ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

PROGRAMMING METHODOLOGY
(PHƯƠNG PHÁP LẬP TRÌNH)

UNIT 8: Pointers

Acknowledgement

- The contents of these slides have origin from School of Computing, National University of Singapore.
- We greatly appreciate support from Mr. Aaron Tan Tuck Choy for kindly sharing these materials.

Policies for students

- These contents are only used for students PERSONALLY.
- Students are NOT allowed to modify or deliver these contents to anywhere or anyone for any purpose.

Recording of modifications

- Currently, there are no modification on these contents.

Unit 8: Pointers

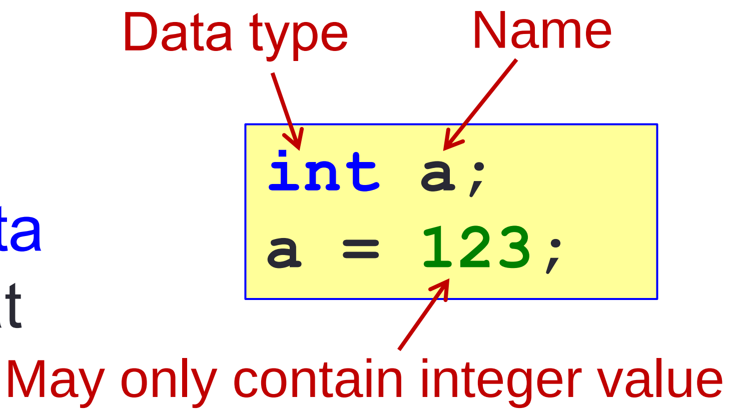
Objective:

- Learning about pointers and how to use them to access other variables

Unit 8: Pointers

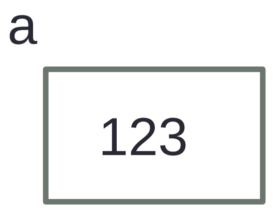
1. Variable and Its Address (1/2)

- A variable has a unique name (identifier) in the function it is declared in, it belongs to some data type, and it contains a value of that type.



```
int a;
a = 123;
```

- A variable occupies some space in the 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. Variable and Its Address (2/2)

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

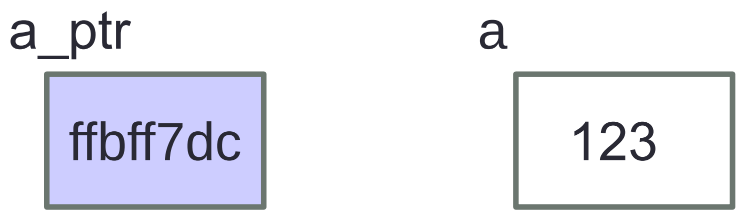
```
int a = 123;
printf("a = %d\n", a);
printf("&a = %p\n", &a);
```

```
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
- Test out Unit8_Address.c

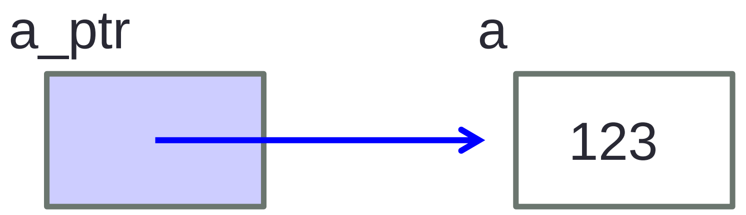
2. Pointer

- 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.



3. 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

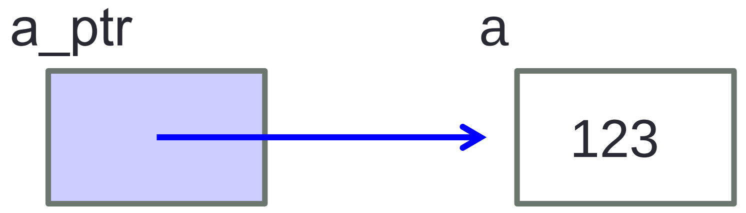
- Example: The following statement declares a pointer variable a_ptr which may point to any int variable
- Good practice to name a pointer with suffix _ptr or _p

```
int *a_ptr;
```

4. Assigning Value to a Pointer

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

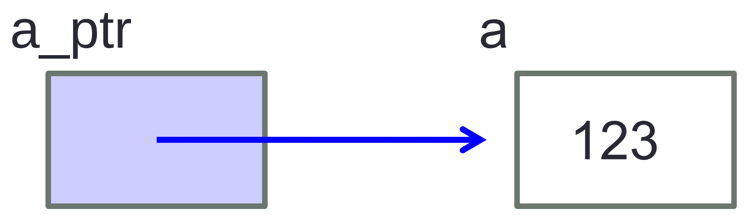
```
int a = 123;
int *a_ptr; // declaring an int pointer

a_ptr = &a;
```



- We may initialise a pointer during its declaration:

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

5. 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

6. 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`

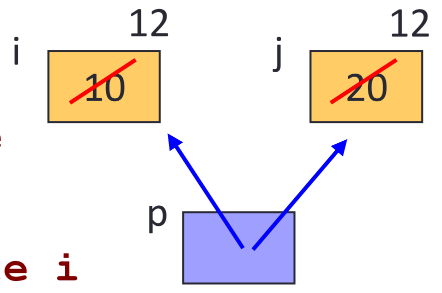
```
printf("value of i is %d\n", *p);
```

```
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;
```

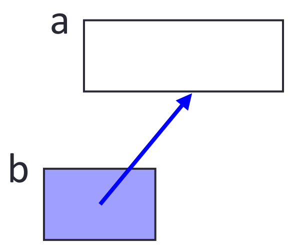


6. Example #2 (1/2)

Unit8_Pointer.c

```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?

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

```
12.340000
```

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

Compile with warning

```c
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.

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

```
ffbff6a0
```

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;
```

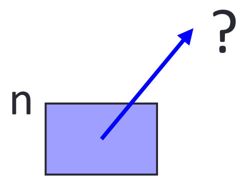
✗

7. Common Mistake

Unit8_Common_Mistake.c

```c
#include <stdio.h>

int main(void) {
   int *n;

   *n = 123;
   printf("%d\n", *n);

   return 0;
}
```

What's wrong with this?
Can you draw the picture?



- Where is the pointer n pointing to?
- Where is the value 123 assigned to?
- Result: Segmentation Fault (core dumped)
  - Remove the file "core" from your directory. It takes up a lot of space!

8. Why Do We Use Pointers?

- It might appear that having a pointer to point to a variable is redundant since we can access the variable directly
- The purpose of pointers is apparent later when we pass the address of a variable into a function, in the following scenarios:
  - To pass the address of the first element of an array to a function so that the function can access all elements in the array (Unit 9 Arrays, and Unit 10 Multidimensional Arrays)
  - To pass the addresses of two or more variables to a function so that the function can pass back to its caller new values for the variables (Unit 11 Modular Programming – More about Functions)

Summary

- In this unit, you have learned about
  - Declaring a pointer variable
  - Using a pointer variable to point to a variable
  - Hence, assessing a variable through the pointer variable that points to it

End of File