1 Pointers

Exercise 1.1. Make declarations of the following entities:

(a) A pointer to an integer

(b) A pointer to a string

(c) A pointer to a constant integer

(d) A constant pointer to a integer

(e) A constant pointer to a constant integer

(f) A pointer to an integer pointer

(g) A pointer to a constant integer pointer

Exercise 1.2. Use words to describe the following declarations:

(a) `int* const*`

(b) `int const*`

(c) `int*`

(d) `std::string*`

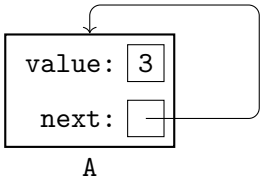
(e) `int**`

(f) `int const* const`

(g) `int* const`

Hint: All of these declarations correspond to a description in exercise 1.1.

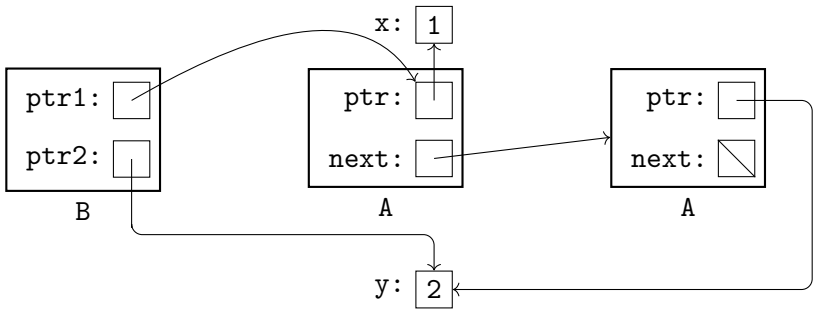
Exercise 1.3. Write code that implements the following diagram:



Exercise 1.4. Draw a diagram that represents the following pointer structure:

```
1  struct A
2  {
3    int* ptr;
4    int value;
5  };
6
7  struct B
8  {
9    int* ptr;
10   A* other;
11 };
12
13 int main()
14 {
15   int x { 3 };
16   A a { &x, 4 };
17   B b { &x, &a };
18 }
```

Exercise 1.5. Write code that implements the following diagram:



Note: `ptr1` in `B` does **not** point at an `A`, it points to an `int*`.

Exercise 1.6. Suppose the `B` object in exercise 1.5 is called `b`. Write a single line of code that uses `b` and returns the value of `x`. Note, you are not allowed to directly refer to `x`.

Exercise 1.7. Suppose that the left-most `A` in exercise 1.5 is called `a`. Write a single line of code that uses `a` and returns the value of `y`. Note, you are not allowed to directly refer to `y`.

2 Dynamic memory

Exercise 2.8. Create a program that:

1. dynamically allocates a `std::string` on the heap with the value `"hello"`.
2. Prints the value of the allocated string.
3. Deallocates the string.

Exercise 2.9. Write a program that places the value `3` on the stack and the value `5` on the heap. Make sure that no memory leaks occur.

Exercise 2.10. Improve the following code by removing unnecessary allocations and ensuring that no memory leaks occur:

```
1  struct Node
2  {
3    int value { };
4    Node* next { };
5  };
6
7  Node* insert_after(Node* node, int value)
8  {
9    node->next = new Node { value, node->next };
10 }
11
12 int main()
13 {
14   int* x { new int { 1 } };
15
16   Node* head { new Node { *x } };
17
18   Node* second { insert_after(head, 2) };
19   Node* third { insert_after(second, 3) };
20
21   insert_after(third, 4);
22
23   delete x;
24 }
```

Exercise 2.11. Fix the potential segmentation faults in this code (and potential bugs too):

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

struct Node
{
  int value { };
  Node* next { };
};

std::string& get_string()
{
  // create a string containing 1024 'a' characters
  std::string my_string(1024, 'a');
  return my_string;
}

int main()
{
  std::cout << get_string() << std::endl;

  Node* head { new Node { 1, new Node { 2 } } };

  Node* current { head };
  while (current->value != 3)
  {
    current = current->next;
  }

  if (current->value == 3)
  {
    std::cout << "Found 3!" << std::endl;
  }
  else
  {
    std::cout << "Didn't find 3..." << std::endl;
  }

  delete head->next;
  delete head;
}
```