CMPT 125

Introduction to Computing Science and Programming II

September 13, 2021

Pointers and References

Pointers and References

- Each variable is stored in a unique location in the memory.
- Its address is represented by a number (can be accessed using pointers and references).
- Thus, a variable has 3 main features:
  - type
  - value
  - address in memory
- Address can be store in a variable explicitly

```
int x = 5;
int* px = &x;  // pointer to the location of x
printf("The address of %d is %p", x, px);
>>     The address of 5 is 0x9a58af3c4
```

%p prints the address (value of the pointer)

Pointers and References

Dereferencing:

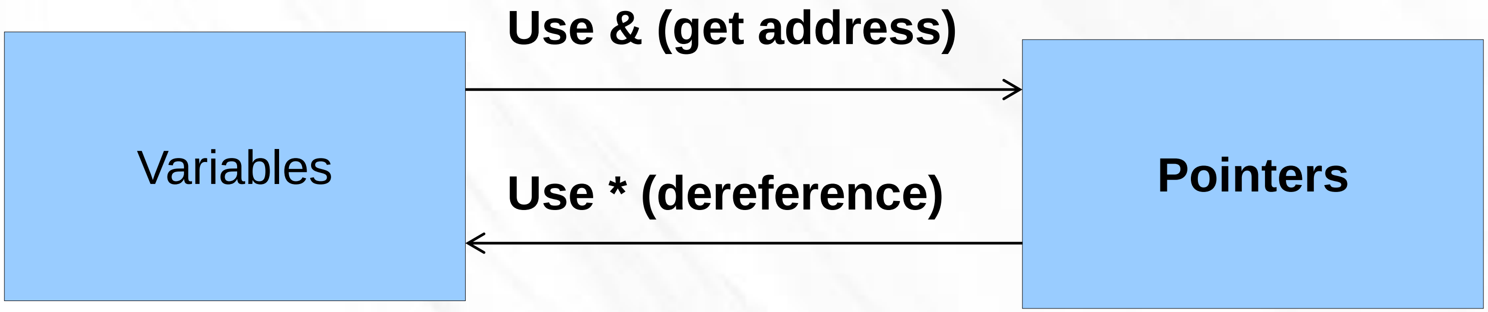
```
int x = 5;   // variable x has value 5
printf("x = %d", x);

int* px = &x; // pointer to the location of x
printf("The value at the address %p is %d", px, *px);


*px = 7;  // dereferencing, changing the value at the address px
printf("x = %d", x);
```

```
>>      x = 5
>>      The value at the address 0x73b8df7a3 is 5
>>      x = 7
```

Pointers and References

Remember the difference:

- Variable – the data
- Pointers – the address



Variables

Use & (get address)

Use * (dereference)

Pointers

Why are pointers useful?

Allows us to modify parameters in a function

```
void swap(int a, int b) {
        int tmp = a;
        a = b;
        b = tmp;
}

int main() {
        int a = 2;
        int b = 3;
        swap(a,b); // the values will not change!!
        ...
```

Why are pointers useful?

Allows us to modify parameters in a function

```
void swap(int* a, int* b) {
        int tmp = *a;
        *a = *b; // dereferencing
        *b = tmp; // dereferencing again
}

int main() {
        int a = 2;
        int b = 3;
        swap(a,b); // WRONG!!! wrong type of parameters
                                (though it will compile on some compilers)
```

Why are pointers useful?

Allows us to modify parameters in a function

```
void swap(int* a, int* b) {
        int tmp = *a;
        *a = *b; // dereferencing
        *b = tmp; // dereferencing again
}

int main() {
        int a = 2;
        int b = 3;
        swap(&a, &b); // the values will change!!

}
```

Why are pointers useful?

- Allows modifying parameters in a function
- Allows us to pass large objects to a function with a single pointer
- Optimization - avoids duplicating data
- Arrays
- Strings

Arrays

Arrays

- An array represents a list of elements
- The list is of a fixed length – once created cannot be resized
- All elements have the same type
- Access by array[index]
- The indexing is [0]..[length-1]

Arrays - example

```
int main() {

        int array[7];

        for (int i=0; i < 7; i++) {
                array[i] = i+5;
        }

        printf("array[3] = %d\n", array[3]); // array[3] = 8

        array[3] = 66;

        printf("array[3] = %d\n", array[3]); // array[3] = 66
        …

}
```

Initializing arrays at declaration

```
int main() {

        int array[7];
        for (int i=0; i < 7; i++) {
                array[i] = i+5;
        }

        …

}
```

OR

```
int main() {
        int array[7] = {5, 6, 7, 8, 9, 10, 11};
```

Iterating through an array

```
int main() {
        int i;
        int array[10] = {0, 1, 8, 2, 18, 3, 6, 2, 2, -4};
        for (i = 0; i < 10; i++)
                printf( "array[%d] = %d\n", i, array[i] );

        for (i = 0; i < 10; i++)
                printf( "array[%d] = %d\n", i, *(array+i) );
}
```

Array – represenation in C

The variable of type *array of ints* is really a *pointer to int*.

The elements are stored in the memory in a contiguous block starting from the position array.

Arithmetics of pointers: Note that size of int = 4
This means that *array+i* really increases by *i*sizeof(int)*.

```
int array[10] = {6, 1, 8, 2, 18, 3, 2, 2, -4};

int* array_ptr = array;
```

Both point to element 6 in the array

Iterating through an array

```
int main() {
        int i;
        int array[10] = {0, 1, 8, 2, 18, 3, 6, 2, 2, -4};
        for (i = 0; i < 10; i++) {
                printf("array[%d] = %d\n", i, *(array+i));
        }
}
```

Iterating through an array

```
int main() {
        int array[10] = {0, 1, 8, 2, 18, 3, 6, 2, 2, -4};

        int* first = array;
        int* last = array + 9;
        int* iter;

        for (iter = first; iter <= last; iter++)

                printf("%d is at the address %p \n",  *iter, iter);
        ...
```

Changing values in an array

```
int main() {
        int array[10] = {0, 1, 8, 2, 18, 3, 6, 2, 2, -4};
        printf("array[3] = %d\n", array[3]); // array[3] = 2;
        array[3] = 66;
        printf("array[3] = %d\n", array[3]); // array[3] = 66;
        *(array+3) = 25;
        printf("array[3] = %d\n", array[3]); // array[3] = 25;
```

Array bounds

```
int array[10] = {0, 1, 8, 2, 18, 3, 6, 2, 2, -4};
```

Q: What happens when trying to access array[-1] or array[10]?

A: Will return *garbage data* or *crash*

Using pointers to access an array

```
int main()  {
        int arr[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
        int* ptr;

        ptr = arr+3;
        printf("*ptr  = %d\n",  *ptr);

        ptr = ptr+2;
        printf("*ptr  = %d\n",  *ptr);

        ptr = &arr[9];
        printf("*ptr  = %d\n",  *ptr);

        arr = &arr[5];
    }
```

NO!! Array cannot be reassigned.
Array is a *constant* pointer

Arrays - recap

- An array represents a list of elements
- The list is of a fixed length
- All elements have the same type
- Access by array[index]
- The indexing is [0]..[length-1]
- The variable of type *array of ints* is really a *constant pointer to int*.
- Can use pointers to access an array

Examples

Write a function that gets an array of floats of length n and outputs the average of the numbers.

```
        float average(float ar[], int n)
```

Write a function that gets two arrays of ints of length n and copies all data from one array into the other.

```
        void array_copy(int dest[], int src[], int n)
```

Examples

Write a function that gets an array of floats of length n and outputs the average of the numbers.

~~float average(float ar[], int n)~~

float average(float const ar[], int n)

OR

float average(const float* ar, int n)

Examples

Write a function that gets two arrays of ints of length n and copies all data from one array into the other.

~~void array_copy(int dest[], int src[], int n)~~

```
void array_copy(int dest[], const int src[], int n)
```

OR

```
void array_copy(int* dest, const int* src, int n)
```

Constants

Constant variables

```
int main()  {
    const int ONE = 1;
    int const TWO = 2;

    ONE = 5; // NO! Modifying the value is not allowed

    return 0;
}
```

Constant pointers

We can also define constant pointers using int* const const_ptr.

int a[] is roughly equivalent to int* const

Note the difference between *const int** and *int* const*

int* const **is a constant pointer**

```
int x, y;
int* const const_ptr = &x;
const_ptr = &y; // NO! Modifying the pointer is not allowed
```

(int*) const

const int* **is a pointer to a constant**

```
const int ONE = 1;
const int* ptr = &ONE;
*ptr = 8; // NO! Modifying the data is not allowed
```

(const int)*

Constant pointers

What’s wrong with this code?

```
void foo(int* a)  {
        … do something…
}

int main()      {
        const int x = 5;
        foo(&x);
        …
}
```

Questions?

Comments?