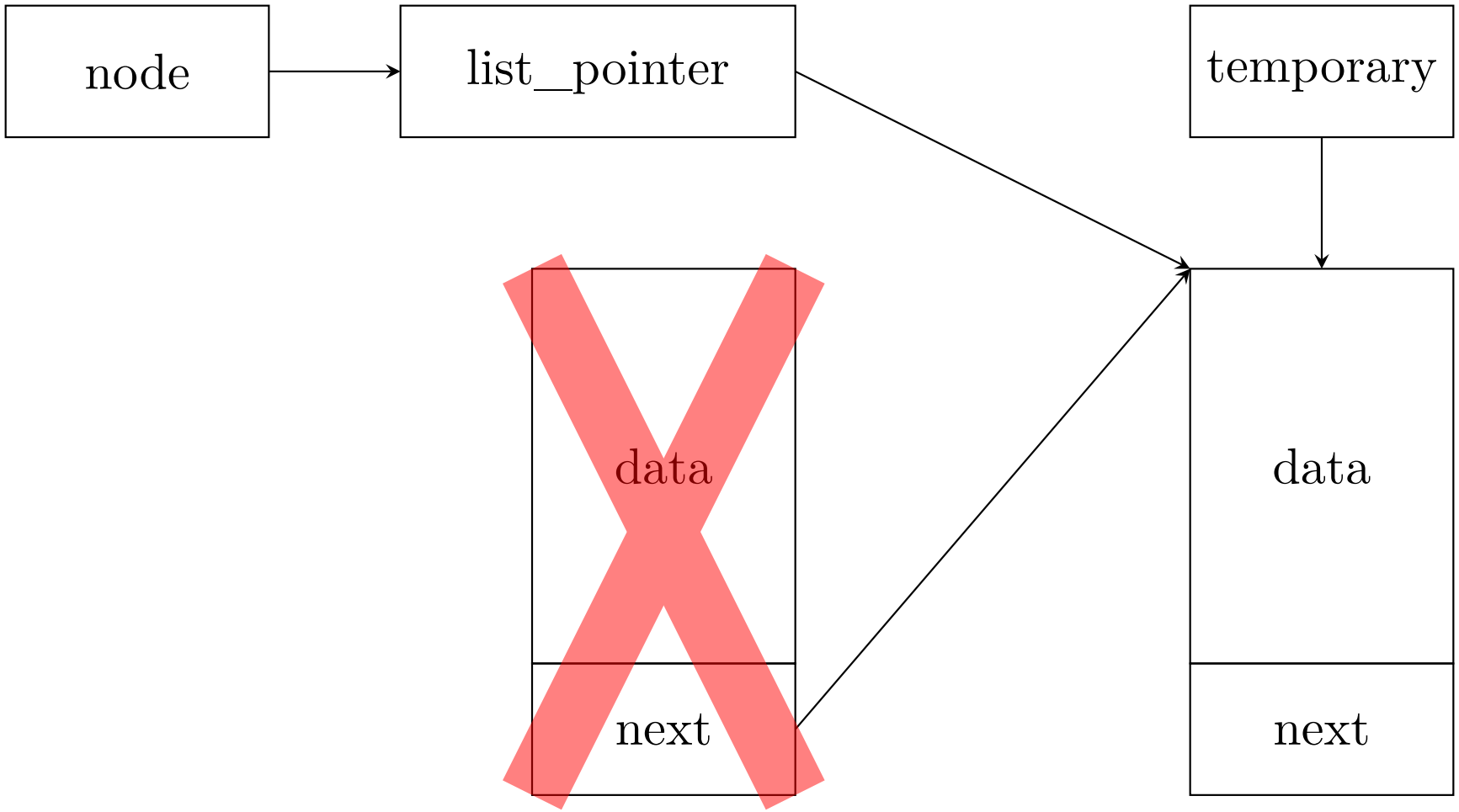
Usunięcie pierwszego węzła



Przed wykonaniem wiersza nr 8 funkcji `delete_node()`

Jednokierunkowa lista liniowa

Usunięcie pierwszego węzła



Po wykonaniu wiersza nr 8 funkcji `delete_node()`

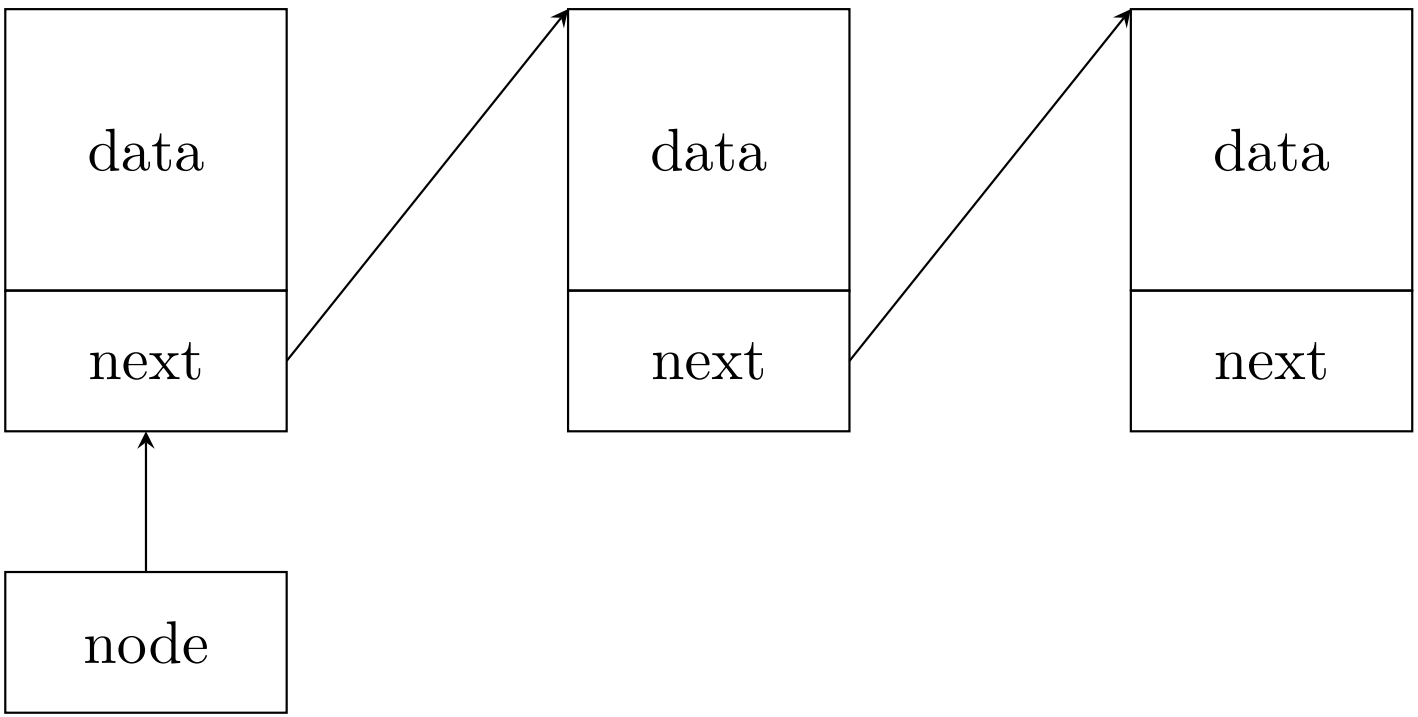
Jednokierunkowa lista liniowa

Usunięcie wewnętrznego węzła

W przypadku, gdy węzeł do usunięcia znajduje się wewnątrz listy, to pętla `while` w funkcji `delete_node()` wykona się tyle razy, aż w `node` znajdzie się adres pola `next` wskazującego na węzeł zawierający poszukiwaną liczbę, czyli znów wyrażenie `(*node)->data != number` będzie fałszywe. Za to warunek w wierszu nr 5 będzie spełniony. Funkcja zapisze w zmiennej `temporary` adres kolejnego węzła w liście, po tym do usunięcia. Następnie zwolni ona pamięć przeznaczoną na usuwany węzeł (wiersz nr 7) i zapisze w polu `next` wskazywanym przez `node` adres zapamiętany w `temporary`. Dzięki temu węzeł, który dotychczas poprzedzał ten usunięty zacznie wskazywać na ten, który dotychczas był za usuniętym. Dzięki temu lista zachowa ciągłość.

Jednokierunkowa lista liniowa

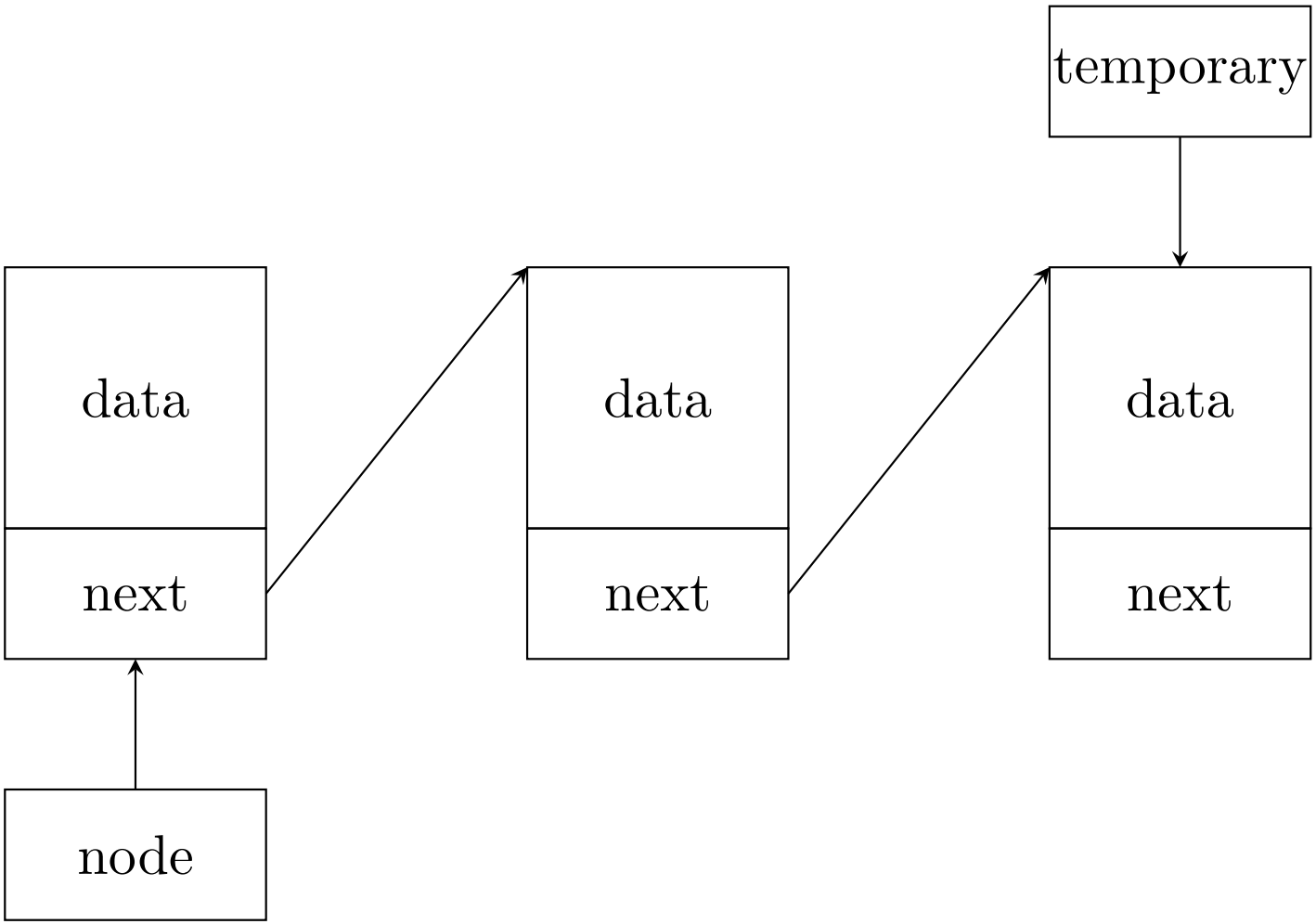
Usunięcie wewnętrznego węzła



Przed wykonaniem wiersza nr 6 funkcji `delete_node()`

Jednokierunkowa lista liniowa

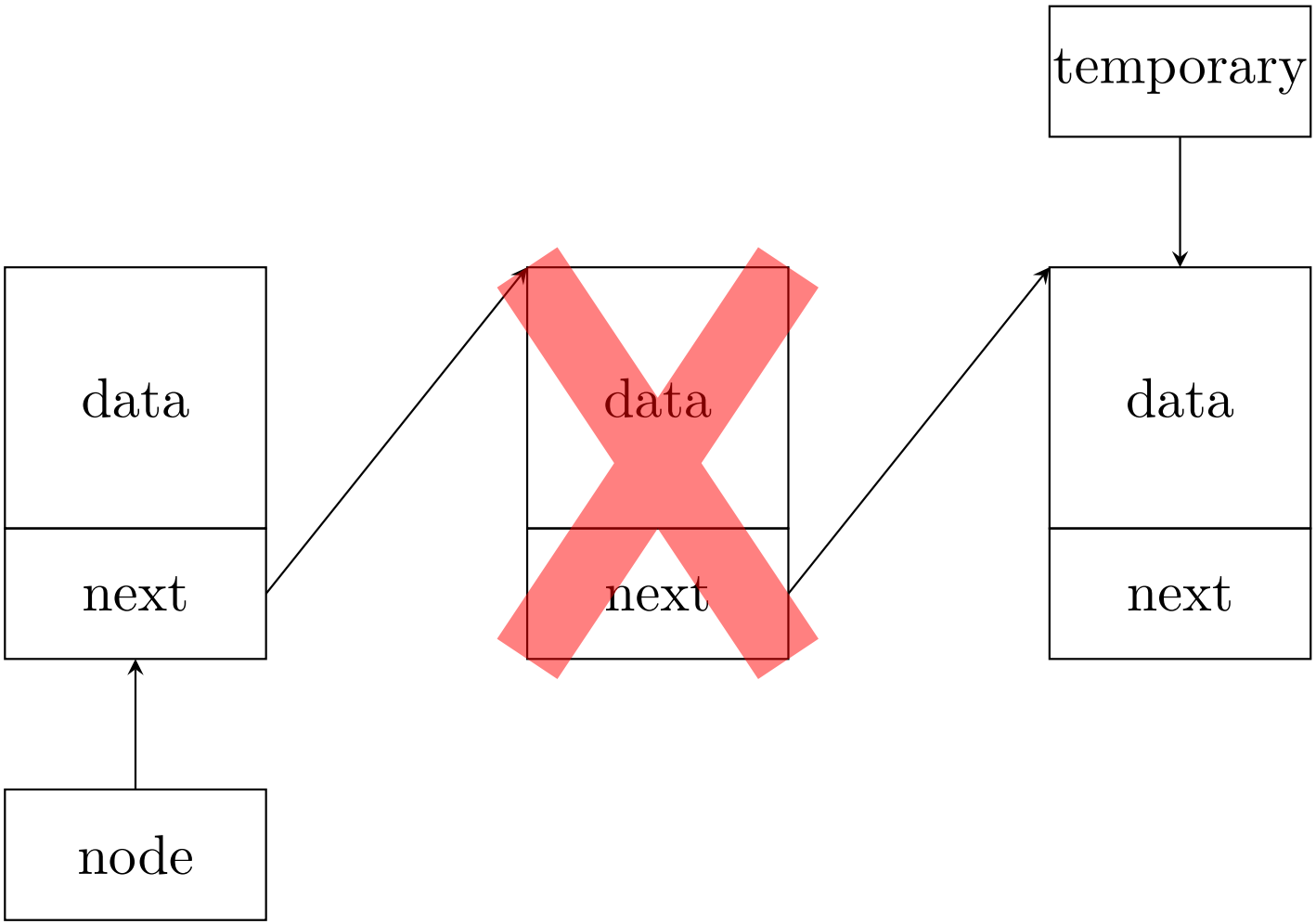
Usunięcie wewnętrznego węzła



Przed wykonaniem wiersza nr 7 funkcji `delete_node()`

Jednokierunkowa lista liniowa

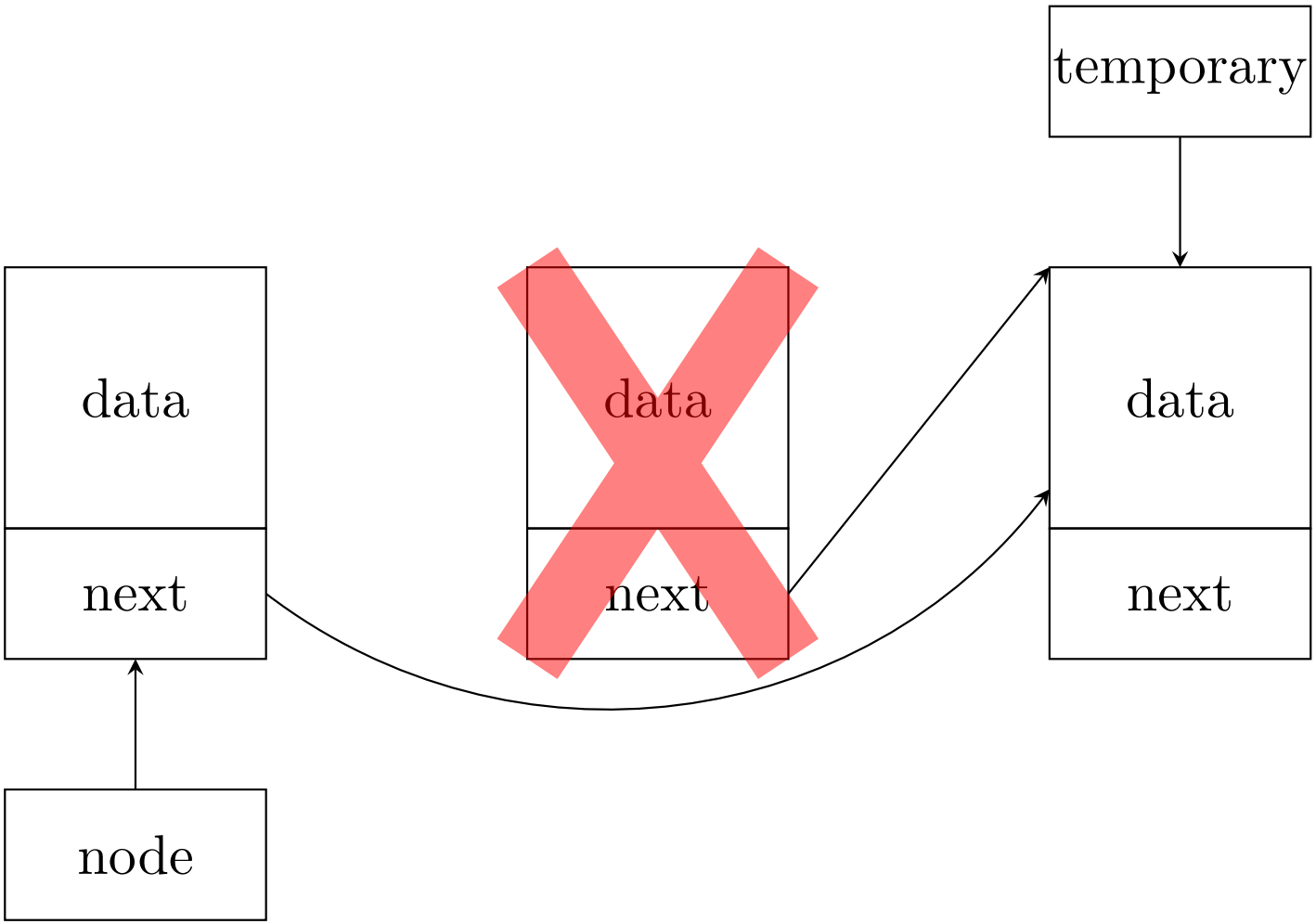
Usunięcie wewnętrznego węzła



Przed wykonaniem wiersza nr 8 funkcji `delete_node()`

Jednokierunkowa lista liniowa

Usunięcie wewnętrznego węzła



Po wykonaniu wiersza nr 8 funkcji `delete_node()`

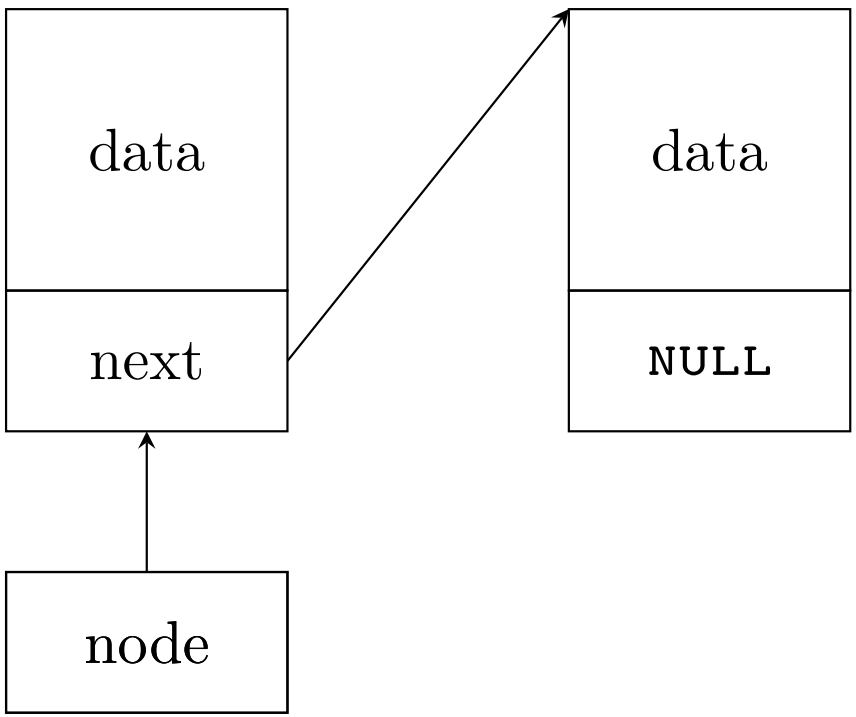
Jednokierunkowa lista liniowa

Usunięcie ostatniego węzła

W trzecim przypadku pętla `while` w funkcji `delete_node()` również zakończy się, gdy parametr `node` będzie wskazywał na pole `next`, wskazujące na węzeł, w którym zapisana jest poszukiwana liczba. Zatem ponownie warunek `(*node)->data != number` nie będzie spełniony, ale tym razem szukana liczba będzie znajdowała się w ostatnim węźle listy. Po sprawdzeniu warunku w wierszu nr 5 funkcja zapisze w zmiennej `temporary` adres znajdujący się w polu `next` węzła wskazywanego przez wskaźnik, którego adres jest w `node` (wiersz nr 6). Ten wskaźnik, to pole `next` *przedostatniego węzła w liście*, a adres zapisany w zmiennej `temporary`, to po prostu wartość `NULL`. Po wykonaniu tej operacji funkcja `delete_node()` zwolni pamięć przeznaczoną na ostatni węzeł (wiersz nr 7) i zapisze w polu `next` dotychczas przedostatniego węzła (wiersz nr 8) wartość `NULL` — teraz jest to już ostatni węzeł listy.

Jednokierunkowa lista liniowa

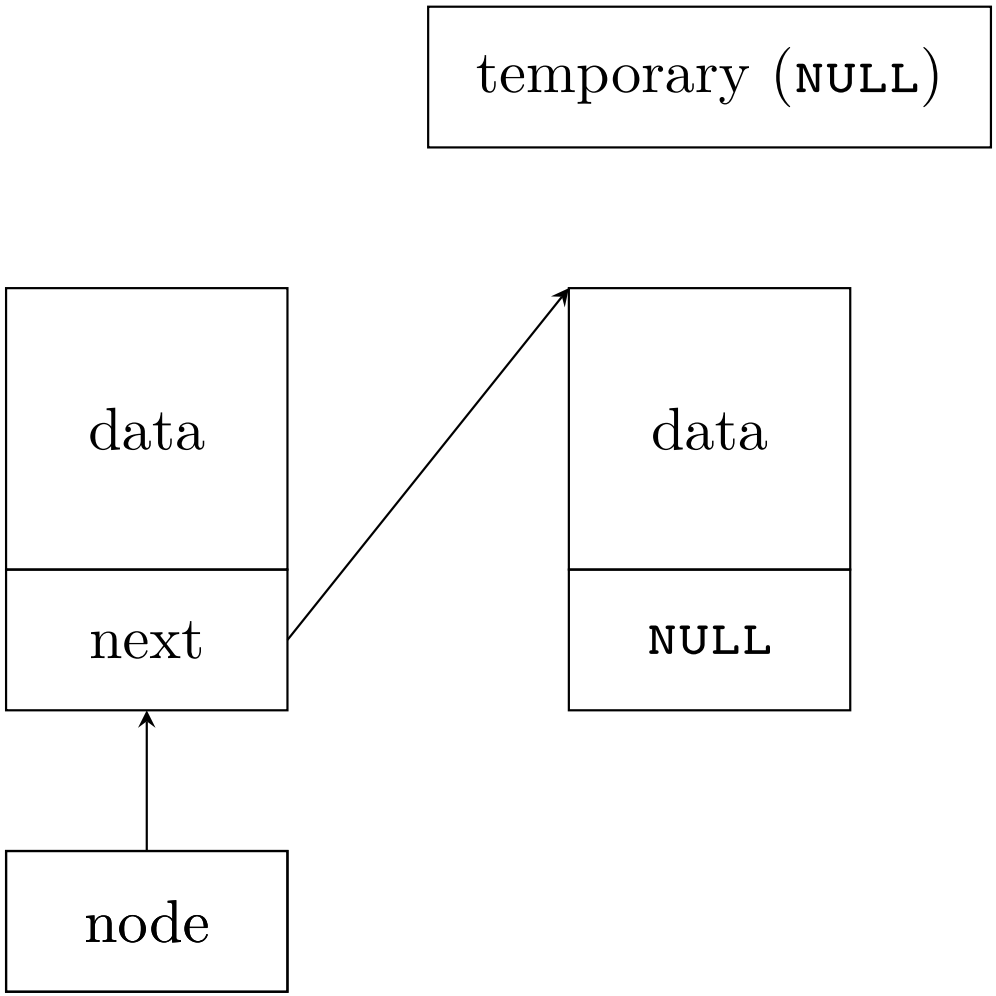
Usunięcie ostatniego węzła



Przed wykonaniem wiersza nr 6 funkcji `delete_node()`

Jednokierunkowa lista liniowa

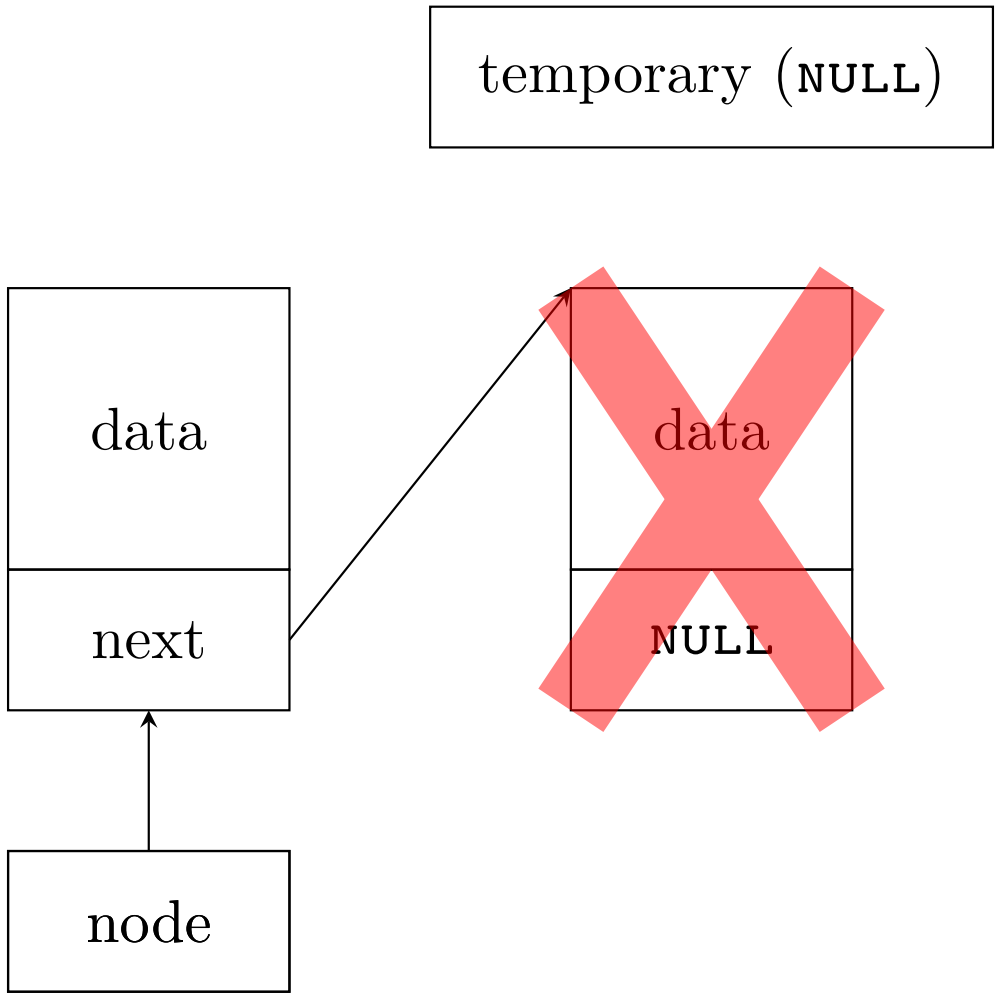
Usunięcie ostatniego węzła



Przed wykonaniem wiersza nr 7 funkcji `delete_node()`

Jednokierunkowa lista liniowa

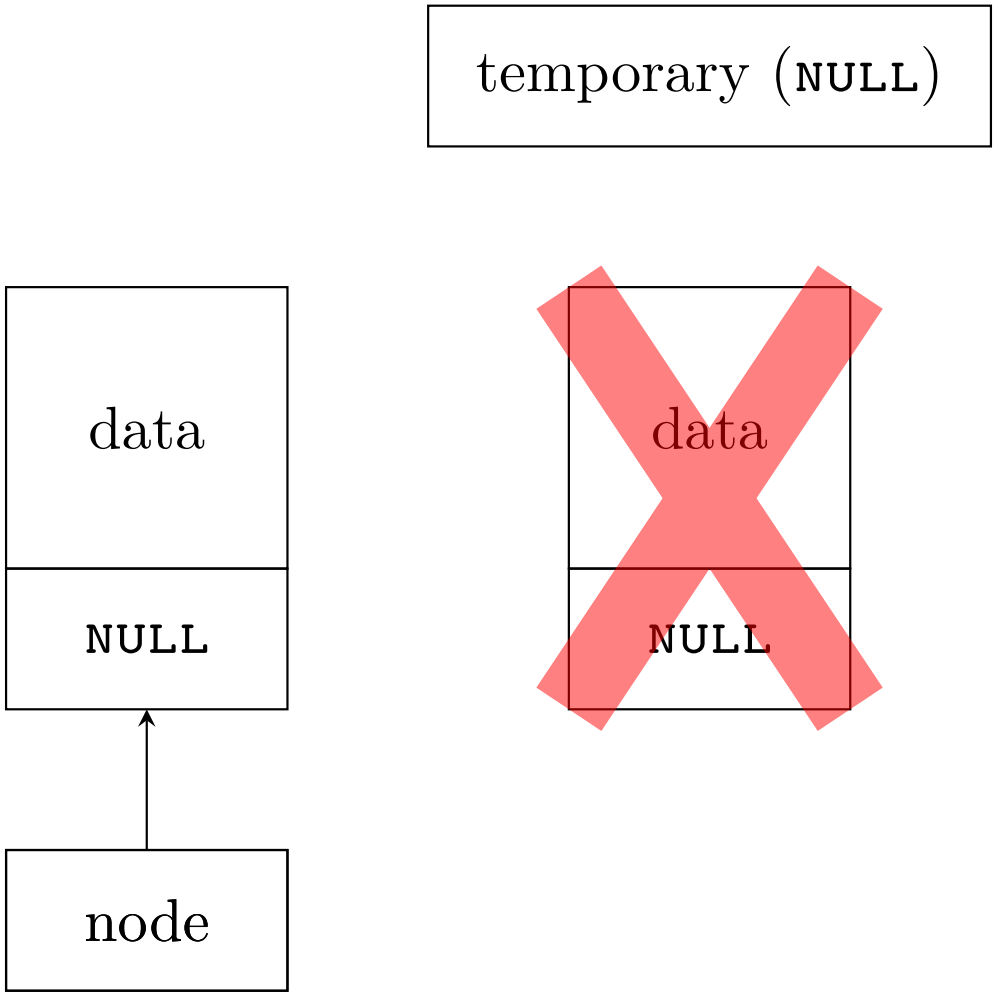
Usunięcie ostatniego węzła



Przed wykonaniem wiersza nr 8 funkcji `delete_node()`

Jednokierunkowa lista liniowa

Usunięcie ostatniego węzła



Po wykonaniu wiersza nr 8 funkcji `delete_node()`

Jednokierunkowa lista liniowa

Funkcja `print_list()`

```
1  void print_list(struct list_node *node)
2  {
3      while(node) {
4          printf("%d ", node->data);
5          node = node->next;
6      }
7      puts("");
8  }
```

Jednokierunkowa lista liniowa

Funkcja `print_list()`

Funkcja `print_list()` wypisuje liczby przechowywane w liście na ekranie. Jako argument wywołania należy do niej przekazać adres pierwszego węzła listy. Jeśli lista będzie pusta, to funkcja nie wypisze żadnego komunikatu. Ponieważ w funkcji nie ma potrzeby modyfikowania ani wskaźnika listy, ani pól wskaźnikowych `next`, to jej parametrem jest pojedynczy wskaźnik. Nie zwraca ona także żadnej wartości. Wewnątrz funkcji wykonywana jest pętla `while` (wiersze 3–6), która iteruje po liście. Czyni to tak długo, jak długo parametr `node` ma wartość różną od `NULL`. Wewnątrz tej pętli wypisywana jest liczba z pola `data` węzła bieżąco wskazywanego przez `node` (wiersz nr 4), a następnie ten wskaźnik jest przestawiany na kolejny węzeł listy (wiersz nr 5).

Jednokierunkowa lista liniowa

Funkcja `remove_list()`

```
1  void remove_list(struct list_node **node)
2  {
3      while(*node) {
4          struct list_node *temporary = (*node)->next;
5          free(*node);
6          *node = temporary;
7      }
8  }
```

Jednokierunkowa lista liniowa

Funkcja `remove_list()`

Funkcja `remove_list()` odpowiedzialna jest za usunięcie listy, czyli zwolnienie pamięci przeznaczonej na każdy z jej węzłów. Ponieważ modyfikuje ona strukturę listy, to posiada parametr (`node`), który jest podwójnym wskaźnikiem. Za ten parametr, w miejscu jej wywołania należy podstawić *adres wskaźnika listy*. Funkcja `remove_list()` nie zwraca żadnej wartości.

Aby usnąć listę `remove_list()` wykonuje pętlę `while` (wiersze 3–7), podobną do tej z `print_list()`. Jednakże w jej warunku jest sprawdzane, czy *wskaźnik wskazywany przez* `node` nie jest pusty. Wewnątrz pętli adres kolejnego węzła na liście (o ile on istnieje), jest zapamiętywany w zmiennej `temporary` (wiersz nr 4), następnie węzeł wskazywany przez wskaźnik, którego adres znajduje się w `node` jest usuwany (wiersz nr 5) i do wskaźnika, który go wskazywał zapisywany jest adres przechowywany w `temporary`. W ten sposób zostaną usunięte wszystkie węzły listy, począwszy od pierwszego, a skończywszy na ostatnim.

Jednokierunkowa lista liniowa

Funkcja `main()`, część 1

```
1  int main(void)
2  {
3      for(int i=1; i<5; i++)
4          if(add_node(&list_pointer,i)==-1)
5              fprintf(stderr,"Błąd dodawania elementu o
   ↪  wartości %d do listy!\n",i);
6      for(int i=6; i<10; i++)
7          if(add_node(&list_pointer,i)==-1)
8              fprintf(stderr,"Błąd dodawania elementu o
   ↪  wartości %d do listy!\n",i);
9      print_list(list_pointer);
10     if(add_node(&list_pointer,0)==-1)
11         fprintf(stderr,"Błąd dodawania elementu o
   ↪  wartości %d do listy!\n",0);
12     print_list(list_pointer);
```

Jednokierunkowa lista liniowa

Funkcja `main()`, część 2

```
1       if(add_node(&list_pointer,5)==-1)
2           fprintf(stderr,"Błąd dodawania elementu o
    ↪   wartości %d do listy!\n",5);
3       print_list(list_pointer);
4       if(add_node(&list_pointer,7)==-1)
5           fprintf(stderr,"Błąd dodawania elementu o
    ↪   wartości %d do listy!\n",7);
6       print_list(list_pointer);
7       if(add_node(&list_pointer,10)==-1)
8           fprintf(stderr,"Błąd dodawania elementu o
    ↪   wartości %d do listy!\n",10);
9       print_list(list_pointer);
10      puts("");
```

Jednokierunkowa lista liniowa

Funkcja `main()`, część 3

```
1       delete_node(&list_pointer,0);
2       print_list(list_pointer);
3       delete_node(&list_pointer,1);
4       print_list(list_pointer);
5       delete_node(&list_pointer,1);
6       print_list(list_pointer);
7       delete_node(&list_pointer,4);
8       print_list(list_pointer);
9       delete_node(&list_pointer,7);
10      print_list(list_pointer);
11      delete_node(&list_pointer,10);
12      print_list(list_pointer);
13      remove_list(&list_pointer);
14      return 0;
15  }
```

Jednokierunkowa lista liniowa

Funkcja `main()`

W funkcji `main()` programu zostały wywołane wszystkie opisane wcześniej (oprócz `create_and_add_node()`, która jest wywoływana z poziomu `add_node()`) tak, aby przetestować je w działaniu. Najpierw od listy dodawane są w pętli `for` węzły o wartościach od 1 do 5 (wiersze 3–5, slajd nr 37). Proszę zwrócić uwagę, na sposób wywołania funkcji `add_node()`. W każdej iteracji `for` sprawdzane jest, czy nie zwróciła ona liczby -1 oznaczającej wyjątek. Jeśli tak by się stało, to program wypisałby na ekranie stosowny komunikat. Proszę także zwrócić uwagę na pierwszy argument tej funkcji — tak jak było to opisane wcześniej, jest nim adres wskaźnika listy. Kolejna pętla `for` (wiersze 6–8, slajd nr 37) dodaje do listy węzły zawierające liczby od 6 do 10. Następnie, wywoływana jest funkcja `print_list()` (wiersz nr 9, slajd nr 37), która powinna wypisać na ekranie wszystkie wymienione wcześniej liczby.

Jednokierunkowa lista liniowa

Funkcja `main()`

Dalej, w funkcji `main()` dodawane są do listy kolejne węzły zawierające liczby 0 (wiersze 10–11, slajd nr 37), 5 (wiersze 1–2, slajd nr 38), 7 (wiersze 4–5, slajd nr 38) i 10 (wiersze 7–8, slajd nr 38). Po każdej takiej operacji wywoływana jest funkcja `print_list()`. Liczby w nowych węzłach zostały dobrane tak, aby przetestować, czy funkcja `add_node()` prawidłowo dodaje węzły na początku listy (liczba 0), wewnątrz listy (liczba 5), wewnątrz listy, jeśli już istnieje węzeł o takiej samej wartości (liczba 7) i na końcu listy (liczba 10).

Jednokierunkowa lista liniowa

Funkcja `main()`

Po zakończeniu dodawania węzłów funkcja `main()` rozpoczyna ich usuwanie przy pomocy `delete_node()`. Najpierw usuwany jest węzeł przechowujący liczbę 0 (wiersz nr 1, slajd nr 39), celem sprawdzenia, czy funkcja `delete_node()` prawidłowo usuwa pierwszy węzeł. Potem usuwany jest węzeł zawierający liczbę 1, czyli ponownie pierwszy na liście (wiersz nr 3, slajd nr 39), potem następuje próba ponownego usunięcia węzła zawierającego liczbę 1 (wiersz nr 5, slajd nr 39). Tym razem takiego węzła nie ma w liście, ale pozwala to sprawdzić, czy `delete_node()` prawidłowo obsłuży taki przypadek. Następnie usuwany jest węzeł zawierający 4 (wiersz nr 7, slajd nr 39), znajdujący się wewnątrz listy, węzeł z 7 (wiersz nr 9, slajd nr 39), który również jest wewnętrzny, ale zawiera zdublowaną wartość i węzeł z liczbą 10 (wiersz nr 11, slajd nr 39), znajdujący się na końcu listy. Po każdym usunięciu węzła wywoływana jest funkcja `print_list()`, aby zobrazować zmian. Na koniec uruchamiana jest `remove_list()`, celem usunięcia pozostałych elementów listy.

Dwukierunkowa lista liniowa

Pod względem budowy dwukierunkowa lista liniowa różni się od jednokierunkowej tym, że każdy z jej węzłów zawiera adres węzła poprzedzającego go w liście, z wyjątkiem pierwszego, który nie posiada poprzednika.
Ponieważ oba programy są podobne, to w opisach funkcji drugiego z nich zostaną tylko podkreślone różnice w stosunku do analogicznych podprogramów znajdujących się w pierwszym.

Dwukierunkowa lista liniowa

Typ danych węzła

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  struct list_node
5  {
6      int data;
7      struct list_node *previous, *next;
8  } *list_pointer;
```

Dwukierunkowa lista liniowa

Typ danych węzła

Początek drugiego programu jest prawie taki sam, jak poprzedniego. Różnica polega na tym, że typ danych pojedynczego węzła zawiera dodatkową składową, która jest polem wskaźnikowym o nazwie `previous`. W tym polu będzie przechowywany adres poprzednika węzła na liście. W przypadku pierwszego węzła listy to pole będzie miało wartość `NULL`.

Możliwe jest utworzenie listy dwukierunkowej z tylko jednym polem wskaźnikowym w każdym elemencie. Taka lista nazywa się w języku angielskim XOR-list. Pozwala ona zaoszczędzić miejsce w pamięci operacyjnej, które zajmowałoby dodatkowe pole wskaźnikowe, ale wymaga złożonych operacji związanych z jej obsługą, więc jest rzadko stosowana i nie będzie omawiana na tym wykładzie.

Dwukierunkowa lista liniowa

Funkcja `create_and_add_node()`

```
1  int create_and_add_node(struct list_node **node, struct
   ↪  list_node* preceding, int number)
2  {
3      struct list_node *new_node = (struct list_node
   ↪  *)malloc(sizeof(struct list_node));
4      if(!new_node)
5          return -1;
6      new_node->data = number;
7      new_node->next = *node;
8      new_node->previous = preceding;
9      if(*node)
10         (*node)->previous = new_node;
11     *node = new_node;
12     return 0;
13 }
```

Dwukierunkowa lista liniowa

Funkcja `create_and_add_node()`

Funkcja **`create_and_add_node()`**, w stosunku do swojej odpowiedniczki dla listy jednokierunkowej, posiada dodatkowy parametr wskaźnikowy (**`preceding`**), przez który jest do niej przekazywany adres węzła zawierającego pole **`next`** wskazywane przez **`node`**, lub wartość `NULL`, w zależności od miejsca, gdzie ma być dodany nowy węzeł do listy. Funkcja ta musi także uwzględniać istnienie w nowym węźle pola **`previous`**, dlatego w wierszu nr 8 przypisuje mu wartość wskaźnika **`preceding`**. Dodatkowo, sprawdza ona czy istnieje węzeł, który będzie następnikiem nowego w liście (wiersz nr 9) i jeśli tak jest, to zapisuje w jego polu **`previous`** adres tego nowego węzła (wiersz nr 10).

Dwukierunkowa lista liniowa

Funkcja `add_node()`

```
1  int add_node(struct list_node **node, int number)
2  {
3      struct list_node *preceding = NULL;
4      while(*node != NULL && (*node)->data < number) {
5          preceding = *node;
6          node = &(*node)->next;
7      }
8      return create_and_add_node(node, preceding,
   ↪   number);
9  }
```

Dwukierunkowa lista liniowa

Funkcja `add_node()`

W porównaniu ze swoją odpowiedniczką dla jednokierunkowej listy liniowej funkcja `add_node()` zawiera lokalny wskaźnik o nazwie `preceding`, który początkowo ma wartość `NULL` (wiersz nr 3), ale w każdej iteracji pętli `while` jest mu przypisywany adres węzła zawierającego pole `next`, którego adres jest przypisywany do `node` w wierszu nr 6. Wskaźnik ten będzie pełnił rolę pomocniczą i jest przekazywany jako argument do funkcji `create_and_add_node()` (wiersz nr 8).
Jak poprzednio prześledzimy działanie tych dwóch funkcji dla czterech opisanych wcześniej przypadków dodawania nowego węzła do listy, ale tym razem zwrócimy głównie uwagę na różnicę w ich działaniu w stosunku do ich odpowiedniczek z listy jednokierunkowej.

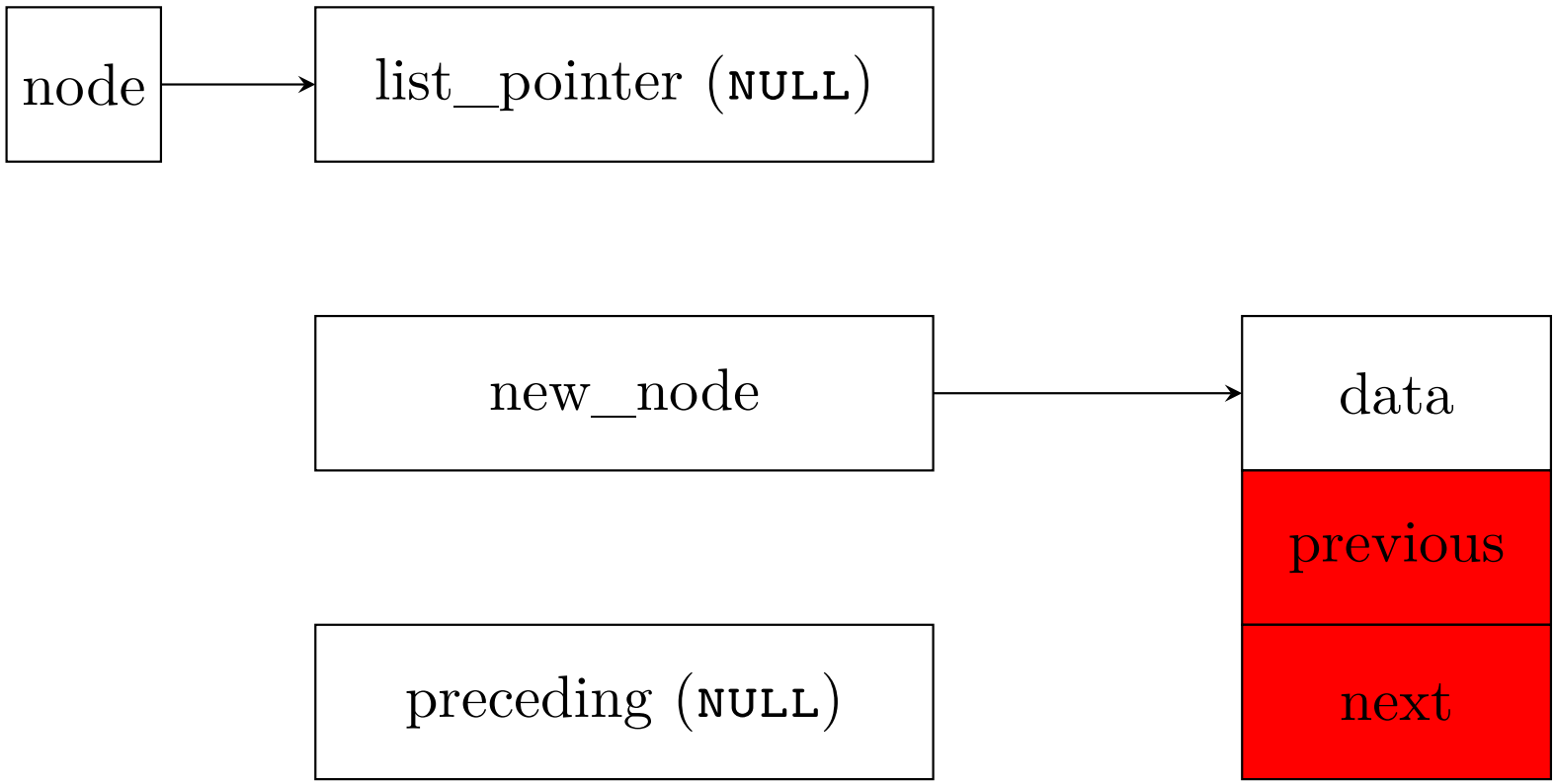
Dwukierunkowa lista liniowa

Dodanie pierwszego węzła

W przypadku dodawania pierwszego węzła do listy pętla `while` w funkcji `add_node()` od razu się zatrzyma, a `node` będzie wskazywał na pusty wskaźnik listy `list_pointer`. Wartość `NULL` będzie miał także wskaźnik `preceding`. Wywołana następnie funkcja `create_and_add_node()` po stworzeniu nowego węzła i zapisaniu w nim liczby przekazanej przez parametr `number` zapisze zarówno w jego polu `next` (wiersz nr 7), jak i `previous` (wiersz nr 8) wartość `NULL`. Ponieważ nie istnieje żaden węzeł, który będzie za nowym w liście, to warunek z wiersza nr 9 nie będzie spełniony i funkcja od razu wykona instrukcję z wiersza nr 11, zapisując tym samym we wskaźniku listy adres nowego węzła, który stanie się zarazem jej pierwszym i ostatnim. Po tym funkcja zwróci wartość 0 i zakończy działanie.

Dwukierunkowa lista liniowa

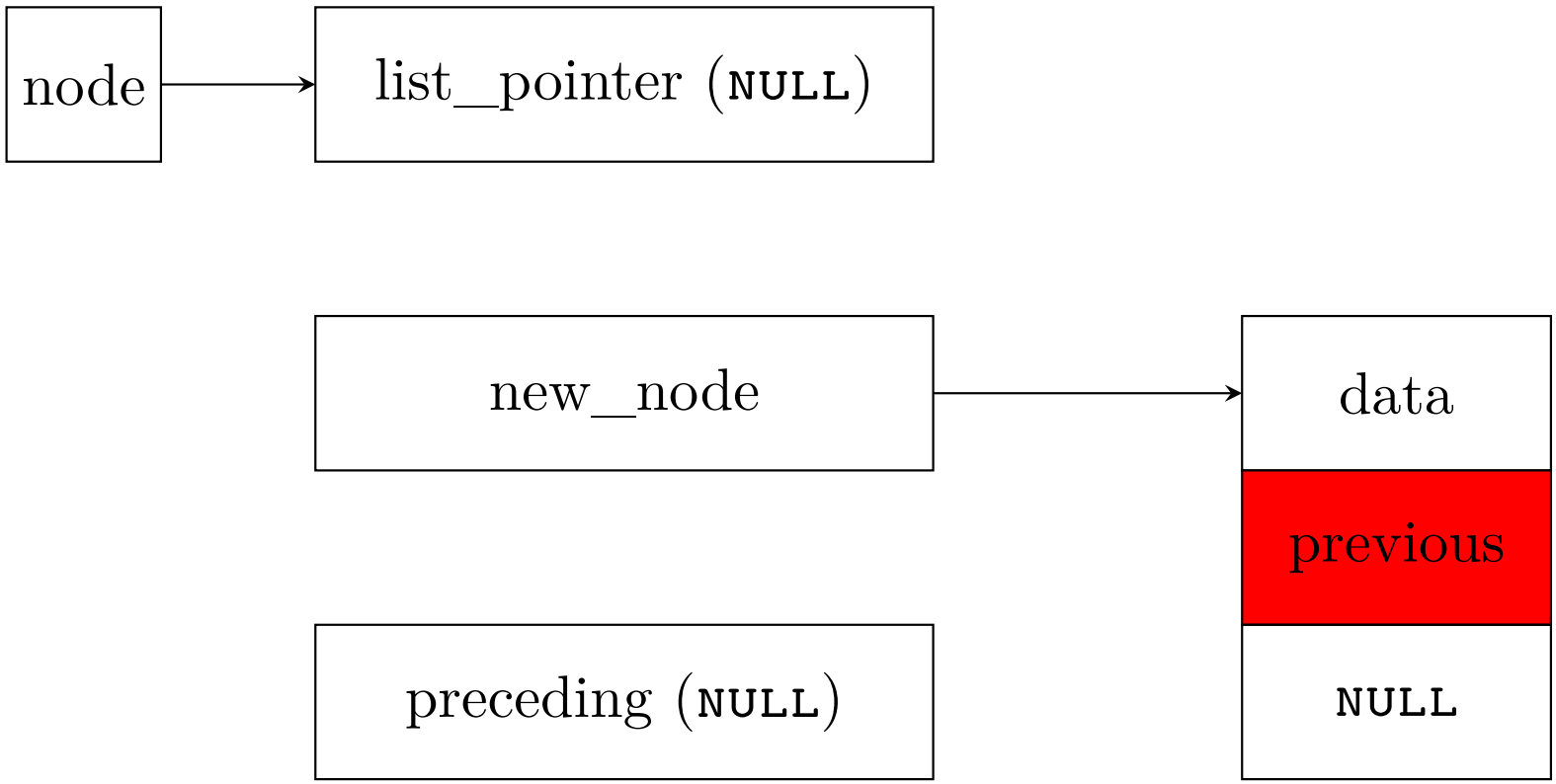
Dodanie pierwszego węzła



Przed wykonaniem wiersza nr 7 funkcji `create_and_add_node()`

Dwukierunkowa lista liniowa

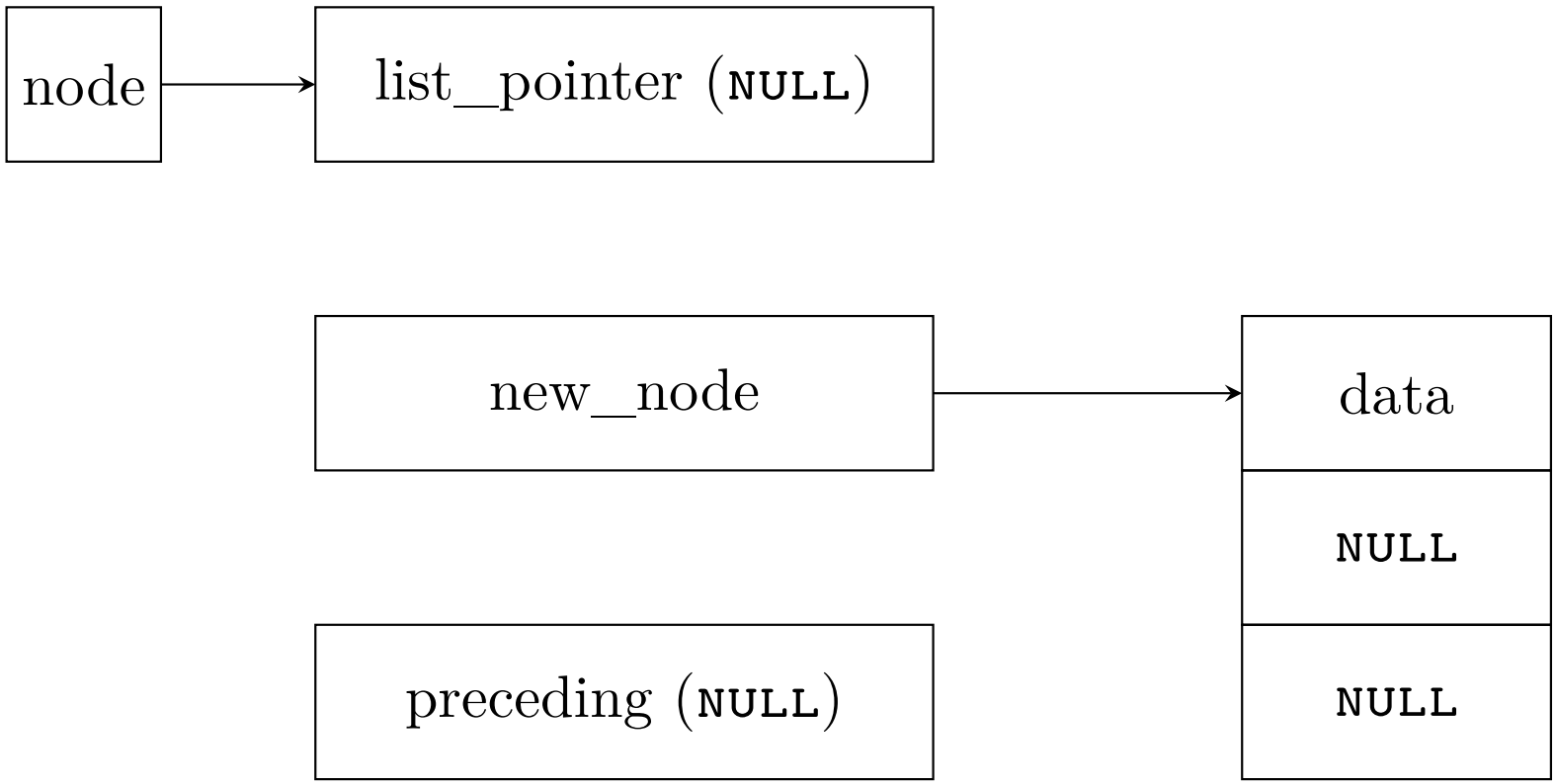
Dodanie pierwszego węzła



Przed wykonaniem wiersza nr 8 funkcji `create_and_add_node()`

Dwukierunkowa lista liniowa

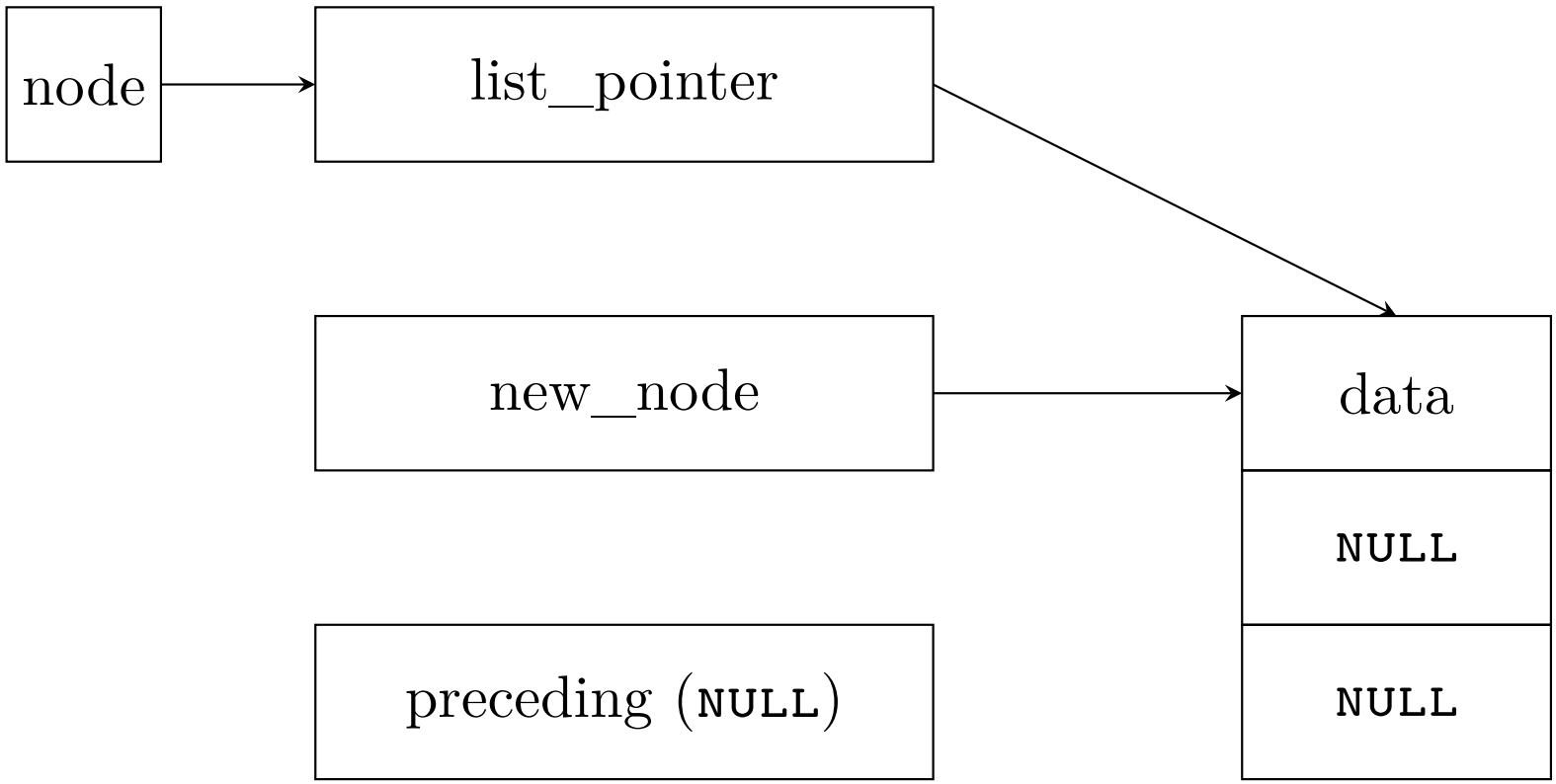
Dodanie pierwszego węzła



Przed wykonaniem wiersza nr 11 funkcji `create_and_add_node()`

Dwukierunkowa lista liniowa

Dodanie pierwszego węzła



Po wykonaniu wiersza nr 11 funkcji `create_and_add_node()`

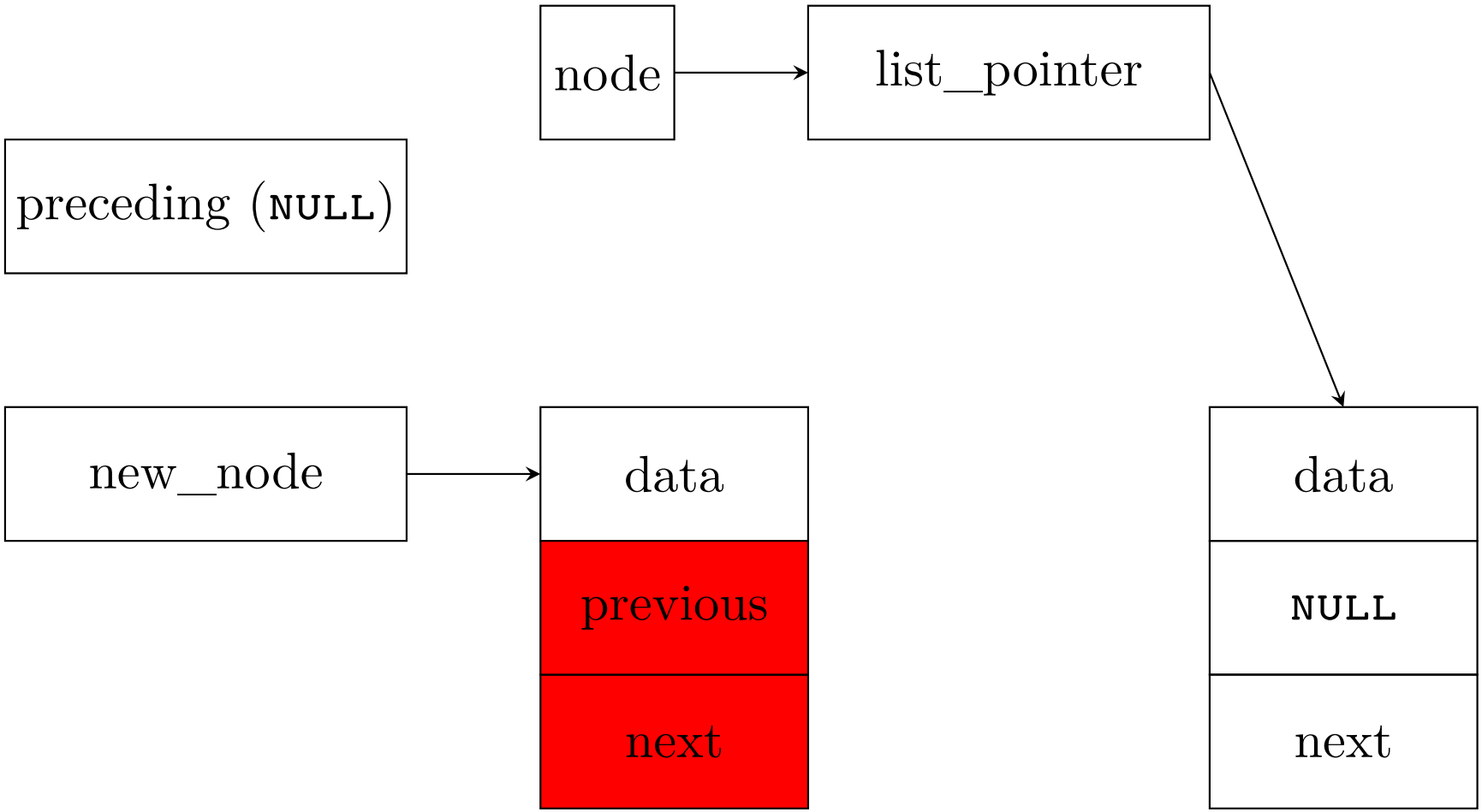
Dwukierunkowa lista liniowa

Dodanie na początku listy

W przypadku gdy nowy węzeł powinien być dodany na początku listy, po zakończeniu pętli `while` w funkcji `add_node()` wskaźnik `preceding` będzie miał wartość `NULL`, ale wskaźnik `node`, będzie wskazywał na wskaźnik listy `list_node`, w którym będzie adres pierwszego węzła listy. Funkcja `create_and_add_node()` po stworzeniu nowego węzła i zapisaniu w nim liczb przekazanej przez parameter `number`, przypisze do jego pola `next` adres bieżącego pierwszego węzła listy (wiersz nr 7), a następnie w polu `previous` umieści wartość `NULL` pobraną z parametru `preceding` (wiersz nr 8). Potem sprawdzi, że istnieje węzeł, który będzie następnikiem nowego w liście (jest to bieżący pierwszy element listy) i zapisze w polu `previous` tego węzła adres nowego (wiersz nr 10). Po wykonaniu tej operacji, zapisze ona we wskaźniku listy adres nowego węzła, ponieważ po dodaniu, to on stał się pierwszy na liście (wiersz nr 11).

Dwukierunkowa lista liniowa

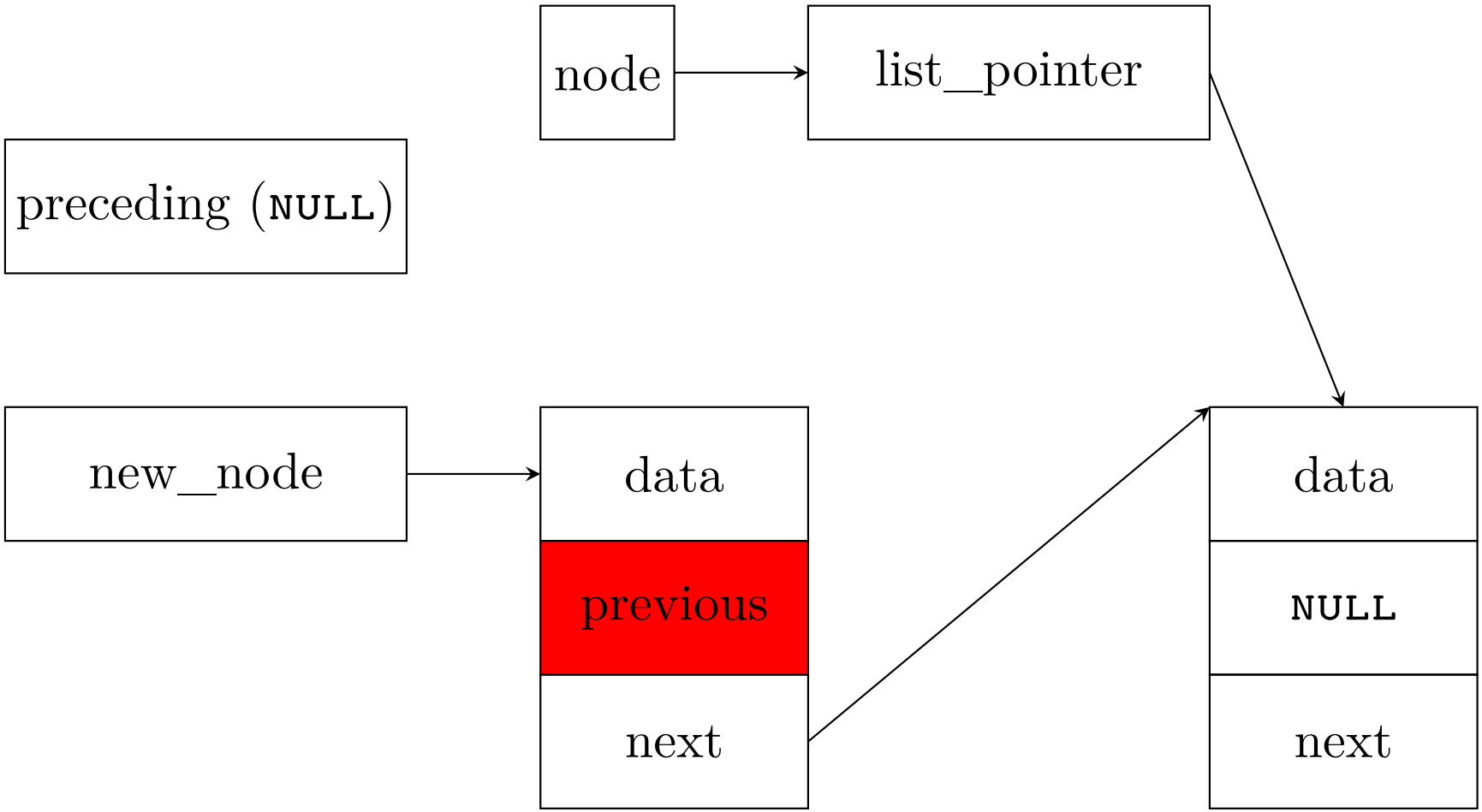
Dodanie na początku listy



Przed wykonaniem wiersza nr 7 funkcji `create_and_add_node()`

Dwukierunkowa lista liniowa

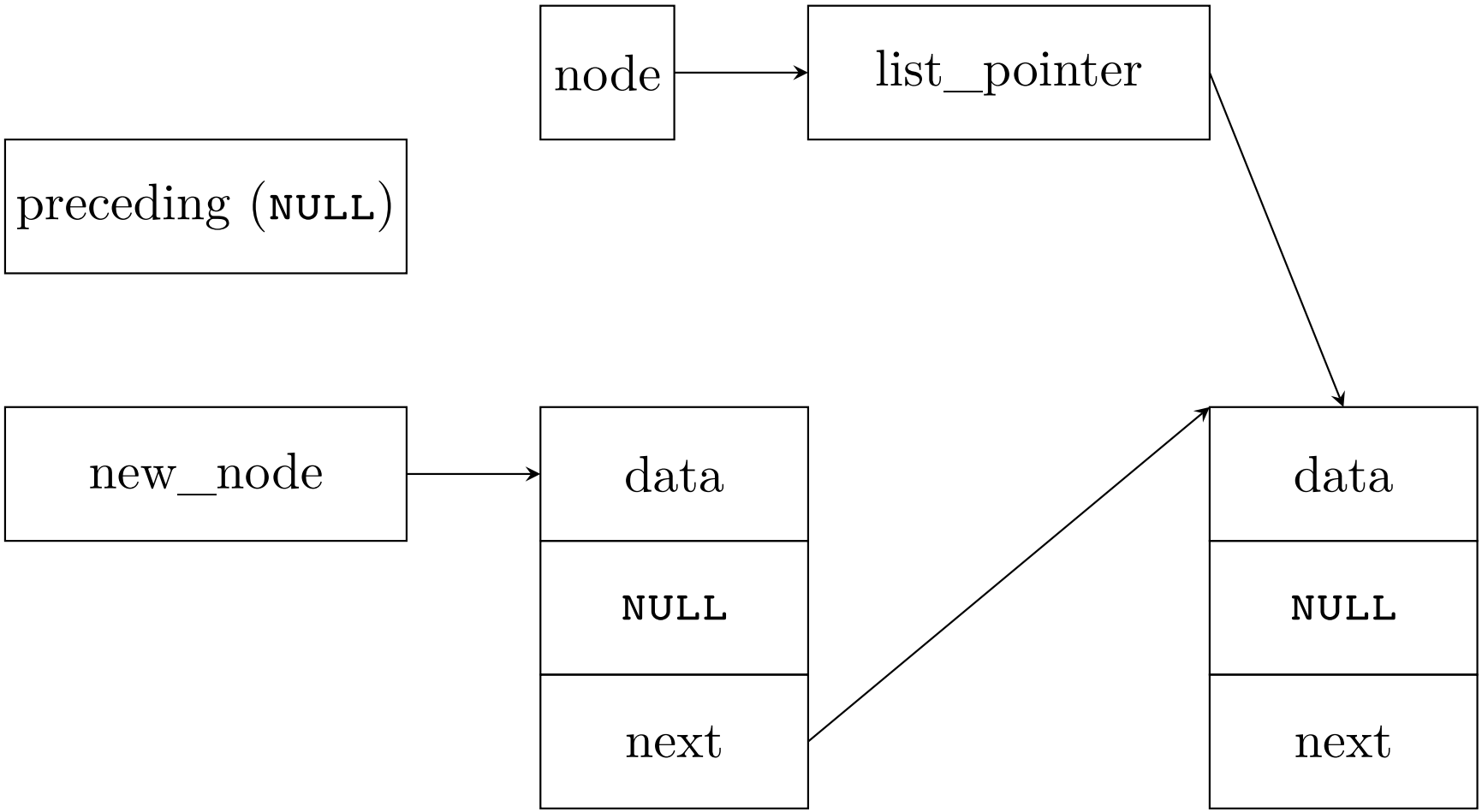
Dodanie na początku listy



Przed wykonaniem wiersza nr 8 funkcji `create_and_add_node()`

Dwukierunkowa lista liniowa

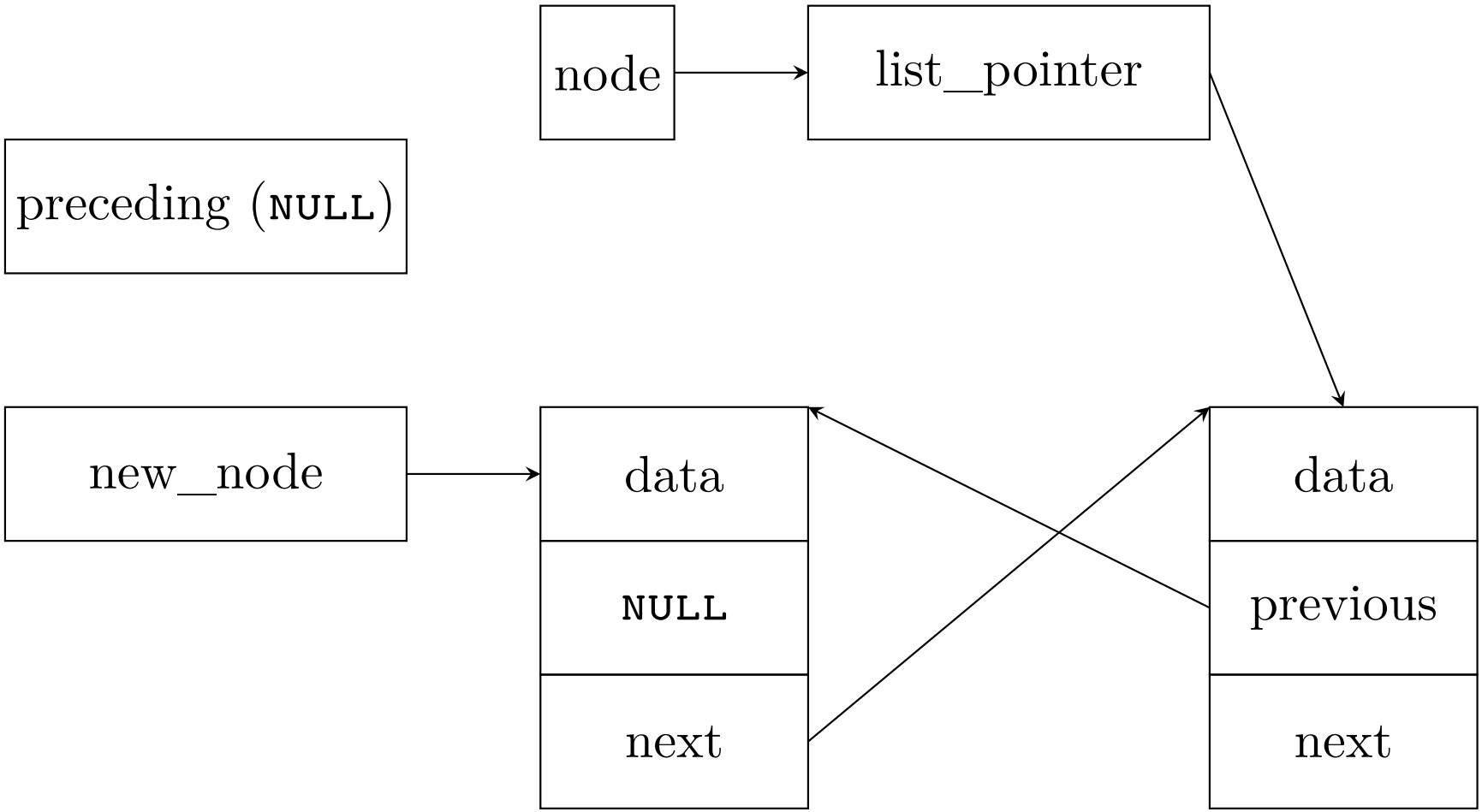
Dodanie na początku listy



Przed wykonaniem wiersza nr 10 funkcji `create_and_add_node()`

Dwukierunkowa lista liniowa

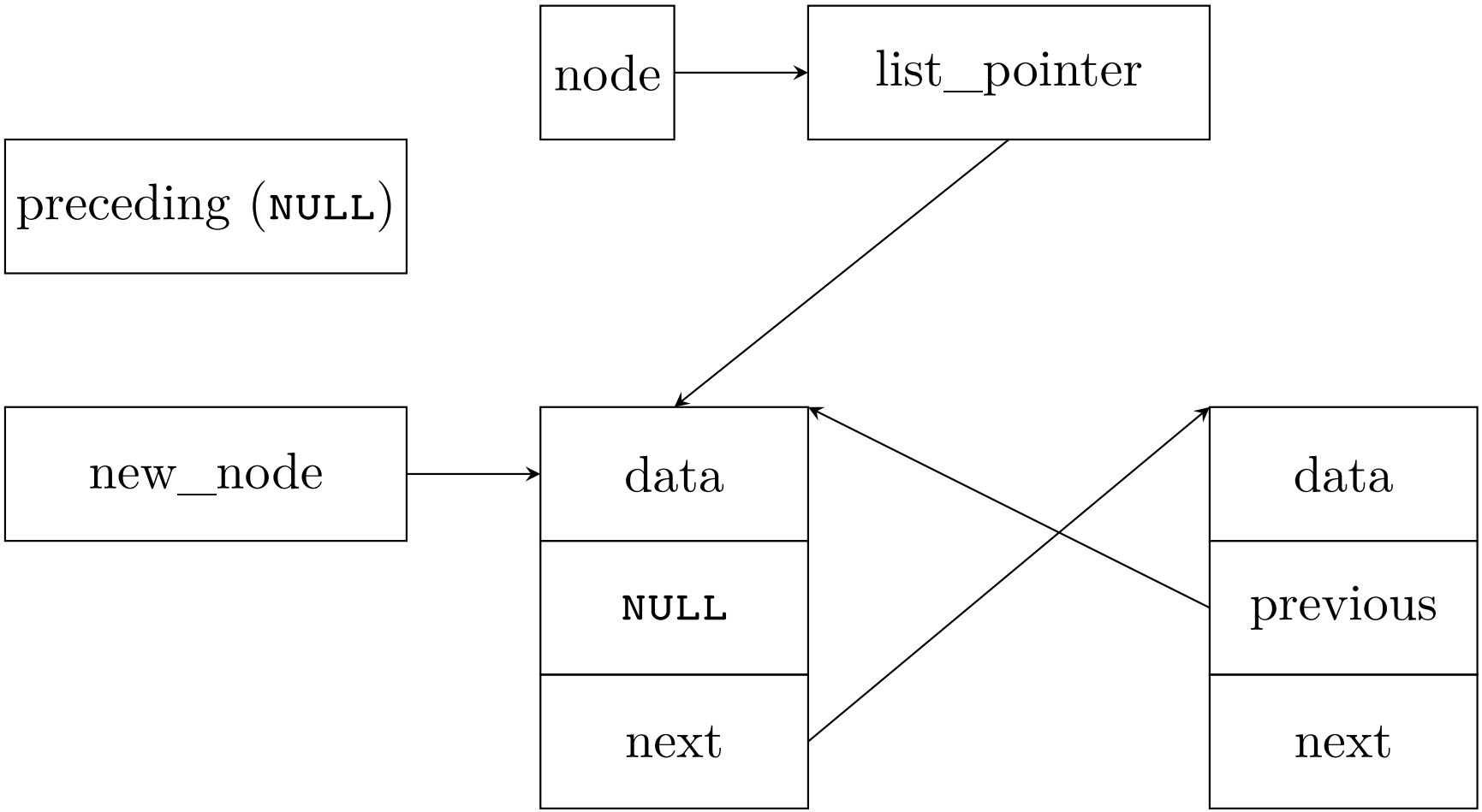
Dodanie na początku listy



Przed wykonaniem wiersza nr 11 funkcji `create_and_add_node()`

Dwukierunkowa lista liniowa

Dodanie na początku listy



Po wykonaniu wiersza nr 11 funkcji `create_and_add_node()`

Dwukierunkowa lista liniowa

Dodanie wewnątrz

Jeśli nowy węzeł powinien zostać dodany wewnątrz listy, to po zakończeniu pętli `while` w funkcji `add_node()` wskaźnik `preceding` będzie zawierał adres węzła *za* którym trzeba będzie wstawić nowy, a `node` będzie zawierał adres pola `next` tego węzła. Funkcja `create_and_add_node()`, po stworzeniu nowego węzła i zapisaniu w nim liczby, umieści w polu `next` tego węzła adres elementu listy wskazywanego przez pole `next`, którego adres znajduje się w `node` (wiersz nr 7). Zauważmy, że ostatnie wymienione pole `next` należy do węzła wskazywanego przez `preceding`.

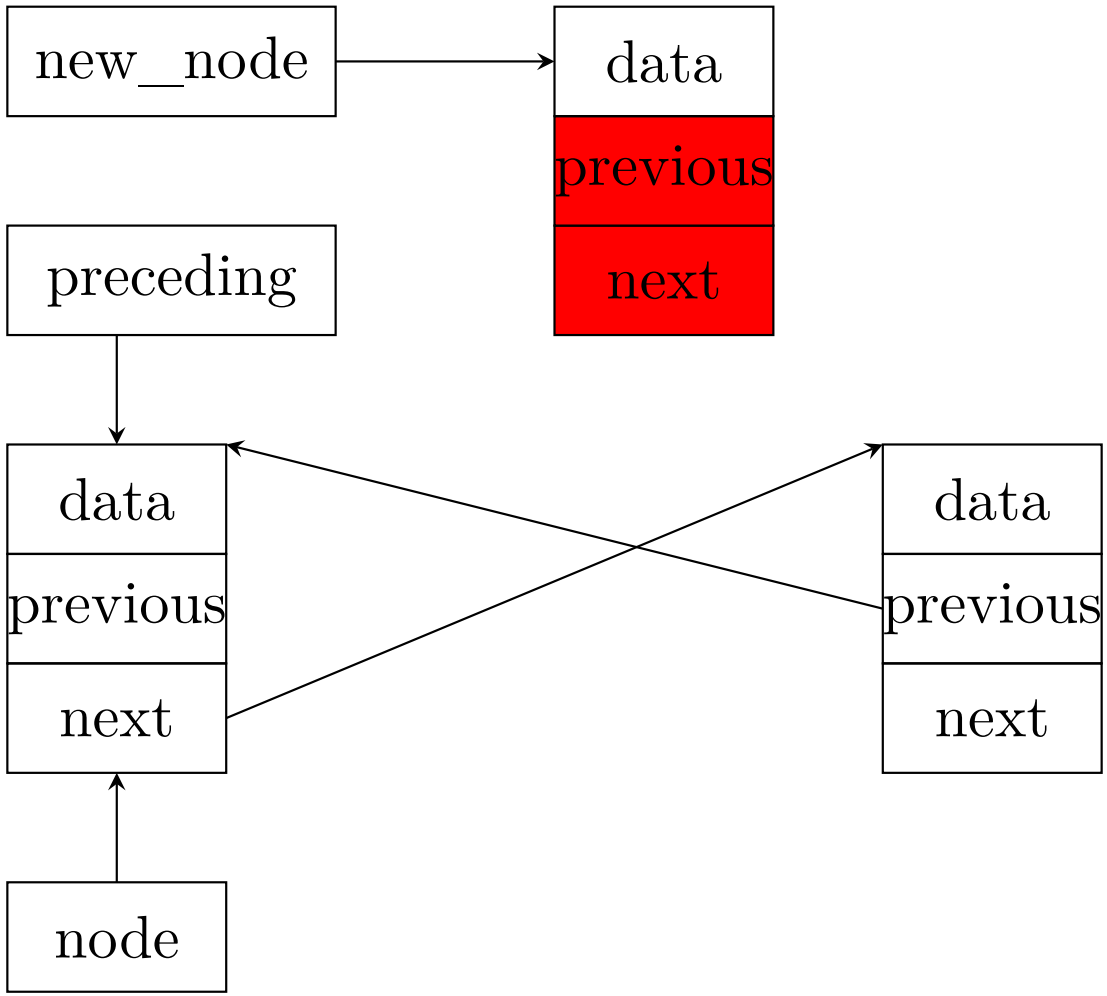
Dwukierunkowa lista liniowa

Dodanie wewnątrz

Następnie w polu `previous` nowego elementu funkcja zapisze wartość ze wskaźnika `preceding`, czyli adres węzła, który będzie poprzedzał nowy w liście (wiersz nr 8). Potem sprawdzi ona, że istnieje węzeł, który będzie następnikiem nowego na liście (wiersz nr 9) i zapisze w jego polu `previous` adres nowego elementu (wiersz nr 10). Ten sam adres zostanie zapisany także w polu `next` węzła wskazywanego przez `preceding`. Dzięki temu nowy węzeł zostanie prawidłowo dodany do listy i funkcja zakończy swe działanie zwracając 0.

Dwukierunkowa lista liniowa

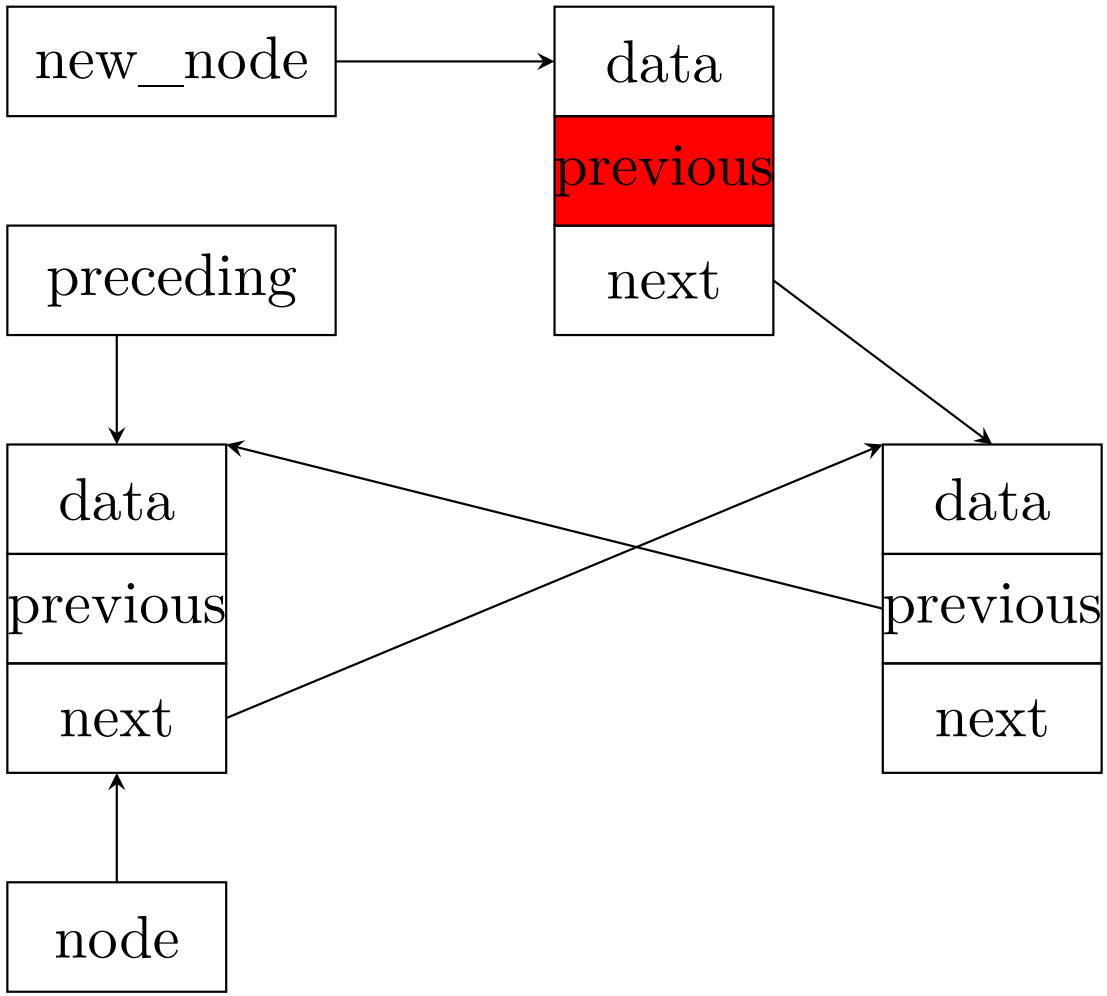
Dodanie wewnątrz



Przed wykonaniem wiersza nr 7 funkcji `create_and_add_node()`

Dwukierunkowa lista liniowa

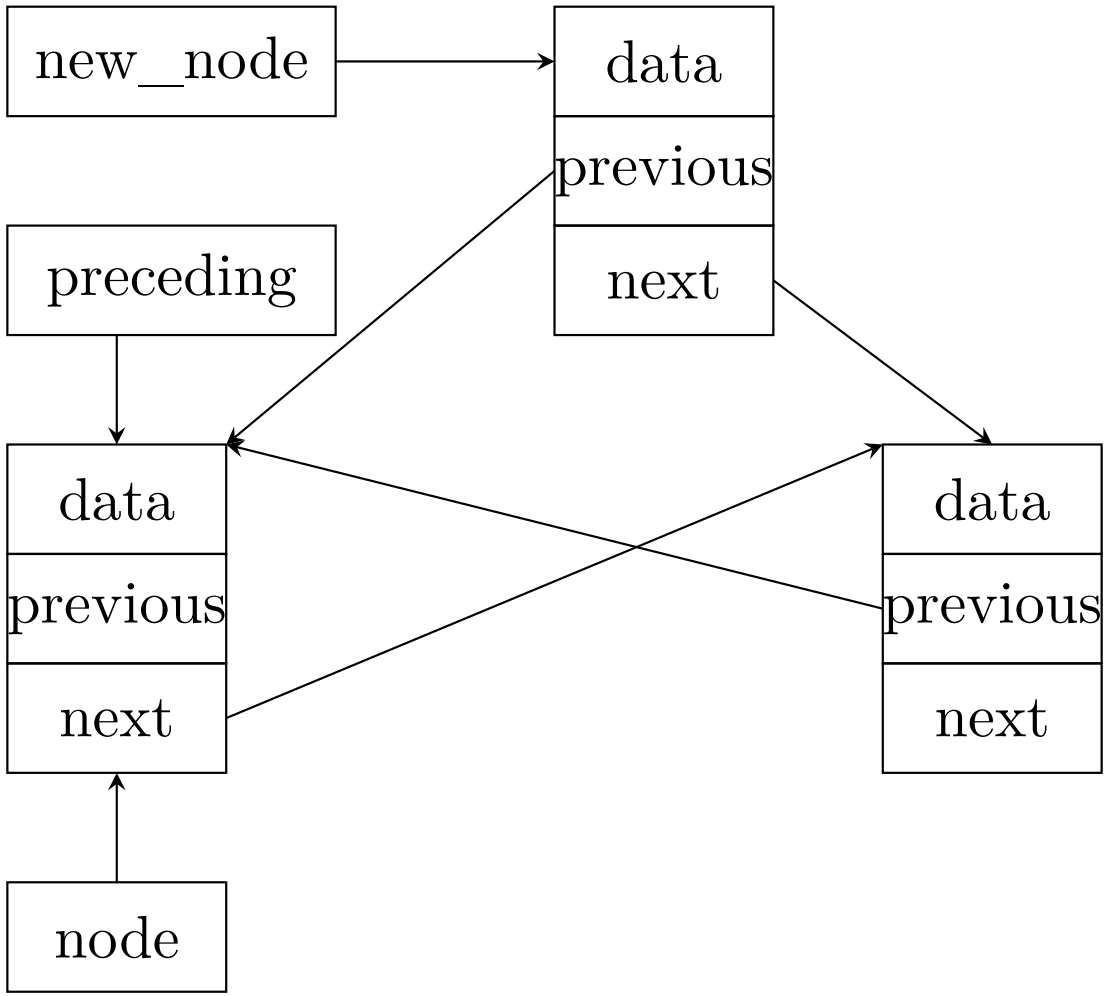
Dodanie wewnątrz



Przed wykonaniem wiersza nr 8 funkcji `create_and_add_node()`

Dwukierunkowa lista liniowa

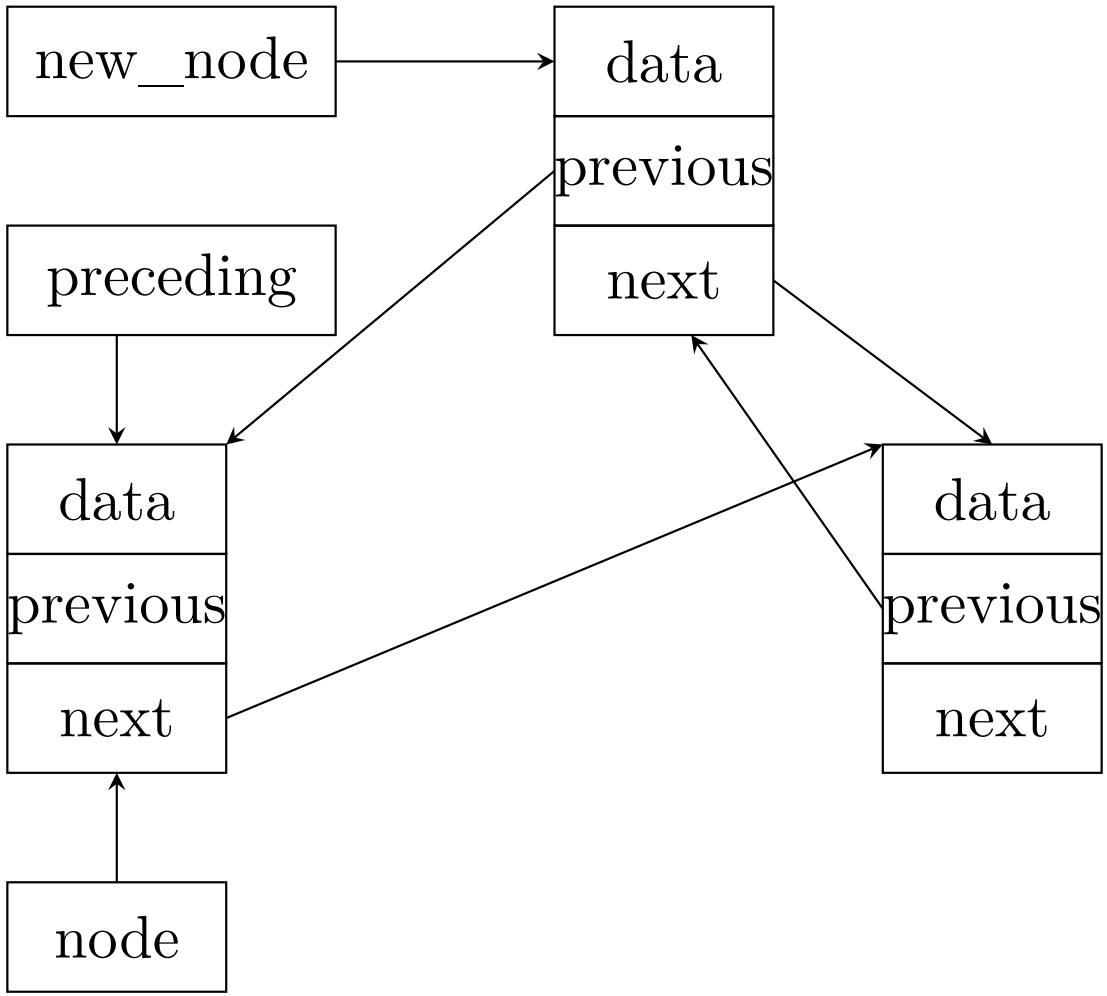
Dodanie wewnątrz



Przed wykonaniem wiersza nr 10 funkcji `create_and_add_node()`

Dwukierunkowa lista liniowa

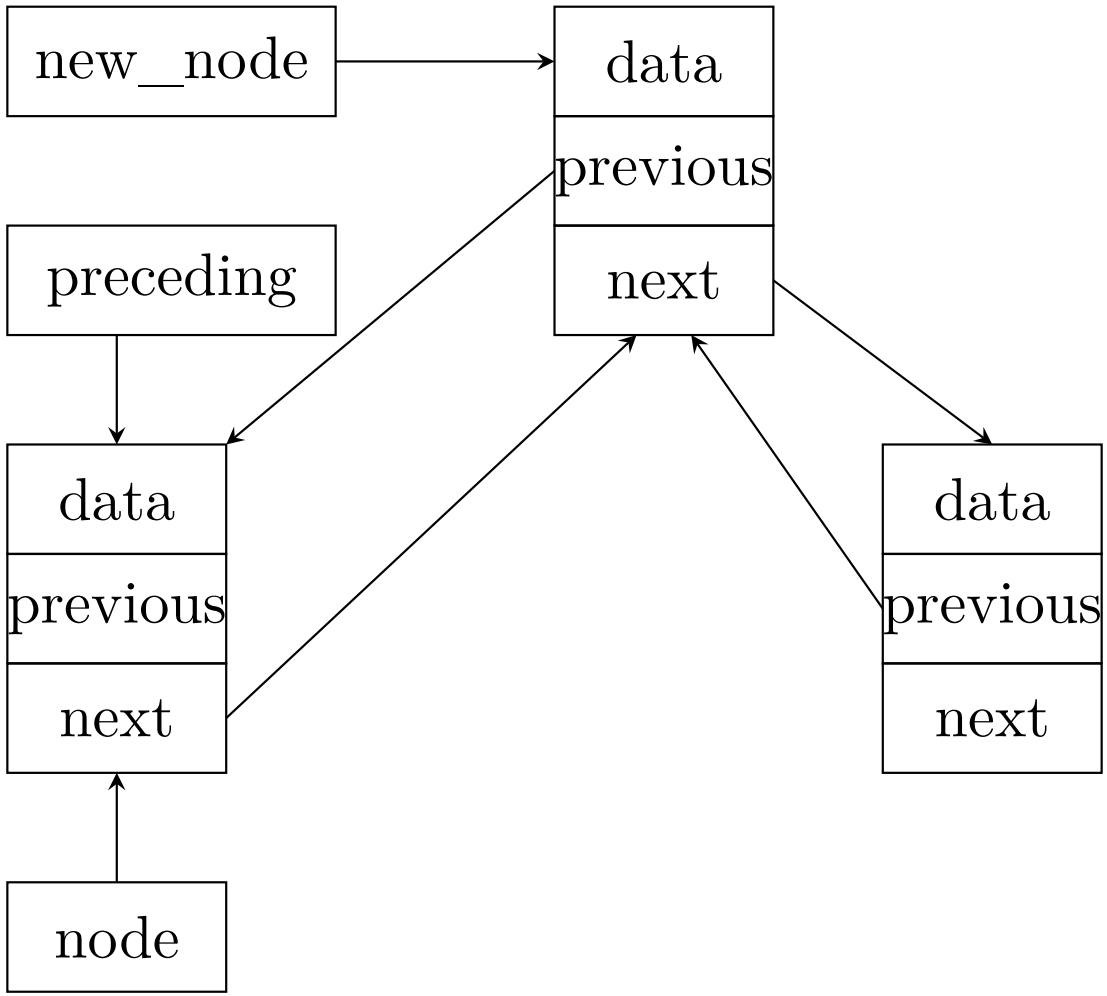
Dodanie wewnątrz



Przed wykonaniem wiersza nr 11 funkcji `create_and_add_node()`

Dwukierunkowa lista liniowa

Dodanie wewnątrz



Po wykonaniu wiersza nr 11 funkcji `create_and_add_node()`

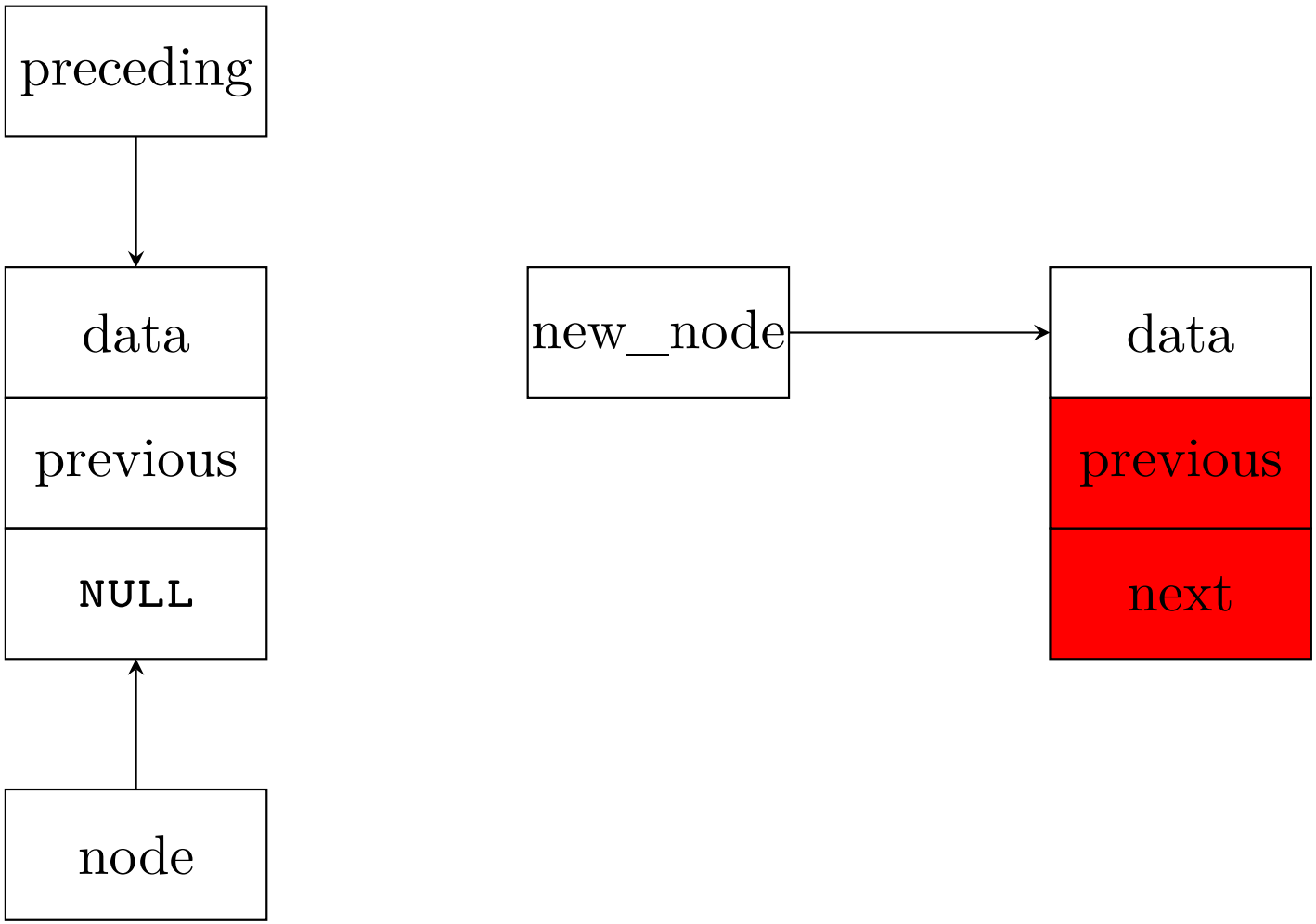
Dwukierunkowa lista liniowa

Dodanie na końcu

W przypadku, gdy nowy węzeł ma być dodany na końcu listy, to po zakończeniu pętli `while` w funkcji `add_node()` wskaźnik `preceding` zawiera adres ostatniego węzła w liście, a `node` adres jego pola `next`. Funkcja `create_and_add_node()` po utworzeniu nowego węzła i przypisaniu do jego pola `data` liczby przekazanej przez parametr `number`, zapisze w polu `next` tego węzła wartość `NULL`, gdyż taką będzie miało pole, którego adres jest we wskaźniku `node` (wiersz nr 7). Następnie w polu `previous` nowego węzła funkcja zapisze adres węzła wskazywanego przez `preceding` (wiersz nr 8). Warunek z wiersza nr 9 nie będzie spełniony, gdyż nowy węzeł nie będzie miał następnika, będzie ostatni w liście. Zatem funkcja od razu wykona wiersz nr 11, zapisując w polu `next` węzła wskazywanego przez `preceding` adres nowego elementu. Na tym zakończy się operacja dodawania elementu na koniec listy.

Dwukierunkowa lista liniowa

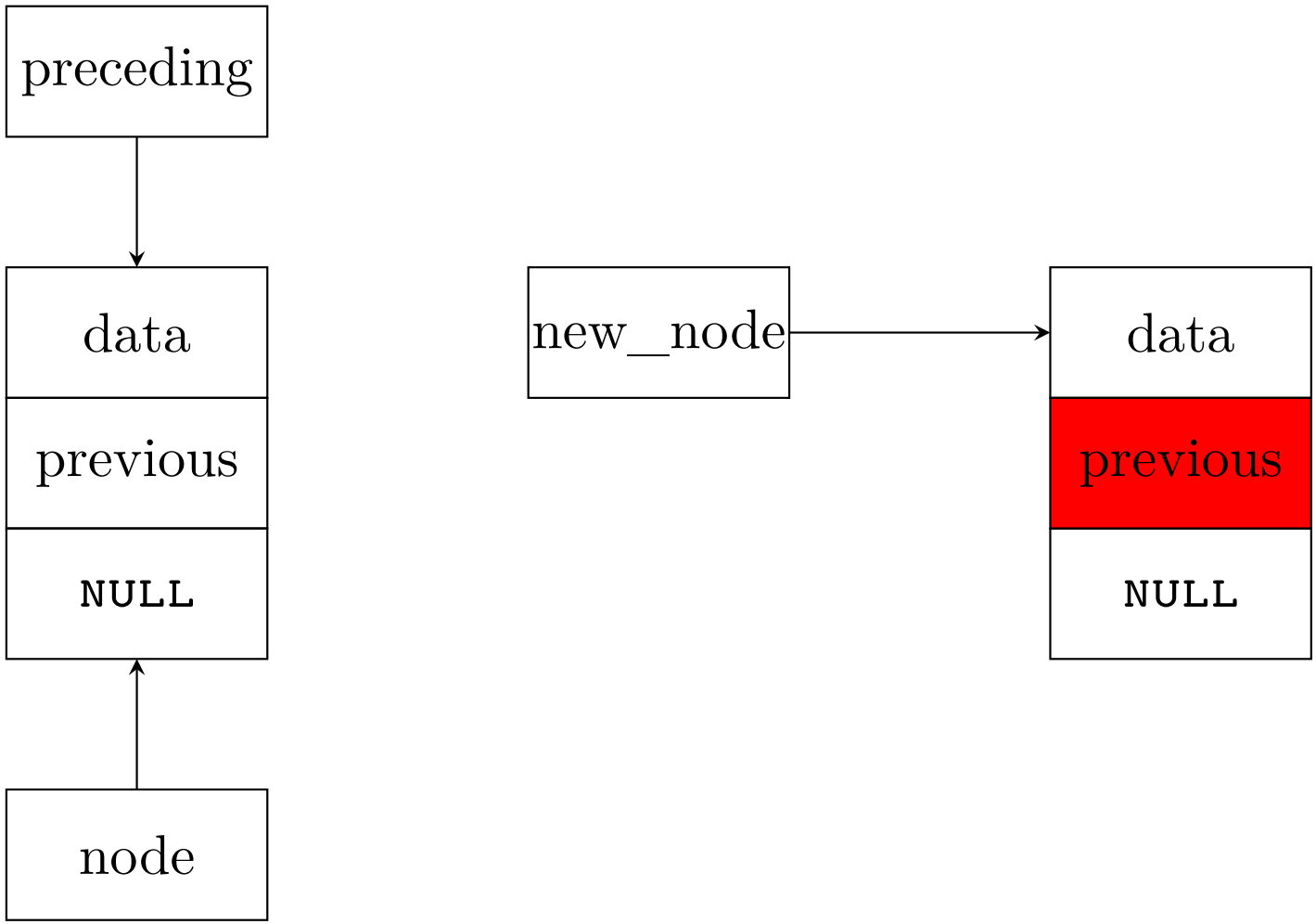
Dodanie na końcu



Przed wykonaniem wiersza nr 7 funkcji `create_and_add_node()`

Dwukierunkowa lista liniowa

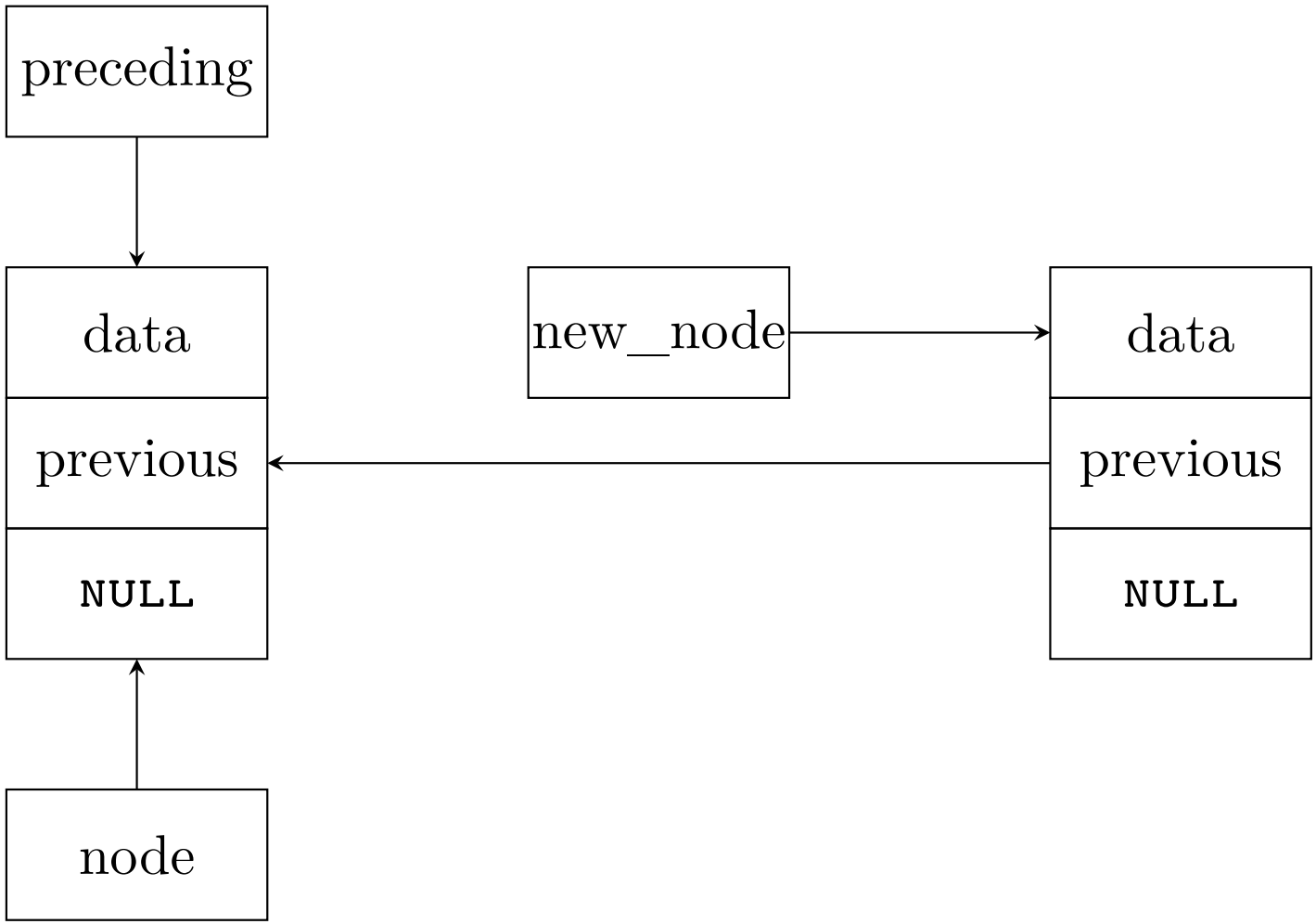
Dodanie na końcu



Przed wykonaniem wiersza nr 8 funkcji `create_and_add_node()`

Dwukierunkowa lista liniowa

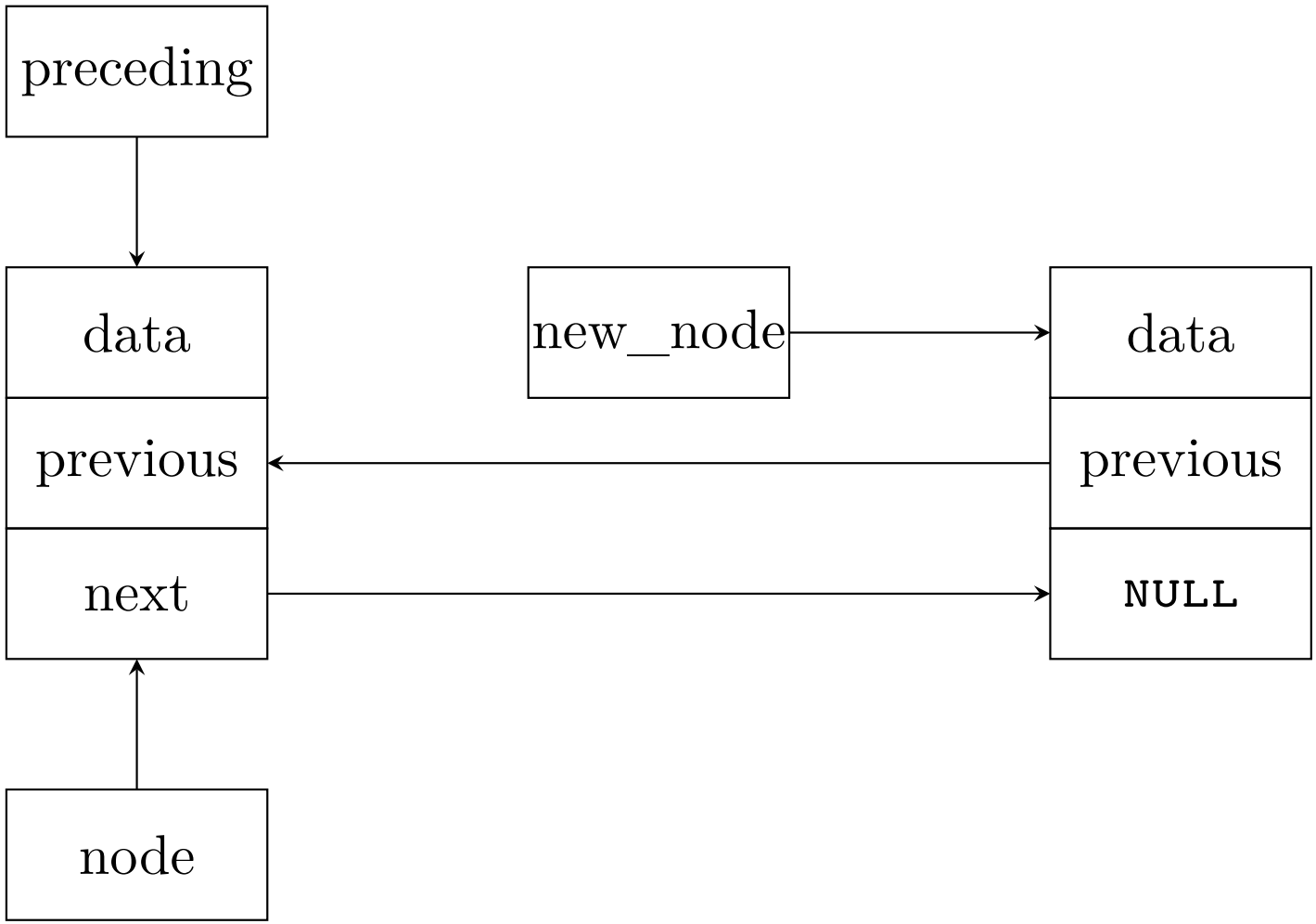
Dodanie na końcu



Przed wykonaniem wiersza nr 11 funkcji `create_and_add_node()`

Dwukierunkowa lista liniowa

Dodanie na końcu



Po wykonaniu wiersza nr 11 funkcji `create_and_add_node()`

Dwukierunkowa lista liniowa

Funkcja `delete_node()`

```
1  void delete_node(struct list_node **node, int number)
2  {
3      while(*node && (*node)->data != number)
4          node = &(*node)->next;
5      if(*node) {
6          struct list_node *temporary = (*node)->next;
7          if((*node)->next)
8              (*node)->next->previous =
   ↪   (*node)->previous;
9          free(*node);
10         *node = temporary;
11     }
12 }
```

Dwukierunkowa lista liniowa

Funkcja `delete_node()`

Funkcja `delete_node()` w wersji dla listy dwukierunkowej musi uwzględniać to, że w węzłach listy, które potencjalnie sąsiadują z usuwanym elementem są pola wskaźnikowe `previous`. Skutkuje to tym, że ma ona dodatkową instrukcję warunkową, w stosunku do jej odpowiedniczki dla jednokierunkowej listy (wiersze 7–8), która sprawdza, czy istnieje następnik usuwanego węzła i jeśli tak jest, to zapisuje w jego polu `previous` adres węzła, który poprzedza ten usuwany. Dzięki tym dodatkowym operacjom węzeł jest prawidłowo wyłączany z listy i może być bezpiecznie usunięty.
Przeanalizujemy działanie tej funkcji, dla tych samych przypadków, które zostały opisane dla jej odpowiedniczki wykonującej tę samą operację dla jednokierunkowej listy liniowej.

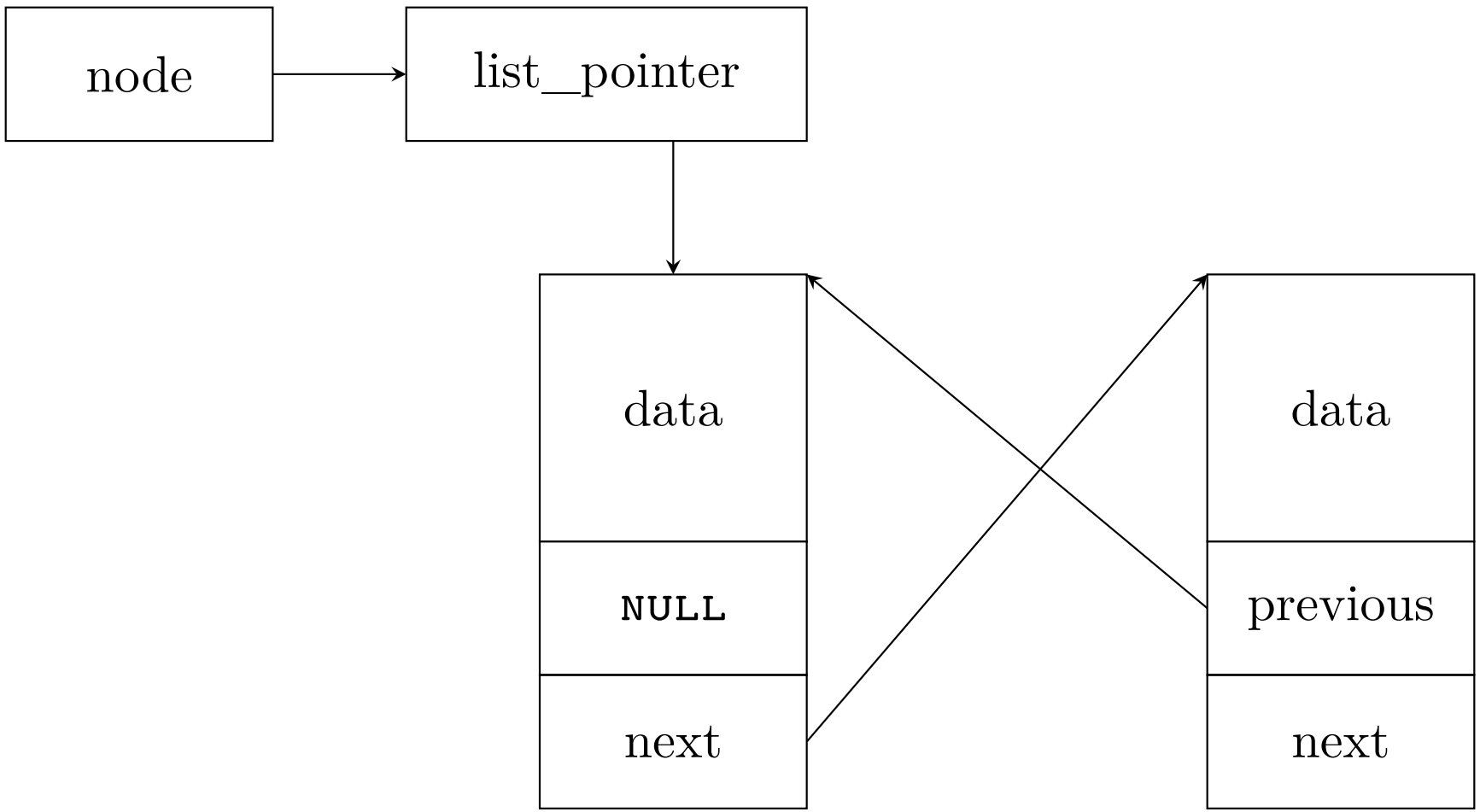
Dwukierunkowa lista liniowa

Usunięcie pierwszego węzła

W przypadku kiedy ma zostać usunięty pierwszy węzeł z listy, to po wykonaniu pętli `while` parametr `node` wskazuje na wskaźnik listy (`list_pointer`), który zawiera adres pierwszego węzła. Funkcja zapisuje w zmiennej `temporary` adres następnego węzła na liście (lub `NULL`, jeśli ten nie istnieje), który pobiera z pola `next` pierwszego węzła (wiersz nr 6). Następnie sprawdza ona, czy ten element faktycznie istnieje (wiersz nr 7) i jeśli tak, to w jego polu `previous` zapisuje adres, który jest w polu o takiej samej nazwie, ale znajdującym się w pierwszym elemencie. W opisywanym przypadku będzie to wartość `NULL`. W wierszu nr 9 funkcja zwolni pamięć przeznaczoną na pierwszy element, a w wierszu nr 10 zapisze we wskaźniku listy, wskazywanym przez `node`, adres węzła, który do tej pory był drugi, ale po usunięciu pierwszego stał się nowym pierwszym węzłem w liście. Proszę zwrócić uwagę, że funkcja `delete_node()` prawidłowo obsługuje również przypadek, kiedy usuwany element jest jedynym w liście.

Dwukierunkowa lista liniowa

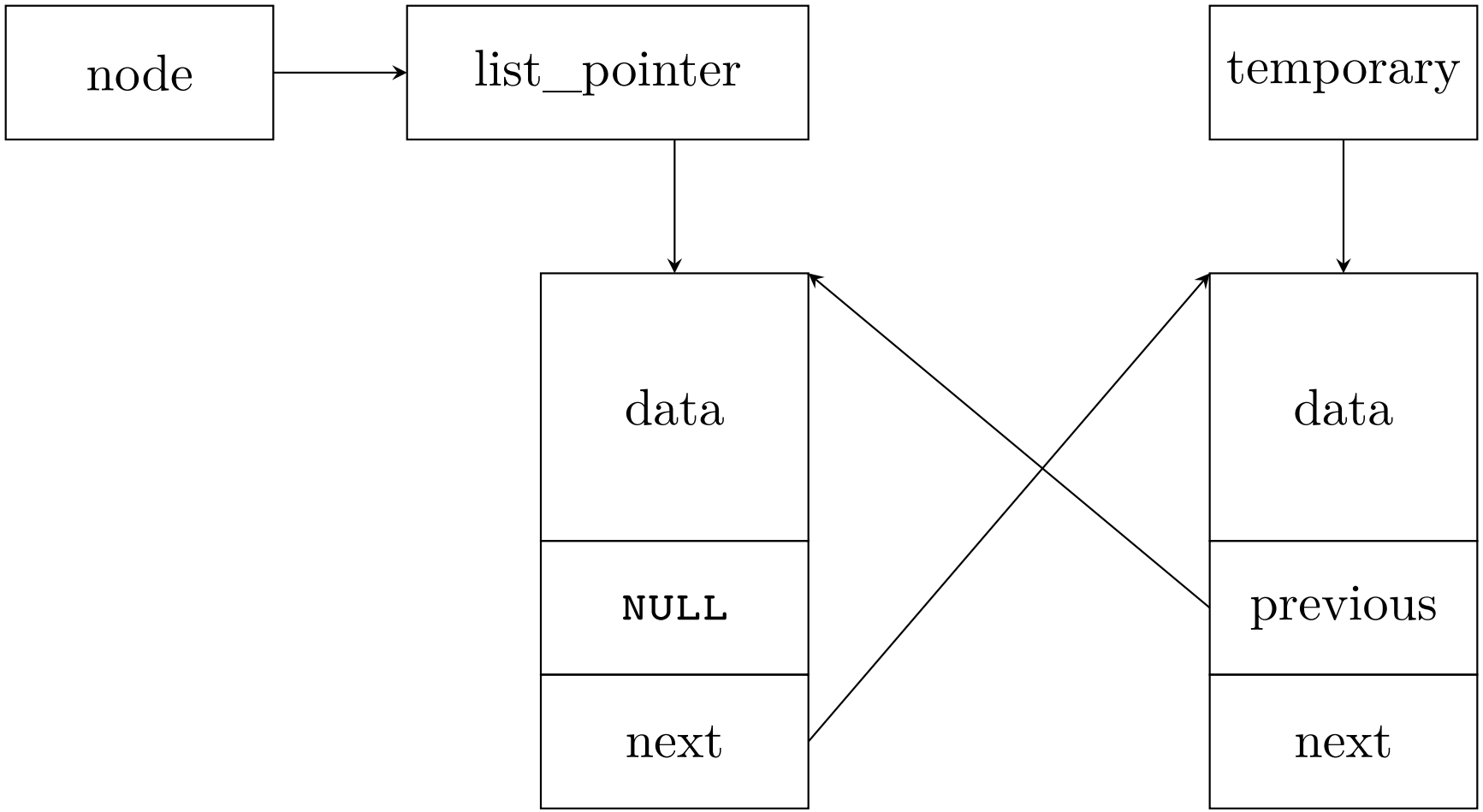
Usunięcie pierwszego węzła



Przed wykonaniem wiersza nr 6 funkcji `delete_node()`

Dwukierunkowa lista liniowa

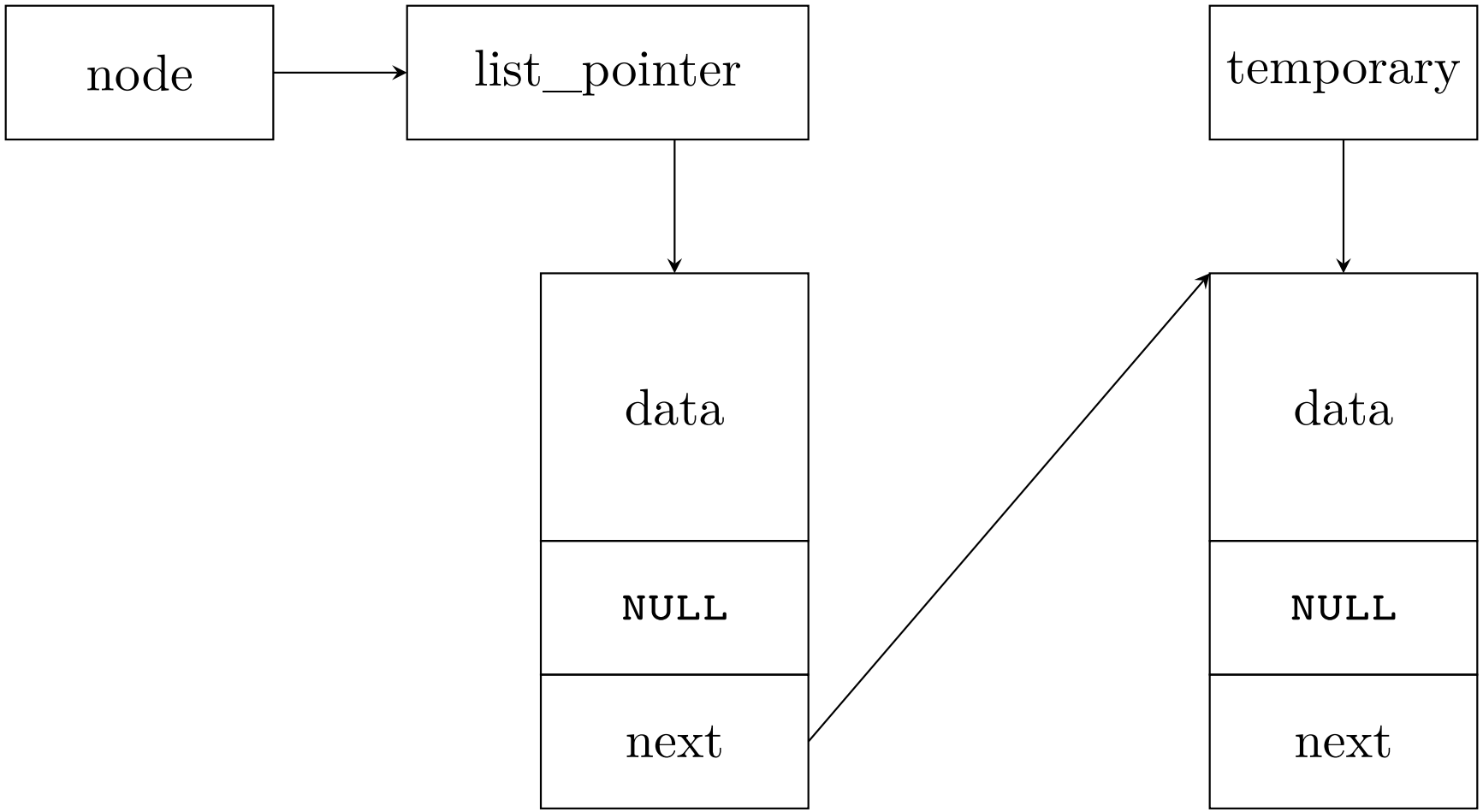
Usunięcie pierwszego węzła



Przed wykonaniem wiersza nr 8 funkcji `delete_node()`

Dwukierunkowa lista liniowa

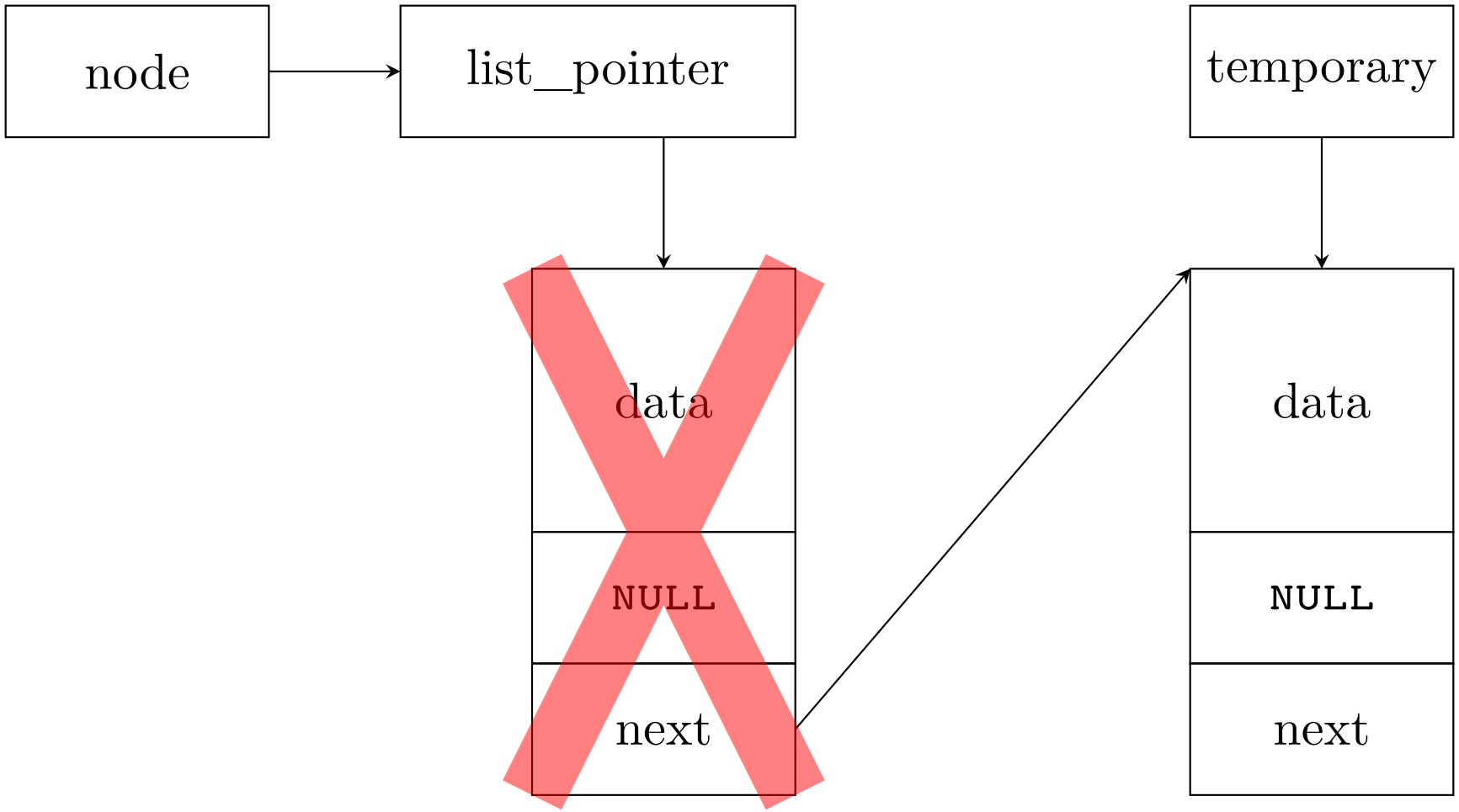
Usunięcie pierwszego węzła



Przed wykonaniem wiersza nr 9 funkcji `delete_node()`

Dwukierunkowa lista liniowa

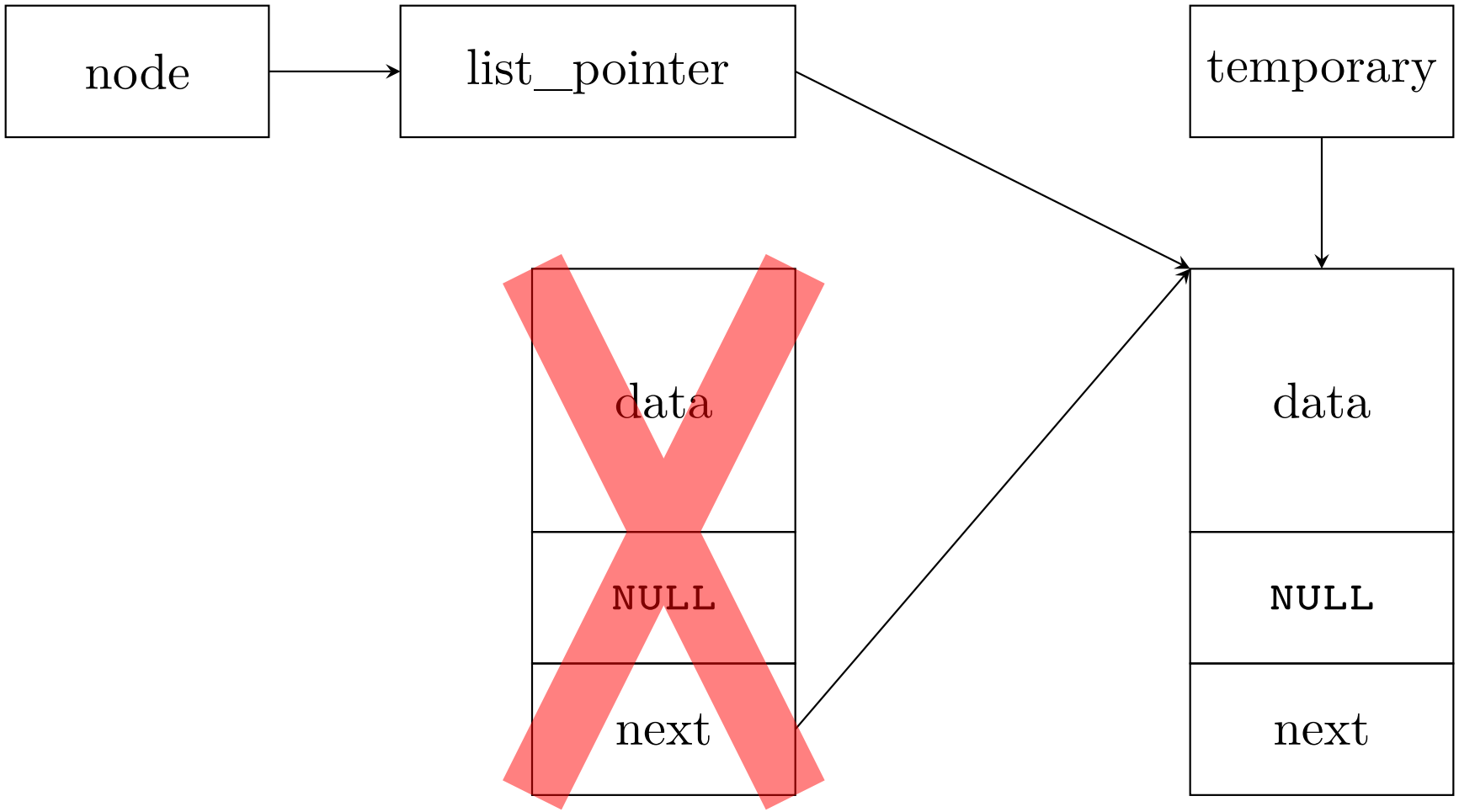
Usunięcie pierwszego węzła



Przed wykonaniem wiersza nr 10 funkcji `delete_node()`

Dwukierunkowa lista liniowa

Usunięcie pierwszego węzła



Po wykonaniu wiersza nr 10 funkcji `delete_node()`

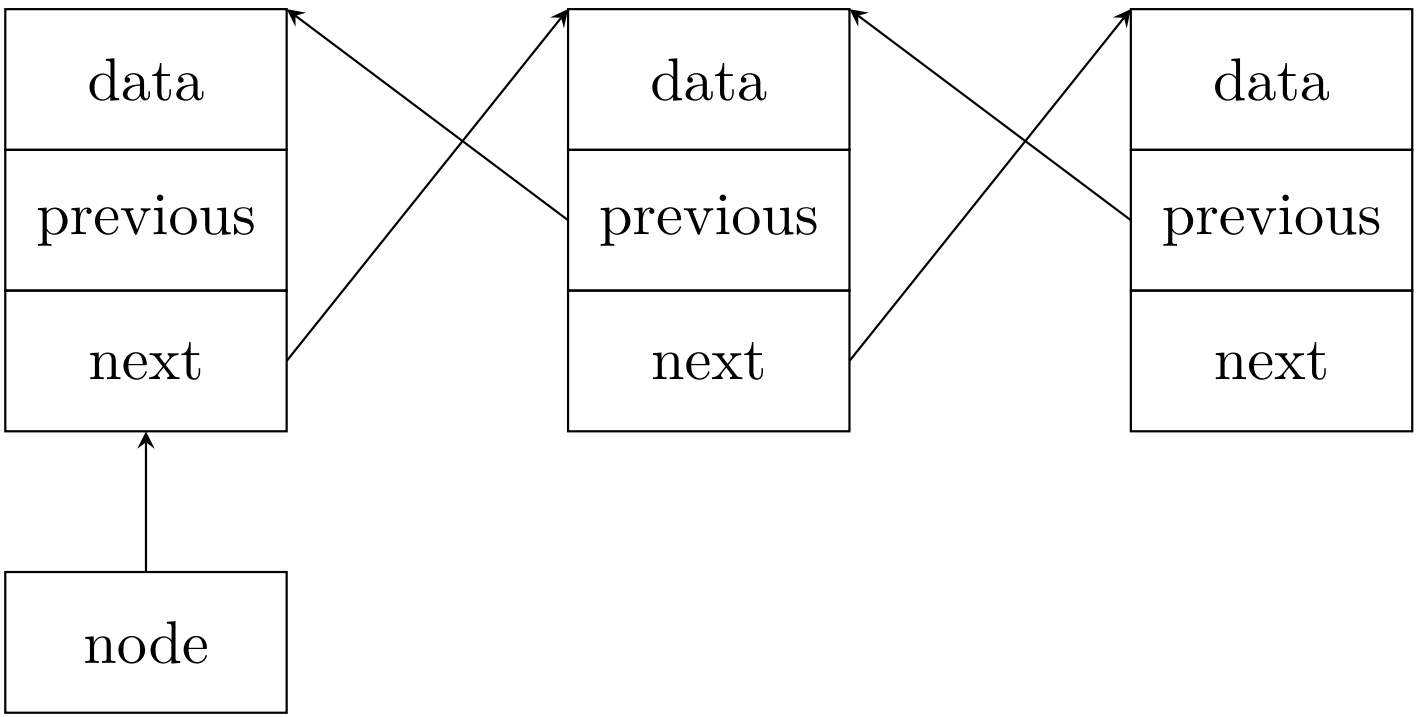
Dwukierunkowa lista liniowa

Usunięcie wewnętrznego węzła

Usunięcie węzła znajdującego się między dwoma innymi jest najbardziej skomplikowanym przypadkiem. W takiej sytuacji, po zakończeniu pętli `while`, parametr `node` wskazuje na pole `next`, które zawiera adres elementu do usunięcia. Funkcja `delete_node()` zapisuje w zmiennej `temporary` adres węzła, znajdującego się w liście za tym do usunięcia (wiersz nr 6), który pobiera z pola `next` tego ostatniego. Następnie, po sprawdzeniu, że ten następnik istnieje (wiersz nr 7), zapisuje w jego polu `previous` adres, który znajduje się w polu o tej samej nazwie, ale należącym do usuwanego węzła (wiersz nr 8). Po tym funkcja zwalnia pamięć przeznaczoną na usuwany węzeł (wiersz nr 9) i w polu `next` jego poprzednika zapisuje adres jego dotychczasowego następnika (wiersz nr 10), pobrany ze zmiennej `temporary`. Na tym kończy się ta operacja.

Dwukierunkowa lista liniowa

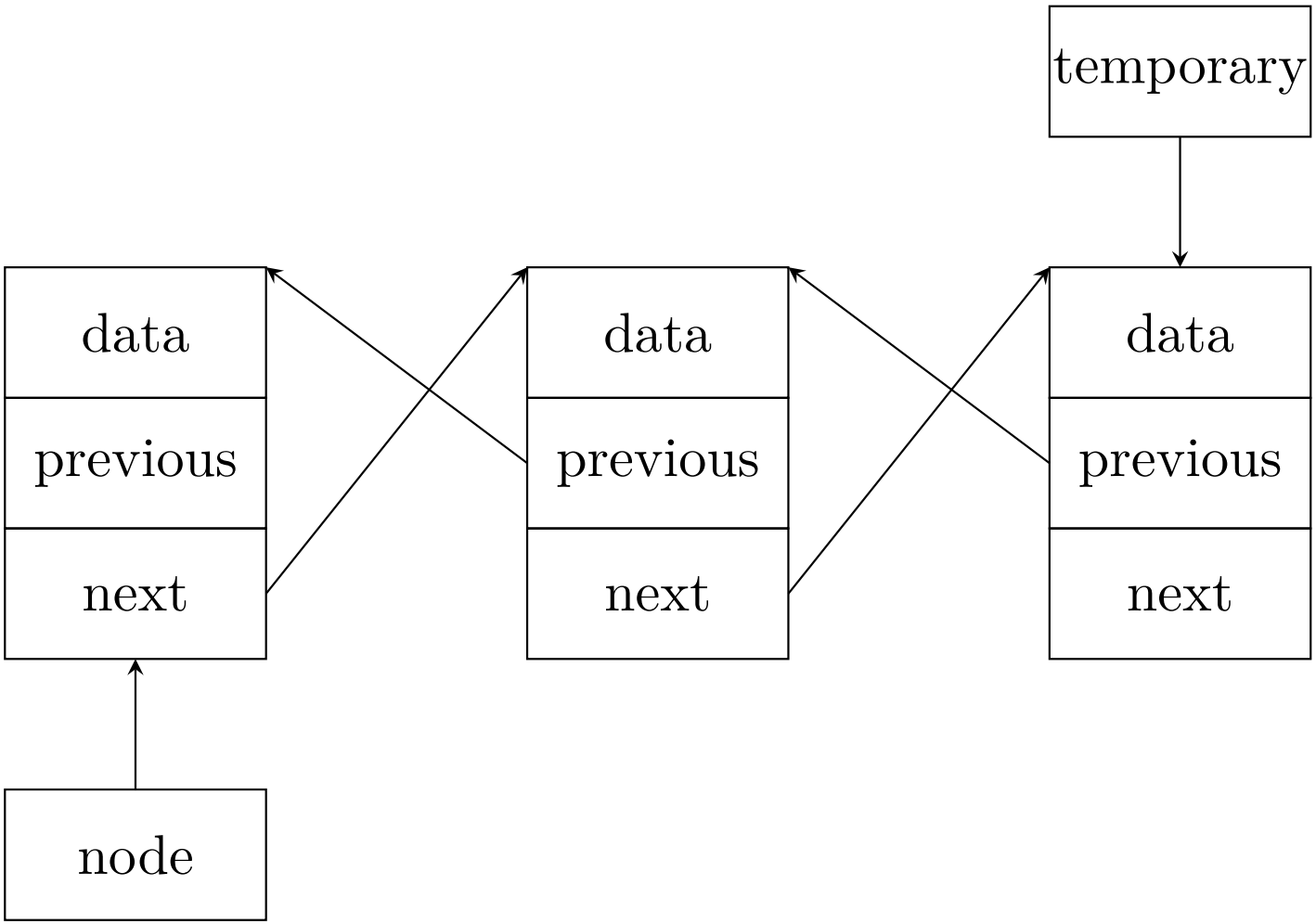
Usunięcie wewnętrznego węzła



Przed wykonaniem wiersza nr 6 funkcji `delete_node()`

Dwukierunkowa lista liniowa

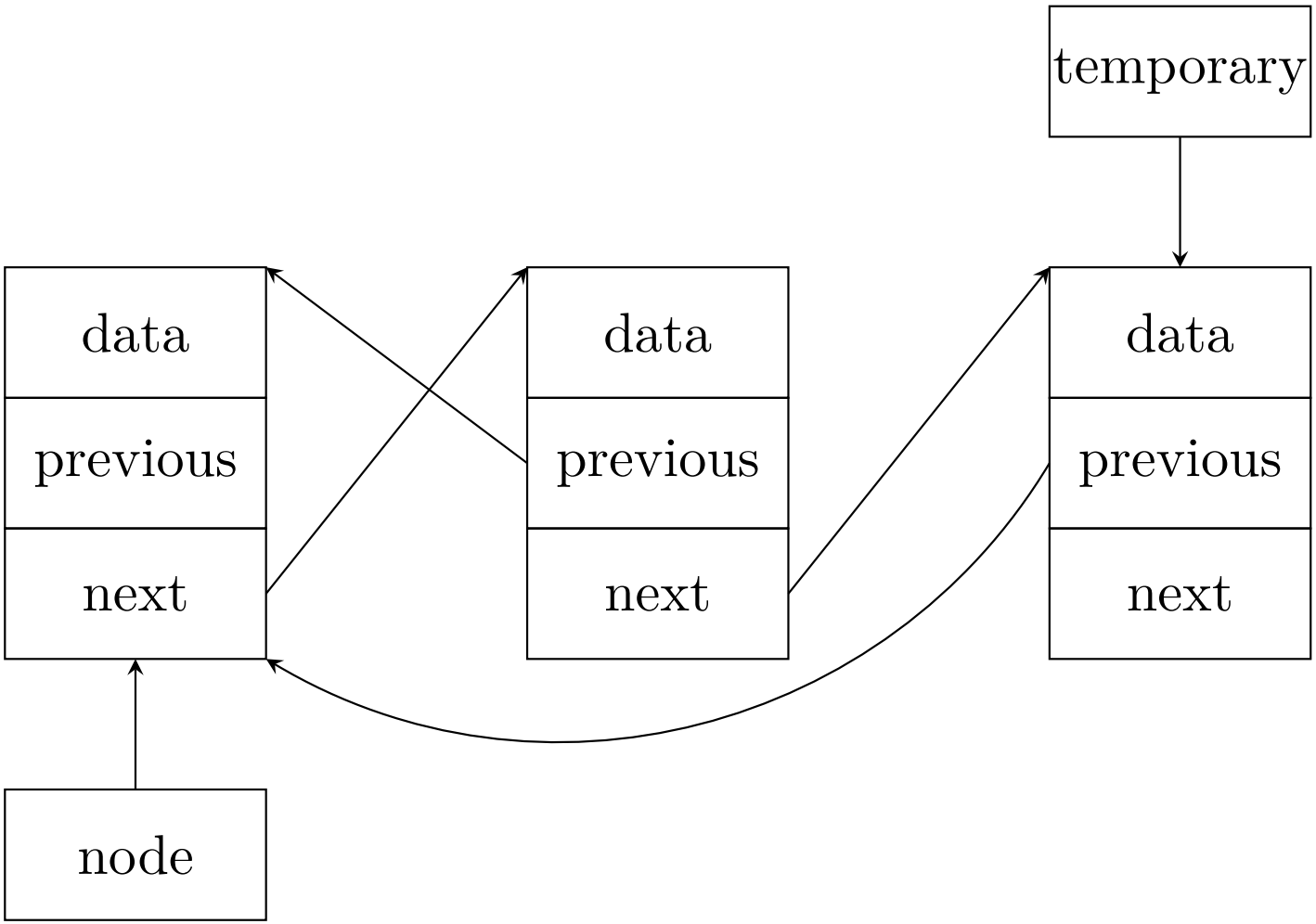
Usunięcie wewnętrznego węzła



Przed wykonaniem wiersza nr 8 funkcji `delete_node()`

Dwukierunkowa lista liniowa

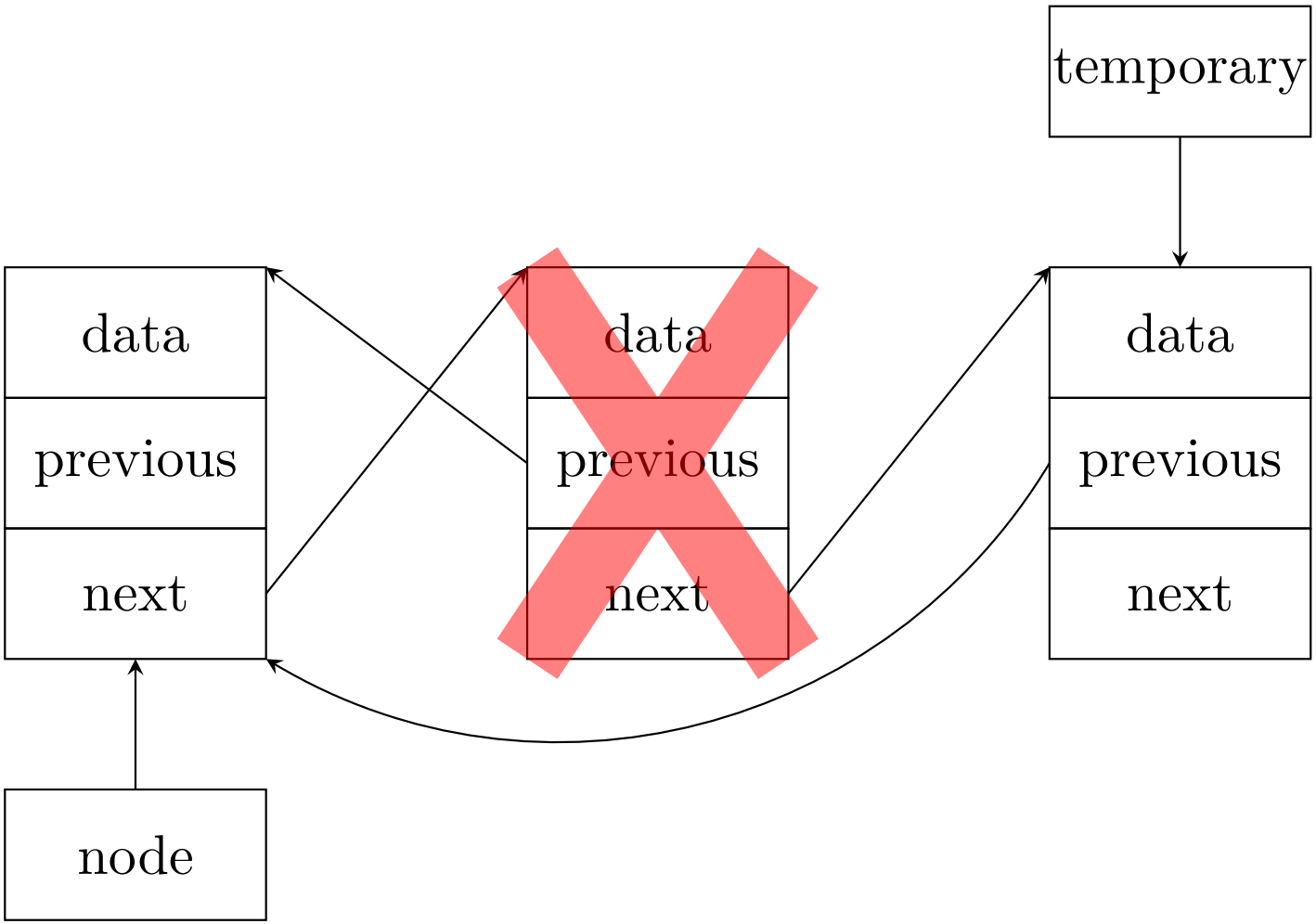
Usunięcie wewnętrznego węzła



Przed wykonaniem wiersza nr 9 funkcji `delete_node()`

Dwukierunkowa lista liniowa

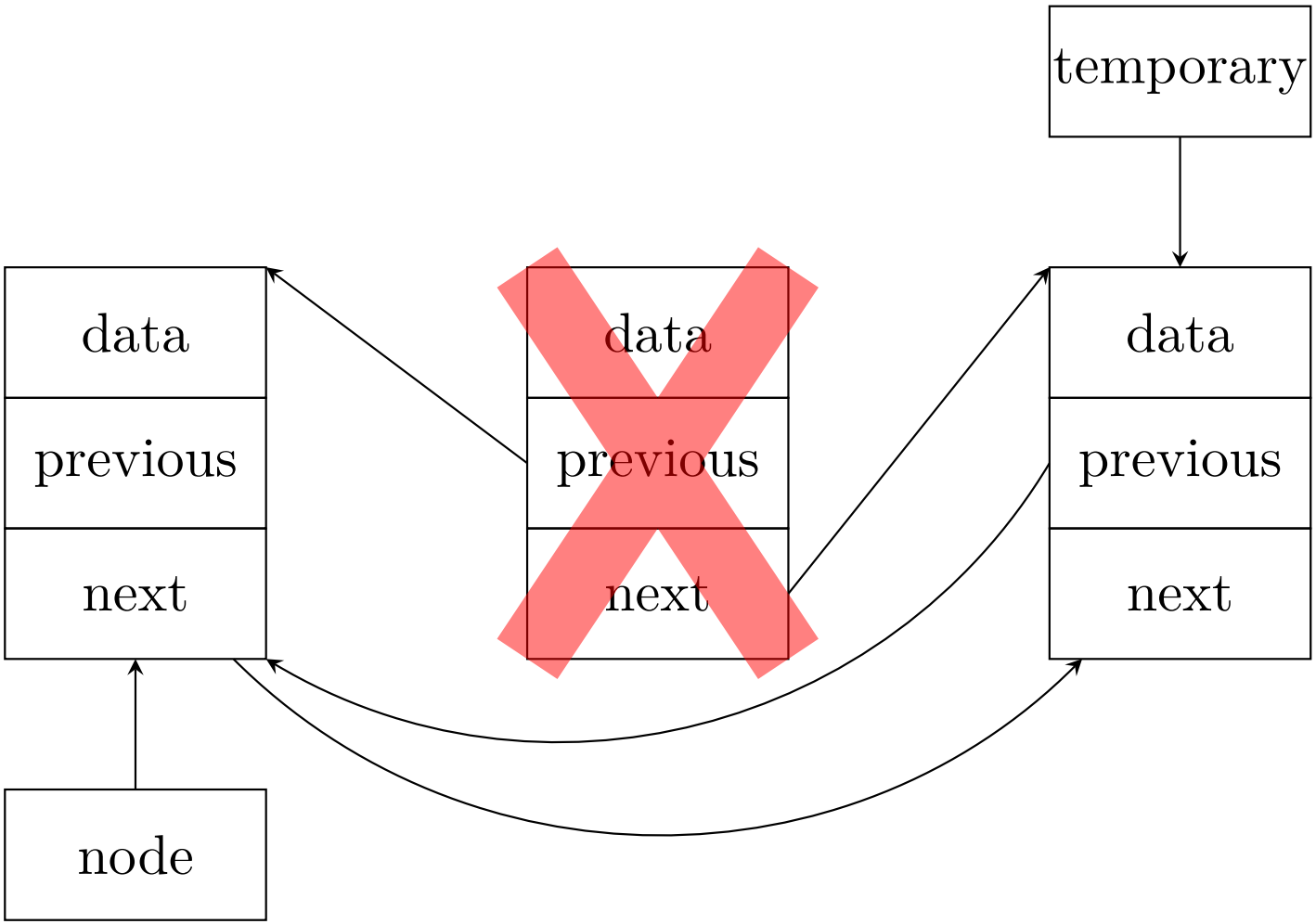
Usunięcie wewnętrznego węzła



Przed wykonaniem wiersza nr 10 funkcji `delete_node()`

Dwukierunkowa lista liniowa

Usunięcie wewnętrznego węzła



Po wykonaniu wiersza nr 10 funkcji `delete_node()`

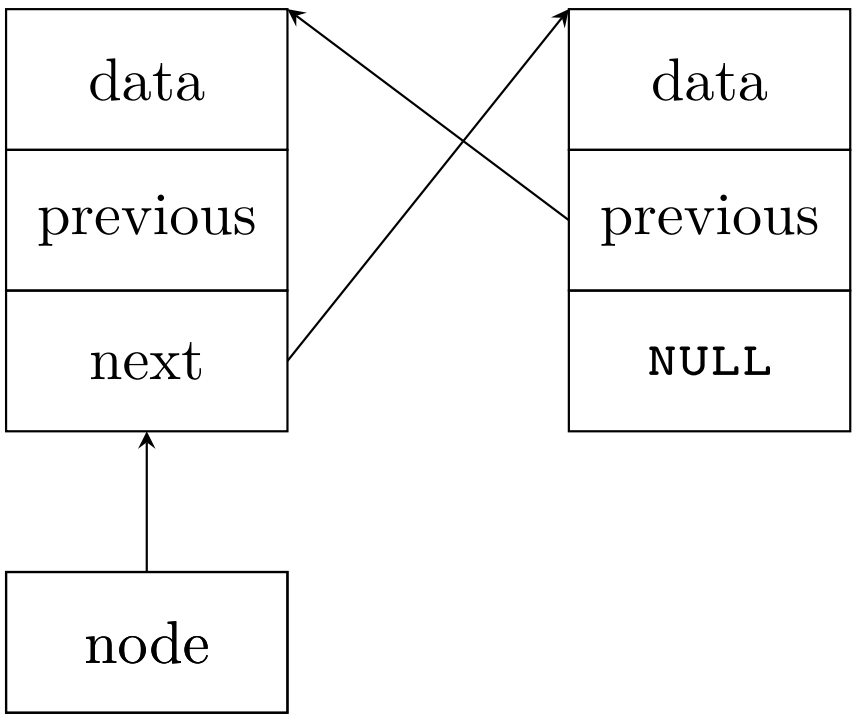
Dwukierunkowa lista liniowa

Usunięcie ostatniego węzła

Przebieg operacji usunięcia ostatniego węzła w liście jest w zasadzie taki sam, jak w przypadku jednokierunkowej listy liniowej, więc jej opis zostanie pominięty, ale następny slajd ilustruje jej przebieg.

Dwukierunkowa lista liniowa

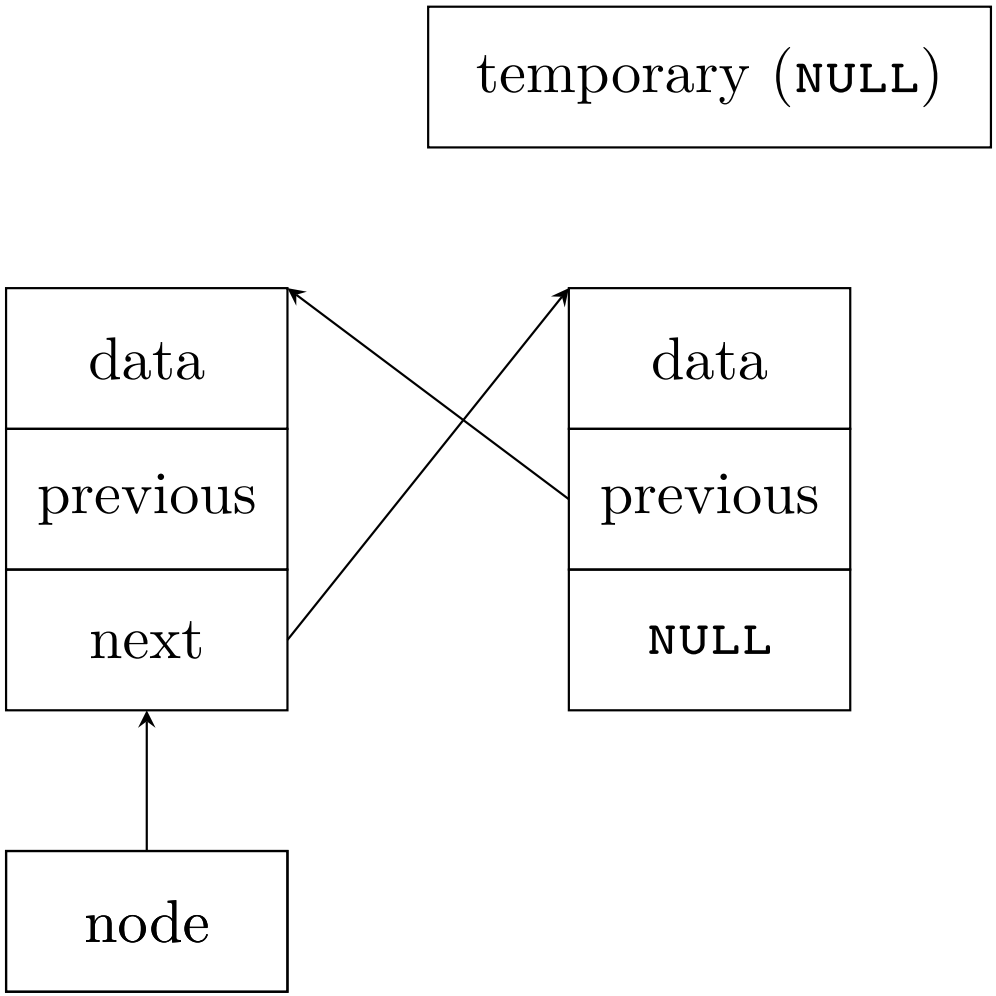
Usunięcie ostatniego węzła



Przed wykonaniem wiersza nr 6 funkcji `delete_node()`

Dwukierunkowa lista liniowa

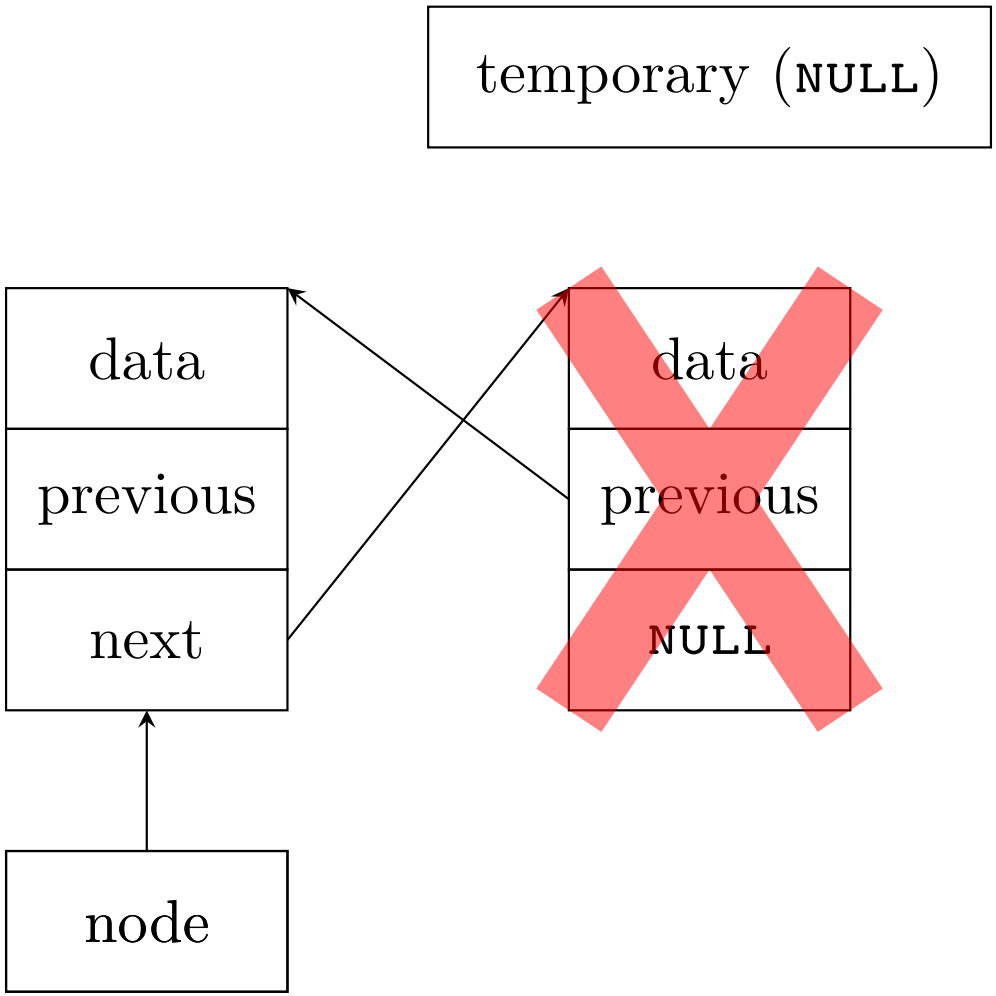
Usunięcie ostatniego węzła



Przed wykonaniem wiersza nr 9 funkcji `delete_node()`

Dwukierunkowa lista liniowa

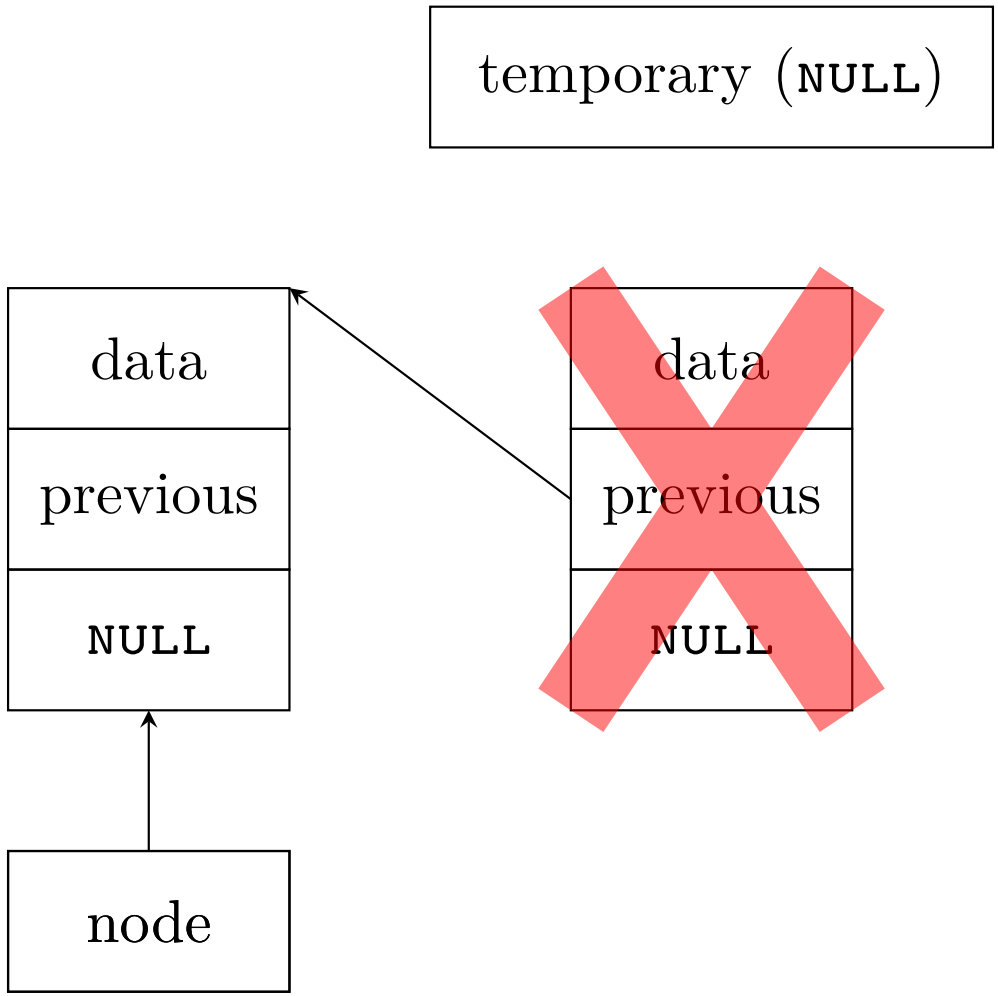
Usunięcie ostatniego węzła



Przed wykonaniem wiersza nr 10 funkcji `delete_node()`

Dwukierunkowa lista liniowa

Usunięcie ostatniego węzła



Po wykonaniu wiersza nr 10 funkcji `delete_node()`

Dwukierunkowa lista liniowa

Funkcja `print_list()`

```
1  void print_list(struct list_node *node)
2  {
3      while(node) {
4          printf("%d ", node->data);
5          node = node->next;
6      }
7      puts("");
8  }
```

Dwukierunkowa lista liniowa

Funkcja `print_list()`

Funkcja `print_list()` jest tak samo zdefiniowana jak dla jednokierunkowej listy liniowej, więc jej opis jest pominięty.

Dwukierunkowa lista liniowa

Funkcja `print_backwards()`

```
1  void print_backwards(struct list_node *node)
2  {
3      while(node && node->next)
4          node = node->next;
5      while (node) {
6          printf("%d ", node->data);
7          node = node->previous;
8      }
9      puts("");
10 }
```

Dwukierunkowa lista liniowa

Funkcja `print_backwards()`

Zgodnie ze swoją nazwą, dwukierunkowa lista linowa ma strukturę, która pozwala iterować po niej w dwóch kierunkach: od pierwszego węzła do ostatniego i od ostatniego do pierwszego. Korzysta z tego funkcja `print_backwards()`. Pierwsza z zawartych w niej pętli `while` (wiersze 3–4) wykonywana jest tak długo, aż wskaźnik `node` będzie wskazywał na ostatni węzeł w liście (ten, którego pole `next` zawiera wartość `NULL`). Następnie druga pętla `while` (wiersze 5–8) iteruje po tej liście, aż wartość `node` nie będzie równa `NULL`. Wewnątrz tej pętli wypisywana jest liczba znajdująca się w węźle bieżąco wskazywanym przez `node` (wiersz nr 6), a następnie adres we wskaźniku `node` jest zastępowany tym przechowywanym w polu `previous` węzła bieżąco przez niego wskazywanego (wiersz nr 7). Innymi słowy ten wskaźnik jest „przestawiany” na poprzednika tego węzła. Dzięki temu funkcja `print_backwards()` wypisze liczby z listy w odwrotnej kolejności.

Dwukierunkowa lista liniowa

Funkcja `remove_list()`

```
1  void remove_list(struct list_node **node)
2  {
3      while(*node) {
4          struct list_node *temporary = (*node)->next;
5          free(*node);
6          *node = temporary;
7      }
8  }
```

Dwukierunkowa lista liniowa

Funkcja `remove_list()`

Funkcja `remove_list()` jest zaimplementowana tak samo, jak w przypadku jednokierunkowej listy liniowej, więc jej opis zostanie tutaj pominięty.

Dwukierunkowa lista liniowa

Funkcja `main()`, część 1

```
1  int main(void)
2  {
3      for(int i=1; i<5; i++)
4          if(add_node(&list_pointer,i)==-1)
5              fprintf(stderr,"Błąd dodawania elementu o
   ↪  wartości %d do listy!\n",i);
6      for(int i=6; i<10; i++)
7          if(add_node(&list_pointer,i)==-1)
8              fprintf(stderr,"Błąd dodawania elementu o
   ↪  wartości %d do listy!\n",i);
9      print_list(list_pointer);
10     print_backwards(list_pointer);
```

Dwukierunkowa lista liniowa

Funkcja `main()`, część 2

```
1       if(add_node(&list_pointer,0)==-1)
2           fprintf(stderr,"Błąd dodawania elementu o
    ↪   wartości %d do listy!\n",0);
3       print_list(list_pointer);
4       print_backwards(list_pointer);
5       if(add_node(&list_pointer,5)==-1)
6           fprintf(stderr,"Błąd dodawania elementu o
    ↪   wartości %d do listy!\n",5);
7       print_list(list_pointer);
8       print_backwards(list_pointer);
9       if(add_node(&list_pointer,7)==-1)
10          fprintf(stderr,"Błąd dodawania elementu o
    ↪   wartości %d do listy!\n",7);
11      print_list(list_pointer);
12      print_backwards(list_pointer);
```

Dwukierunkowa lista liniowa

Funkcja `main()`, część 3

```
1       if(add_node(&list_pointer,10)==-1)
2           fprintf(stderr,"Błąd dodawania elementu o
    ↪   wartości %d do listy!\n",10);
3       print_list(list_pointer);
4       print_backwards(list_pointer);
5       puts("");
6       delete_node(&list_pointer,0);
7       print_list(list_pointer);
8       print_backwards(list_pointer);
9       delete_node(&list_pointer,1);
10      print_list(list_pointer);
11      print_backwards(list_pointer);
12      delete_node(&list_pointer,1);
13      print_list(list_pointer);
14      print_backwards(list_pointer);
```

Dwukierunkowa lista liniowa

Funkcja `main()`, część 4

```
1       delete_node(&list_pointer,4);
2       print_list(list_pointer);
3       print_backwards(list_pointer);
4       delete_node(&list_pointer,7);
5       print_list(list_pointer);
6       print_backwards(list_pointer);
7       delete_node(&list_pointer,10);
8       print_list(list_pointer);
9       print_backwards(list_pointer);
10      remove_list(&list_pointer);
11      return 0;
12  }
```

Dwukierunkowa lista liniowa

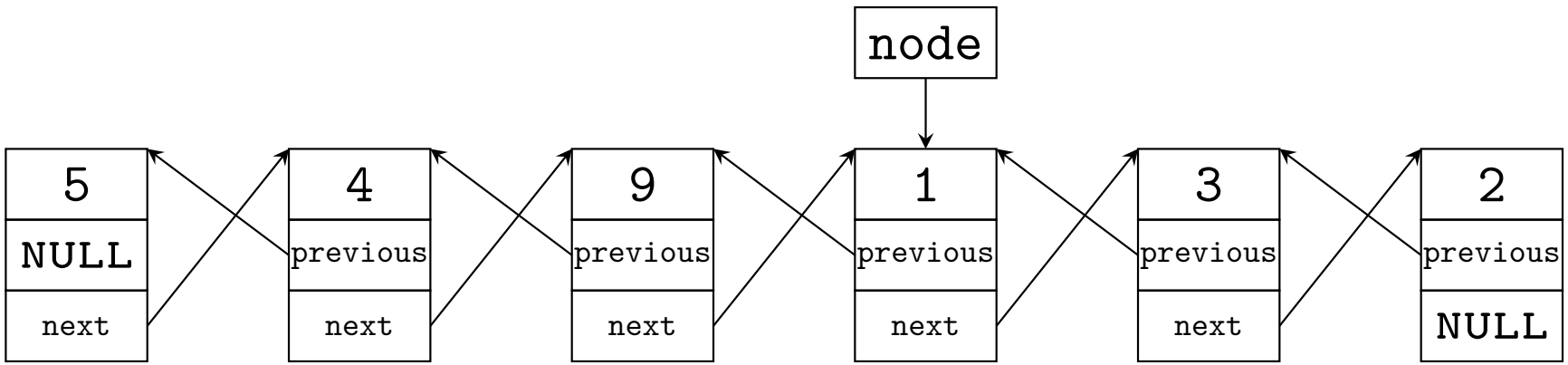
Funkcja `main()`

Jedyna różnica w kodzie funkcji `main()` w stosunku do jej odpowiedniczki w programie demonstrującym użycie jednokierunkowej listy liniowej polega na tym, że po każdym wywoływaniu funkcji `print_list()` uruchamiana jest także funkcja `print_backwards()`, która wypisuje znajdujące się w liście liczby w odwrotnej kolejności, aby tym samym sprawdzić, że lista jest spójna.

Podsumowanie

W zaprezentowanych funkcjach, związanych z dodawaniem i usuwaniem węzłów, można zauważyć skomplikowane wyrażenia budowane przy użyciu wskaźników. Następne slajdy pokazują równie skomplikowane przypadki takich wyrażeń. Odnoszą się one do przedstawionej w górnej części slajdów listy. Wskaźnik `node`, który występuje na początku każdego takiego wyrażenia również jest zamieszczony na podanej ilustracji. Proszę spróbować ustalić wartości tych wyrażeń.

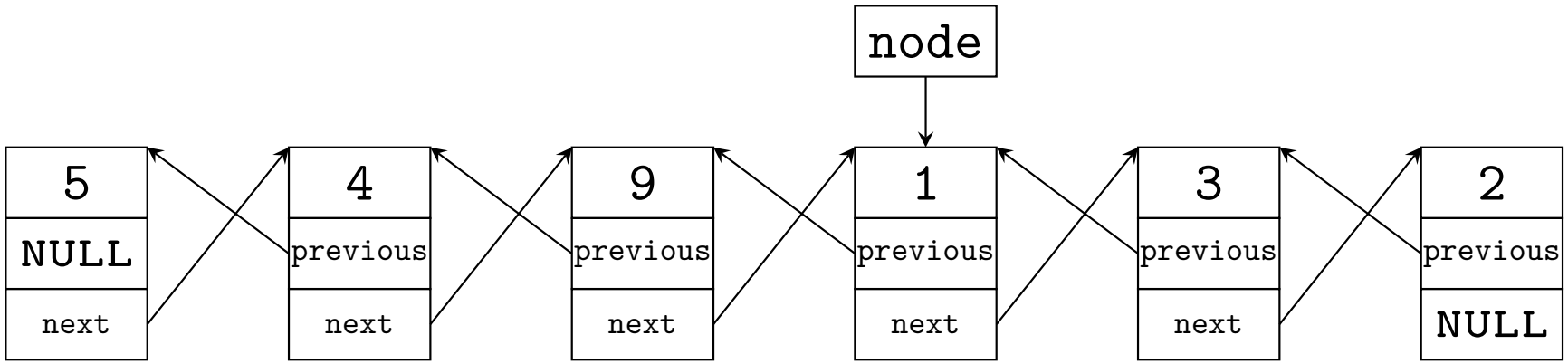
Podsumowanie



Wyrażenie nr 1

```
node->next->next->data
```

Podsumowanie



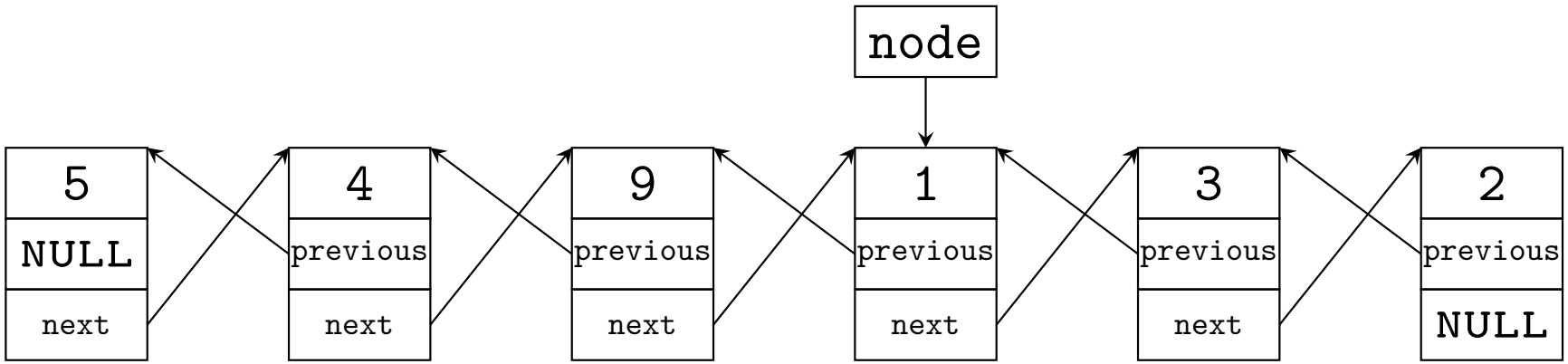
Wyrażenie nr 1

```
node->next->next->data
```

Odpowiedź nr 1

2

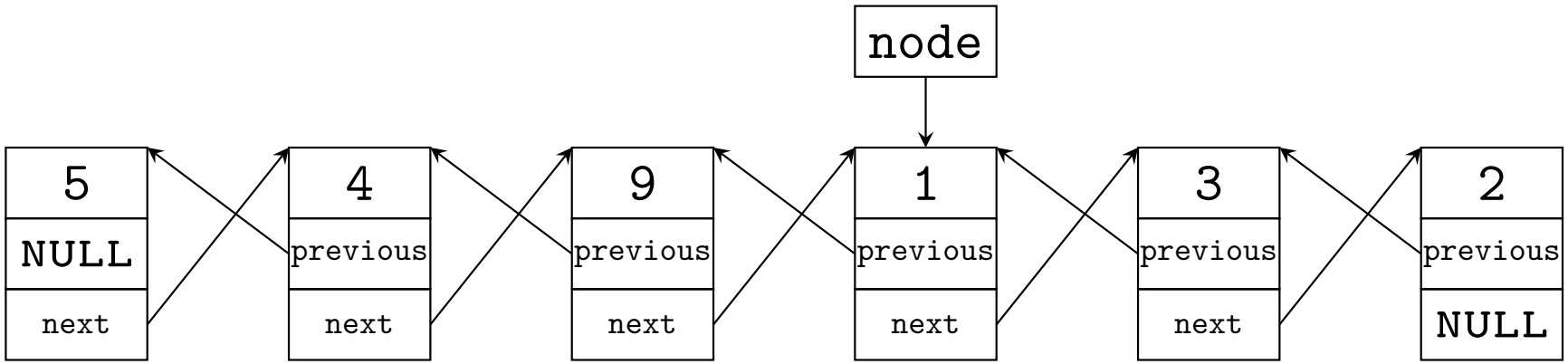
Podsumowanie



Wyrażenie nr 2

```
node->previous->previous->previous->data
```

Podsumowanie



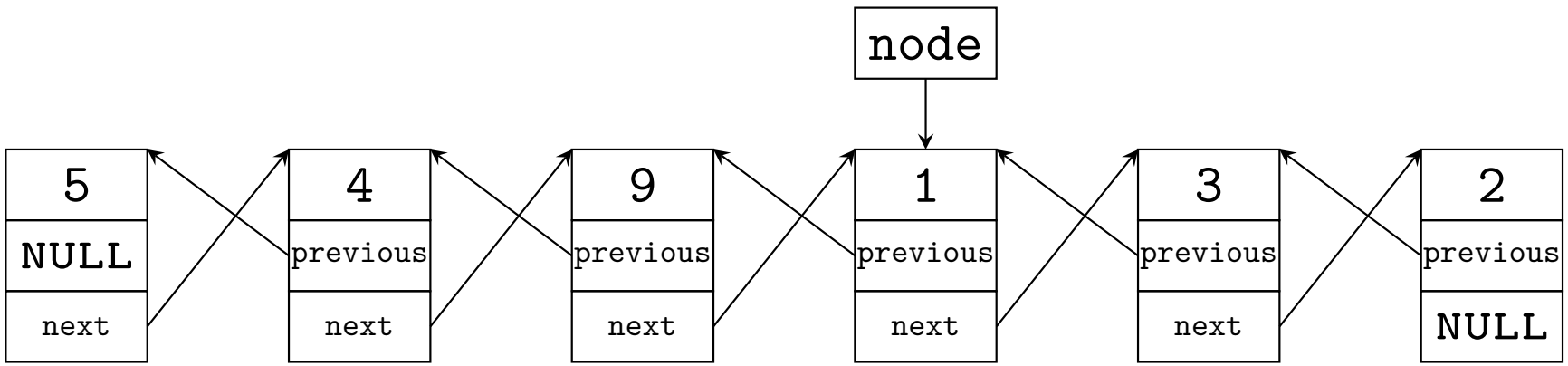
Wyrażenie nr 2

```
node->previous->previous->previous->data
```

Odpowiedź nr 2

5

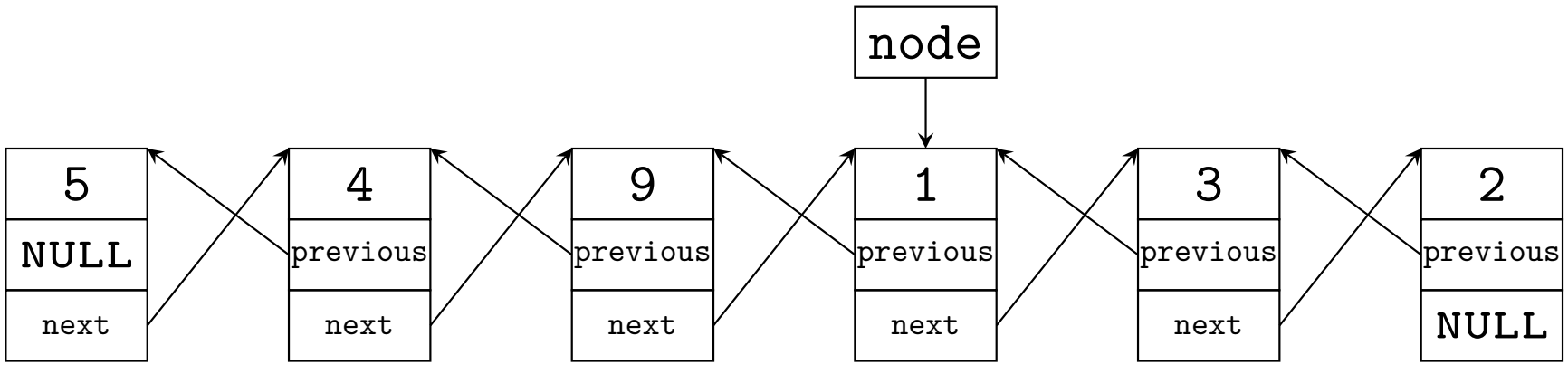
Podsumowanie



Wyrażenie nr 3

```
node->next->next->previous->previous->previous->previous->data
```

Podsumowanie



Wyrażenie nr 3

```
node->next->next->previous->previous->previous->previous->data
```

Odpowiedź nr 3

4

Podsumowanie

Reguła odczytu tych wyrażeń jest dosyć prosta — należy podążać za wskaźnikami. Warto jednak zwrócić uwagę na ostanie wyrażenie, gdzie wymieszane jest użycie wskaźników `next` i `previous`. Te wskaźniki wzajemnie „znoszą się". Zatem to wyrażenie można zapisać w skróconej formie jako: `node->previous->previous->data`.
Wniosek jaki nasuwa się po zapoznaniu się z tak rozbudowanymi wyrażeniami wskaźnikowymi jest następujący: Należy wiedzieć jak czytać takie wyrażenia i jak one działają, ale powinno unikać się stosowania ich w programach ☺.

Pytania

?

KONIEC

Dziękuję Państwu za uwagę!