```markdown
# 8-pointer

## `radius.c`

### On Variables
- type, value, address
- `&`: address-of operator
- printf the address (`%p`)
- `lvalue`, `rvalue`???

### On Pointers
- `int *` syntax
- int * vs. double * (type cast???)
- refs to itself (int ** vs. int *)
- Visualization

- `scanf`: how does it work???

## `Swap` (`selection-sort.c`)
- `WrongSwap`
- `Swap`
- Visualization

## Pointers and Arrays (`selection-sort.c`)

- `()`: function call operator
  - `SelectionSort(numbers, LEN)`
- `int arr[]` vs. `(int *arr)`
- `numbers[i]` vs. `*(numbers + i)`
  - pointers arithmetic (in arrays!!!)
  - `pointer + int`, `pointer - int`, `pointer - pointer`
- `&numbers[i]` vs. `numbers + i`

## Array Name (`selection-sort.c`)
- `int arr[] = {1, 2, 3};`
- `arr++`
- `numbers++`

## Dynamic Memory Management (`selection-sort.c`)

- VLA
- `malloc.h` vs. `stdlib.h`
- `malloc`
  - `void *`
    - `int *`
    - `sizeof(*numbers)`
  - size = 0: implementation-defined
  - `unsigned long long`
- `NULL`
  - `(void *) 0`
- `free`
  - memory leak (heap)
  - **undefined behaviors**
```

```markdown
    - double `free`
    - `free` non-`malloc`
      - `numbers = NULL`
    - dereference `free`d memory

## `const` in `Print` (`selection-sort.c`)
```

```c
/**
 * file: radius.c
 *
 * Created by hengxin on 11/24/23.
 */

#include <stdio.h>
#include <stdlib.h>

#define PI 3.14

int main() {
  /********** On radius **********/
  int radius = 100;

  printf("radius = %d\n", radius);

  // every variable has an address
  // &: address-of operator ("□□□"□□□)
  printf("&radius = %p\n", &radius);
  // we have already used the address of a variable before
  // scanf("%d", &radius);

  // radius as a left value; refer to its address (the storage space)
  radius = 200;
  // radius as a right value; refer to its value
  double circumference = 2 * PI * radius;
  printf("circumference = %f\n", circumference);
  /********** On radius **********/

  /********** On ptr_radius1 **********/
  // ptr_radius1 is a variable of type "pointer to int"
  int *ptr_radius1 = &radius;
  // ptr_radius1 is a variable: has its value
  printf("ptr_radius1 = %p\n", ptr_radius1);
  // ptr_radius1 is a variable: has its address
  printf("The address of ptr_radius1 is %p\n", &ptr_radius1);
  /********** On ptr_radius1 **********/

  /********** On *ptr_radius1 **********/
  // IMPORTANT:
  // *ptr_radius1: behaves just like radius
  // type: int; value: the value of radius; address: the address of radius
  // *: indirection/dereference operator ("□□□□"/"□□□"□□□)
  printf("radius = %d\n", *ptr_radius1);
  // *ptr_radius1 as a right value
  circumference = 2 * 3.14 * (*ptr_radius1);
  // take the address of *ptr_radius1
  // &*ptr_radius1 is the same as ptr_radius1
  printf("The address of *ptr_radius1 is %p\n", &*ptr_radius1);
  // *ptr_radius1 as a left value
  *ptr_radius1 = 100;
```

```c
printf("radius = %d\n", *ptr_radius1);
/********** On *ptr_radius1 **********/

/********** Begin: On ptr_radius1 as lvalue and rvalue **********/
// ptr_radius1 as a left value
int radius2 = 200;
int *ptr_radius2 = &radius2;

ptr_radius1 = ptr_radius2;
printf("radius = %d\n", *ptr_radius1);

// ptr_radius1 as a right value
ptr_radius2 = ptr_radius1;
printf("radius = %d\n", *ptr_radius2);
/********** Begin: On ptr_radius1 as lvalue and rvalue **********/

/********** On array names **********/
int numbers[5] = {0};
// vs. numbers[2] = {2};
// numbers++;
// numbers = &radius;
int *ptr_array = numbers;
ptr_array++;
/********** On array names **********/

/********** On malloc/free **********/
// undefined behavior
// free(numbers);
/********** On malloc/free **********/

/********** On const **********/
// const int * and int const *
// You cannot modify the value pointed to by ptr_radius3
// through the pointer (without casting the constness away).
const int *ptr_radius3 = &radius;
// *ptr_radius is read-only
// *ptr_radius3 = 300;
// You are allowed to do this, but you should not do it!
int *ptr_radius4 = ptr_radius3;
*ptr_radius4 = 400;
printf("radius = %d\n", radius);

// int * const
int *const ptr_radius5 = &radius;
// ptr_radius5 = ptr_radius3;
*ptr_radius5 = 500;
printf("radius = %d\n", radius);

// const int * const
const int *const ptr_radius6 = &radius;
// ptr_radius6 = ptr_radius3;
// *ptr_radius6 = 600;
/********** On const **********/
```

```c

  return 0;
}
```

```c
//
// Created by hfwei on 2023/10/12.
// Visualization of Swap: https://pythontutor.com/visualize.html#code=//%0A//%20Created%20by%20hfwei%20on%202023/10/12.%0A//%0A%0A%23include%20%3Cstdio.h%3E%0A%0A%23define%20LEN%205%0A%0Avoid%20SelectionSort%28int%20arr%5B%5D,%20int%20len%29%3B%0Avoid%20WrongSwap%28int%20left,%20int%20right%29%3B%0Avoid%20Swap%28int%20*left,%20int%20*right%29%3B%0Aint%20GetMinIndex%28const%20int%20arr%5B%5D,%20int%20begin,%20int%20end%29%3B%0Avoid%20Print%28const%20int%20arr%5B%5D,%20int%20len%29%3B%0A%0Aint%20main%28void%29%20%7B%0A%20%20int%20numbers%5BLEN%5D%20%3D%20%7B15,%2078,%2023,%208,%2050%7D%3B%0A%0A%20%20Print%28numbers,%20LEN%29%3B%0A%20%20SelectionSort%28numbers,%20LEN%29%3B%0A%20%20Print%28numbers,%20LEN%29%3B%0A%0A%20%20return%200%3B%0A%7D%0A%0A//%20arr%3A%20the%20%28copy%20of%20the%29%20address%20of%20the%20first%20element%20of%20the%20%60numbers%60%20array%0Avoid%20SelectionSort%28int%20arr%5B%5D,%20int%20len%29%20%7B%0A%20%20for%20%28int%20i%20%3D%200%3B%20i%20%3C%20len%3B%20i%2B%2B%29%20%7B%0A%20%20%20%20int%20min_index%20%3D%20GetMinIndex%28arr,%20i,%20len%29%3B%0A%0A%20%20%20%20//%20ERROR%3A%20WrongSwap%28arr%5Bi%5D,%20arr%5Bmin_index%5D%29%3B%0A%20%20%20%20int%20temp%20%3D%20arr%5Bi%5D%3B%0A%20%20%20%20arr%5Bi%5D%20%3D%20arr%5Bmin_index%5D%3B%0A%20%20%20%20arr%5Bmin_index%5D%20%3D%20temp%3B%0A%20%20%7D%0A%7D%0A%0Aint%20GetMinIndex%28const%20int%20arr%5B%5D,%20int%20begin,%20int%20end%29%20%7B%0A%20%20int%20min%20%3D%20arr%5Bbegin%5D%3B%0A%20%20int%20min_index%20%3D%20begin%3B%0A%0A%20%20for%20%28int%20i%20%3D%20begin%20%2B%201%3B%20i%20%3C%20end%3B%20%2B%2Bi%29%20%7B%0A%20%20%20%20if%20%28arr%5Bi%5D%20%3C%20min%29%20%7B%0A%20%20%20%20%20%20min%20%3D%20arr%5Bi%5D%3B%0A%20%20%20%20%20%20min_index%20%3D%20i%3B%0A%20%20%20%20%7D%0A%20%20%7D%0A%0A%20%20return%20min_index%3B%0A%7D%0A%0Avoid%20WrongSwap%28int%20left,%20int%20right%29%20%7B%0A%20%20int%20temp%20%3D%20left%3B%0A%20%20left%20%3D%20right%3B%0A%20%20right%20%3D%20temp%3B%0A%7D%0A%0Avoid%20Swap%28int%20*left,%20int%20*right%29%20%7B%0A%20%20int%20temp%20%3D%20*left%3B%0A%20%20*left%20%3D%20*right%3B%0A%20%20*right%20%3D%20temp%3B%0A%7D%0A%0Avoid%20Print%28const%20int%20arr%5B%5D,%20int%20len%29%20%7B%0A%20%20printf%28%22%5Cn%22%29%3B%0A%20%20for%20%28int%20i%20%3D%200%3B%20i%20%3C%20len%3B%20i%2B%2B%29%20%7B%0A%20%20%20%20printf%28%22%25d%20%22,%20arr%5Bi%5D%29%3B%0A%20%20%7D%0A%20%20printf%28%22%5Cn%22%29%3B%0A%7D&cumulative=true&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D&textReferences=false
// Visualization of malloc:
//

#include <stdio.h>
#include <stdlib.h>

// #define LEN 5

void SelectionSort(int arr[], int len);
void WrongSwap(int left, int right);
void Swap(int *left, int *right);
int GetMinIndex(const int arr[], int begin, int end);
void Print(const int arr[], int len);
```

```c

int main(void) {
  // int numbers[LEN] = {15, 78, 23, 8, 50};

  int len = 0;
  scanf("%d", &len);

  // VLA
  // int numbers[len];

  // return value: (void *)
  int *numbers = malloc(len * sizeof(*numbers));

  // NULL: null pointer (void *) 0
  if (numbers == NULL) {
    printf("Memory allocation failed!\n");
    return 0;
  }

  for (int i = 0; i < len; i++) {
    scanf("%d", &numbers[i]);
  }

  Print(numbers, len);
  // &numbers[0] (numbers[0] is also a variable) of type (int *)
  SelectionSort(numbers, len);
  Print(numbers, len);

  return 0;
}

// arr: the (copy of the) address of the first element of the `numbers
` array
// int arr[] <-> int *arr (in compilers)
void SelectionSort(int arr[], int len) {
  for (int i = 0; i < len; i++) {
    int min_index = GetMinIndex(arr, i, len);

    // ERROR: WrongSwap(arr[i], arr[min_index]);
    // int temp = arr[i];
    // arr[i] = arr[min_index];
    // arr[min_index] = temp;

    // &arr[i] <=> &(*(arr + i)) <=> arr + i
    Swap(&arr[i], &arr[min_index]);
  }
}

// int arr[] <-> int *arr (in compilers)
int GetMinIndex(const int arr[], int begin, int end) {
  int min = arr[begin];
  int min_index = begin;

  for (int i = begin + 1; i < end; ++i) {
    // arr[i] <-> *(arr + i) <-> *(i + arr) <-> i[arr] (subscript operator)
    if (arr[i] < min) {
      min = arr[i];
      min_index = i;
    }
  }

  return min_index;
}

void WrongSwap(int left, int right) {
  int temp = left;
  left = right;
  right = temp;
}

void Swap(int *left, int *right) {
  int temp = *left;
  *left = *right;
  *right = temp;
}

void Print(const int arr[], int len) {
  printf("\n");
  for (int i = 0; i < len; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n");
}
```