CS2100 COMPUTER ORGANISATION

http://www.comp.nus.edu.sg/~cs2100/

Lecture #4a

Pointers and Functions

Questions?

Ask at https://app.sli.do/event/bRPtUxgykAQjjF5XBpLedo

OR

Scan and ask your questions here!
(May be obscured in some slides)

Lecture #4: Pointers and Functions (1/2)

1. Pointers
    - 1.1 Pointer Variable
    - 1.2 Declaring a Pointer
    - 1.3 Assigning Value to a Pointer
    - 1.4 Accessing Value Through Pointer
    - 1.5 Example #1
    - 1.6 Example #2
    - 1.7 Tracing Pointers
    - 1.8 Incrementing a Pointer
    - 1.9 Common Mistake
    - 1.10 Why Do We Use Pointers?

Lecture #4: Pointers and Functions (2/2)

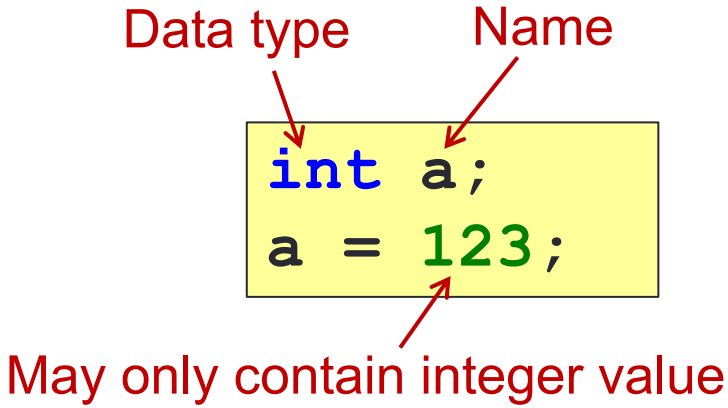
2. Calling Functions
3. User-Defined Functions
4. Pass-by-Value and Scope Rule
   - 4.1 Consequence of Pass-by-Value
5. Functions with Pointer Parameters
   - 5.1 Function to Swap Two Variables
   - 5.2 Examples

1. Pointers (1/3)

- While C is a high-level programming language, it is usually considered to be at the lower end of the spectrum due to a few reasons, among which are:
  - It has pointers which allow direct manipulation of memory contents
  - It has a set of bit manipulation operators, allowing efficient bitwise operations
- In Lecture #2a slide 11, we say that a variable has
  - a name (identifier);
  - a data type; and
  - an address.

1. Pointers (2/3)

- A variable occupies some space in the computer memory, and hence it has an address.
- The programmer usually does not need to know the address of the variable (she simply refers to the variable by its name), but the system keeps track of the variable's address.



Where is variable a located in the memory?

1. Pointers (3/3)

- You may refer to the address of a variable by using the address operator & (ampersand)

Address.c

```
int a = 123;
printf("a = %d\n", a);
printf("&a = %p\n", &a);
```

```
a = 123
&a = ffbff7dc
```

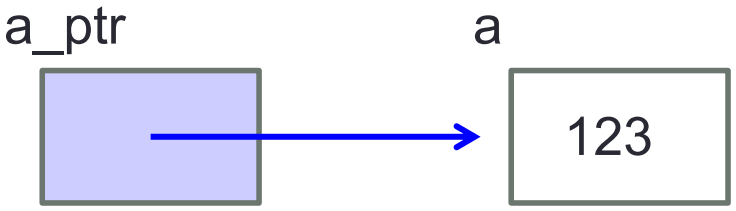
- %p is used as the format specifier for addresses
- Addresses are printed out in hexadecimal (base 16) format
- The address of a variable varies from run to run, as the system allocates any free memory to the variable

1.1 Pointer Variable

- A variable that contains the address of another variable is called a pointer variable, or simply, a pointer.
- Example: a pointer variable a_ptr is shown as a blue box below. It contains the address of variable a.

Assuming that variable a is located at address ffbff7dc.

- Variable a_ptr is said to be pointing to variable a.
- If the address of a is immaterial, we simply draw an arrow from the blue box to the variable it points to.



1.2 Declaring a Pointer

Syntax:

```
type *pointer_name;
```

- pointer_name is the name (identifier) of the pointer
- type is the data type of the variable this pointer may point to

- Example: The following statement declares a pointer variable a_ptr which may point to any int variable
- Good practice to name a pointer with suffix _ptr or _p

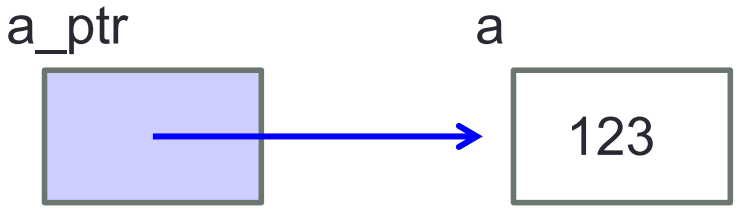
```
int *a_ptr;
```

1.3 Assigning Value to a Pointer

- Since a pointer contains an address, only an address may be assigned to a pointer
- Example: Assigning address of a to a_ptr

```
int a = 123;
int *a_ptr; // declaring an int pointer

a_ptr = &a;
```

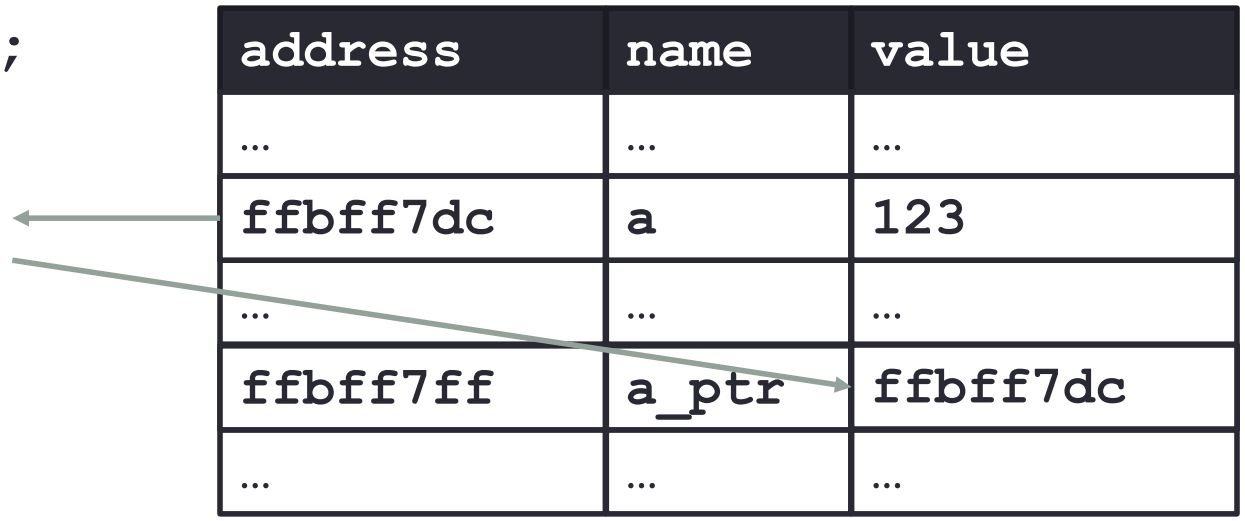


- We may initialise a pointer during its declaration:

```
int a = 123;
int *a_ptr = &a; // initialising a_ptr
```

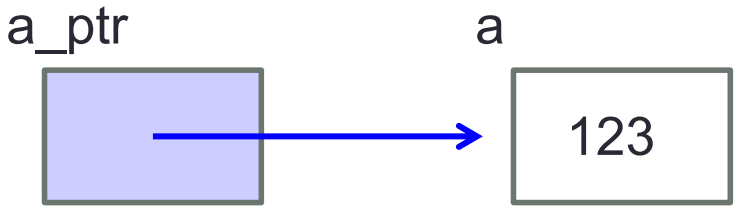
Visualization

- `int a = 123;`
- `int *a_ptr;`
- `a_ptr = &a;`



address	name	value
…	…	…
ffbff7dc	a	123
…	…	…
ffbff7ff	a_ptr	ffbff7dc
…	…	…

1.4 Accessing Variable Through Pointer



- Once we make a_ptr points to a (as shown above), we can now access a directly as usual, or indirectly through a_ptr by using the indirection operator (also called dereferencing operator) *

```
printf("a = %d\n", *a_ptr);
```

≡

```
printf("a = %d\n", a);
```

```
*a_ptr = 456;
```

≡

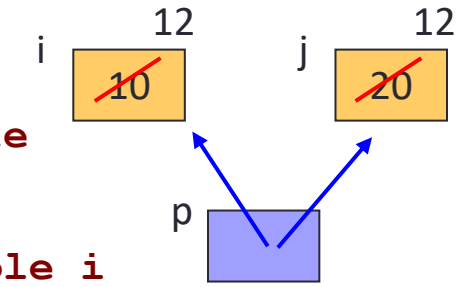
```
a = 456;
```

Hence, *a_ptr is synonymous with a

1.5 Example #1

```
int i = 10, j = 20;
int *p; // p is a pointer to some int variable

p = &i; // p now stores the address of variable i
```



Important! Now `*p` is equivalent to `i`

```
printf("value of i is %d\n", *p);
```

```
value of i is 10
```

```
// *p accesses the value of pointed/referred variable
*p = *p + 2; // increment *p (which is i) by 2
             // same effect as: i = i + 2;

p = &j; // p now stores the address of variable j
```

Important! Now `*p` is equivalent to `j`

```
*p = i;  // value of *p (which is j now) becomes 12
         // same effect as: j = i;
```

1.6 Example #2 (1/2)

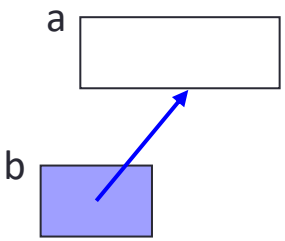
Pointer.c

```
#include <stdio.h>

int main(void) {
   double a, *b;

   b = &a;
   *b = 12.34;
   printf("%f\n", a);

   return 0;
}
```



Can you draw the picture?
What is the output?

```
12.340000
```

What is the output if the `printf()` statement is changed to the following?

```
printf("%f\n", *b);
```

```
12.340000
```

```
printf("%f\n", b);
```

Compile with warning

```
printf("%f\n", *a);
```

Error

What is the proper way to print a pointer?
(Seldom need to do this.)

Value in hexadecimal; varies from run to run.

```
printf("%p\n", b);
```

```
ffbff6a0
```

1.6 Example #2 (2/2)

- How do we interpret the declaration?

```
double a, *b;
```

- The above is equivalent to

```
double a;  // this is straight-forward: a is a double variable
double *b;
```

- We can read the second declaration as
  - *b is a double variable, so this implies that ...
  - b is a pointer to some double variable
- The following are equivalent:

```
double a;
double *b;
b = &a;
```

```
double a;
double *b = &a;
```

But this is not the same as above (and it is not legal):

```
double a;
double b = &a;
```

End of File