CMPT 125

Introduction to Computing Science and Programming II

September 15, 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);
```

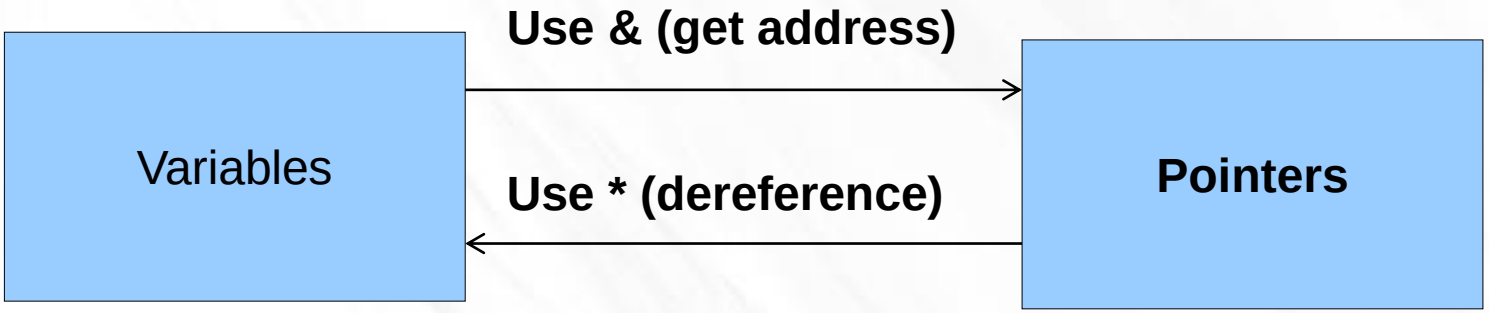
>> The address of 5 is 0x9a58af3c4

%p prints the address (value of the pointer)

Pointers and References

Remember the difference:

- Variable – the data
- Pointers – the address



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]

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) );
}
```

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); // prints *ptr = 3

        ptr = ptr+2;
        printf("*ptr  = %d\n",  *ptr); // prints *ptr = 5

        ptr = &arr[9];    // the address of arr[9]
        printf("*ptr  = %d\n",  *ptr); // prints *ptr = 9

        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 a 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*) const

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

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

(const int)*

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

Constant pointers

What's wrong with this code?

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

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

Strings

Char

How can we implement strings?
A natural idea: an array of chars

char - represents one symbol (letter / digit / punctuation mark)

```
char c1 = ‘a’, c2 = ‘B’, c3 = ‘;’, c4 = ‘6’;
printf("c1 = %c",  c1);
```

char also represents a number (1 byte).
Allows arithmetic on chars

```
char ch = ‘a’;
ch = ch+3; // sets ch = ‘d’
```

Strings

```
#include <stdio.h>
#include <string.h> // includes functions related to strings

int main() {

        char* password = “ABBBAC”;
        char* guess = “ABC”;

        if (password == guess) // WRONG – compares pointers
                … do something…

        if (strcmp(password, guess) == 0)
                printf(“CORRECT”);
        else
                printf(“WRONG”);
}
```

Strings

```
#include <stdio.h>
#include <string.h> // includes functions related to strings

int main() {

        char* password = “ABBBAC”;
        char* guess = “ABC”;

        if (strcmp(password, guess) == 0)
                printf(“CORRECT”);
        else
                printf(“WRONG”);
}
```

Strings

```
#include <stdio.h>
#include <string.h> // includes functions related to strings

int main() {

        char* password = “ABBBAC”;
        char* guess = “ABC”;

        if (strcmp(password, guess) == 0)
                printf(“CORRECT”);
        else
                printf(“WRONG”);
}
```

***<u>Question</u>**: How does strcmp() know the length of the strings?*

Strings

Question: *how does strcmp() know the length of the strings?*

Answer: *A string is an array of chars terminating with '\0'.*

'\0' is the char with value 0.

Comment: *The length of the array can be longer than strlen().*

Example:

```
char* word1 = "Hello";
char word2[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

printf ("%s \n", word1); // prints Hello
printf ("%s \n", word2); // prints Hello
```

Strings

Example:

```
char* word1 = “Hello”;
char word2[6] = {‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’};
```

A comment:

word1 - initializing with “Hello”, creates an array of const chars
immutable strings - cannot be changed

This allows the compiler to perform optimizations on the code

word2 - can be modified, e.g.

```
word2[3] = ‘p’; word2[4] = ‘\0’;
printf (“%s\n”, word2); // prints Help
```

Strings

```
#include <stdio.h>
#include <string.h>

int main() {

    char* password = “ABBBAC”;
    char* guess = “ABC”

    if (strcmp(password,guess) == 0)
            printf(“CORRECT”);
    else
            printf(“WRONG”);
}
```

returns 0 if the strings are equal
returns i>0 if first > last
returns i<0 if first < last

Implement strcmp

```
int strcmp(const char *s1, const char *s2);
```

- If the strings are equal
  returns 0
- Otherwise,
  returns s1[j] – s2[j] for the minimal j where they differ

String.h - two useful functions

int strlen(const char s[])

- Returns the length of the string
- Counts until null terminator
- What happens if there is no ‘\0’ in the string?

char* strcpy(char* dest, const char* src)

- Copies the string src into dest
- Returns the pointer to dest
- What are our requirements about the parameters?
  The length of dest must be sufficient to copy src

- Implement the two functions

String.h – strlen() and strcpy()

```
char str1[]=“Hello";
char str2[40];

strcpy(str2,str1);

printf("%s\n", str2); // prints Hello
printf("%d\n", strlen(str2)); // prints 5

str2[4] = ‘\0’;

printf("%s\n", str2); // prints Hell
printf("%d\n", strlen(str2)); // prints 4
```

String.h – strcat()

```
char* strcat(char* dest, const char* src)
```

- Appends src to the end of dest
- What are our requirements about the parameters?

```
char str[80];
str[0] = '\0';

strcpy (str,"these ");
strcat (str,"strings ");
strcat (str,"are ");
strcat (str,"concatenated.");

printf ("%s", str); // prints "these strings are concatenated."
```

String.h - strcat

```
#include <stdio.h>
#include <string.h>
...

const char* colors[] = {"Red", “Blue", "Green"}; // array of char*
const char* widths[] = {"Thin", "Medium", "Thick", "Bold" };
...

char penText[20]; // array not initialized
...
int penColor = 2, penThickness = 2;

strcpy(penText, colors[penColor]);
strcat(penText, widths[penThickness]);

printf("My pen is %s\n", penText); // prints “My pen is GreenThick”
```

Reading user input

Reading user input

- So far we interacted with the user using printf()
- We can also read user's input using the function scanf()
- The parameter to scanf() is a reference (address) to a variable

```
char name[];
int age;

printf("Enter your name: ");
scanf("%s", name); //&name[0]

printf("Enter your age: ");
scanf("%d", &age);

printf("%s is %d years old\n", name, age);
```

For scanf:
Why are we using &age?
Why name without &?

Questions?

Comments?