Structures

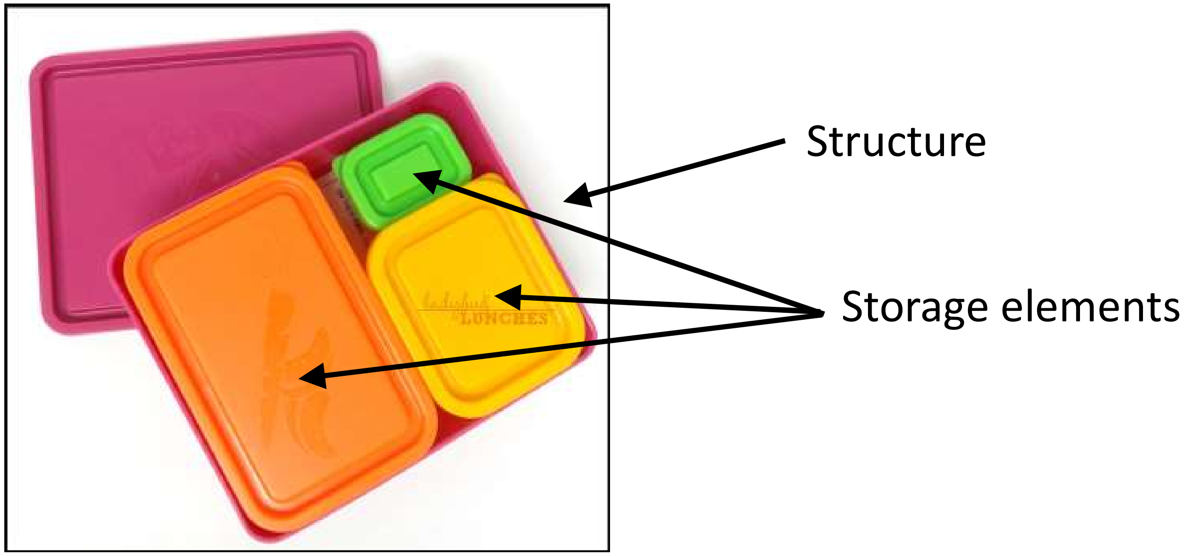
For better data organization (exam included)

Outline

- Introduction
- Basic Syntax
  - Defining a structure
  - Accessing members of a structure
- `typedef`
- More Syntax
  - Initialize the data members in a structure
  - Copy a structure
  - Passing structures to a function by value or by pointer
  - Returning a structure from a function

Introduction

- A ***structure*** is a collection of related storage elements under a single name.



- The elements in a structure can be of different types.
- All elements in a structure typically relate semantically.

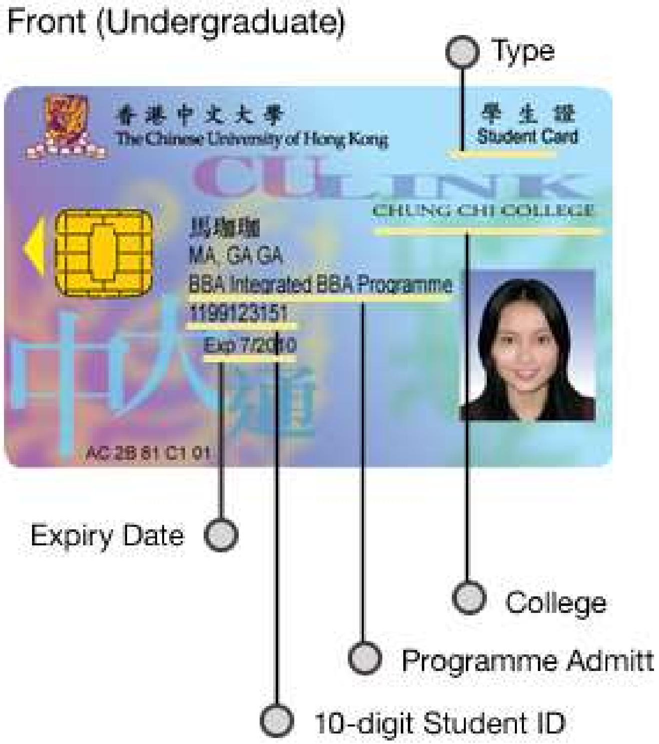
A simple example…

If you want to write a program to process “student data” in CUHK

- How if we simply have one large array of data for each item:
  - Student name -> string
  - Student ID -> string or integer
  - Student age -> integer or char
  - courses -> an array of strings

  etc.
- Compute “mean” student age for all male students in our class?
- We have to keep individual arrays!

“structure” helps to **group relevant data** together in an *organized* way by creating a **new data type**



A simple example…

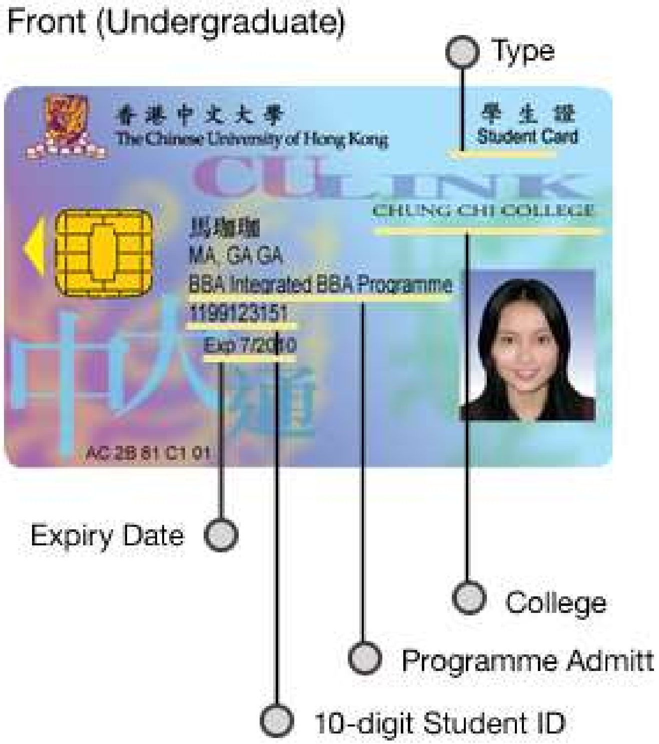
With “structure”, we can create a new data type called “Student” with the following data members in the structure:

- name -> string
- ID -> string or integer
- age -> integer or char
- courses -> an array of strings

Etc.

Then, we may even create an array of “Student”, e.g.,

```
Student students[100] ;
```



Outline

- ~~Introduction~~

- Basic Syntax
  - Defining a structure
  - Accessing members of a structure

- `typedef`

- More Syntax
  - Initialize the data members in a structure
  - Copy a structure
  - Passing structures to a function by value or by pointer
  - Returning a structure from a function

Define a structure data type (syntax)

```
struct struct_name
{
   type1 member1 ;
   type2 member2a , member2b ;
   …
   typeN member ;
} ;
```

Define what kinds of data to hold

- We define *structure type* (or composite type) using the keyword `struct`.

- We need to define a structure type before we can declare variables of that type to store values.

Structure syntax (example)

```c
struct date
{
    int day , month , year ;
};

int main( void )
{
    struct date d1 , d2 ;

    // Assign 10 to
    //  member "day" of d1
    d1.day = 10 ;

    // Assign 2022 to
    //  member "year" of d2
    d2.year = 2022 ;

    return 0 ;
}
```

Define a new structure type named `date`

In this definition, we specify that each "value" of this type contains three members (`day`, `month`, and `year`), which are of type `int`.

Structure syntax (example)

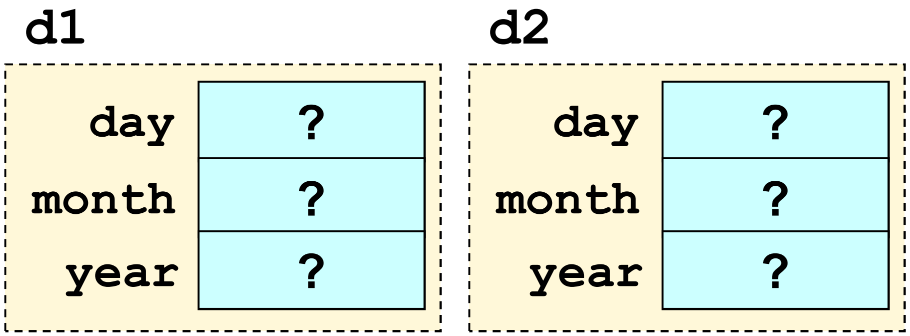
```
1   struct date
2   {
3       int day , month , year ;
4   };
5
6   int main( void )
7   {
8       struct date d1 , d2 ;
9
10      // Assign 10 to
11      //  member "day" of d1
12      d1.day = 10 ;
13
14      // Assign 2022 to
15      //  member "year" of d2
16      d2.year = 2022 ;
17
18      return 0 ;
19  }
```

"`struct date`" is the name of the newly defined data type.

At line 8, we declare two variables `d1` & `d2` of type "`struct date`"

Structure syntax (example)

```c
struct date
{
    int day , month , year ;
};

int main( void )
{
    struct date d1 , d2 ;

    // Assign 10 to
    //  member "day" of d1
    d1.day = 10 ;

    // Assign 2022 to
    //  member "year" of d2
    d2.year = 2022 ;

    return 0 ;
}`
```



d1

day	?
month	?
year	?

d2

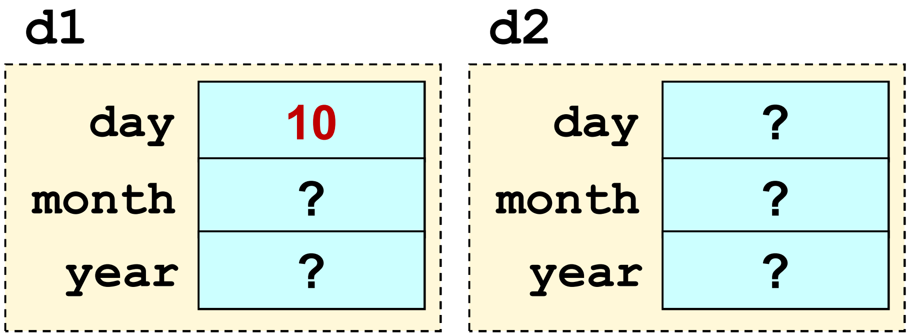
day	?
month	?
year	?

Each variable of type "struct date" has its own members.

Initially the members are uninitialized.

Structure syntax (example)

```c
struct date
{
    int day , month , year ;
};

int main( void )
{
    struct date d1 , d2 ;

    // Assign 10 to
    //  member "day" of d1
    d1.day = 10 ;

    // Assign 2022 to
    //  member "year" of d2
    d2.year = 2022 ;

    return 0 ;
}
```



d1

day	10
month	?
year	?

d2

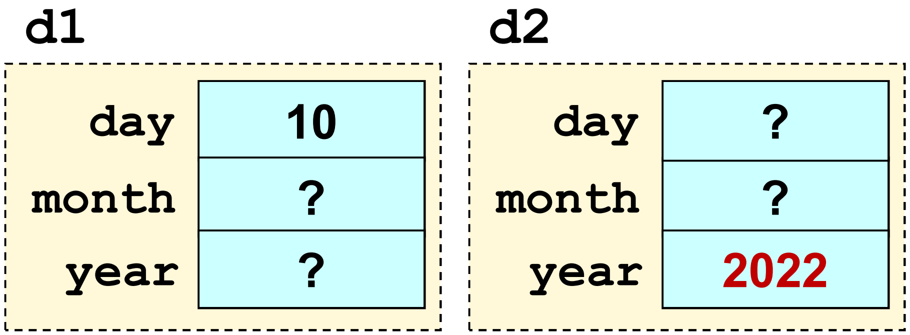
day	?
month	?
year	?

The *dot operator* (.) is called a *member selection* operator.

`d1.day` means "select the member `day` of `d1`".

Structure syntax (example)

```
struct date
{
    int day , month , year ;
};

int main( void )
{
    struct date d1 , d2 ;

    // Assign 10 to
    //  member "day" of d1
    d1.day = 10 ;

    // Assign 2022 to
    //  member "year" of d2
    d2.year = 2022 ;

    return 0 ;
}
```



d1

day	10
month	?
year	?

d2

day	?
month	?
year	2022

A member of type `int` is just like a regular variable of type `int`. Any syntax that is valid for a variable of type `int` is also valid for a member of type `int`.

Access structure's members (example)

```c
struct date { int day , month , year ; } ;

int main( void )
{
    struct date today , dob ;   // Declare 2 variables

    today.year  = 2022 ;
    today.month = 10   ;
    today.day   = 31   ;

    printf( "Date of birth (dd mm yyyy)? " );
    scanf( "%d%d%d" , & dob.day , & dob.month , & dob.year );

   if (  today.month  > dob.month
    || ( today.month == dob.month && today.day >= dob.day ) )
      printf( "Age = %d\n" , today.year - dob.year );
   else
      printf( "Age = %d\n" , today.year - dob.year - 1 );
   return 0 ;
}
```

However, “struct date” doesn't look like a data type

Outline

- ~~Introduction~~

- ~~Basic Syntax~~
  - ~~Defining a structure~~
  - ~~Accessing members of a structure~~

- `typedef`

- More Syntax
  - Initialize the data members in a structure
  - Copy a structure
  - Passing structures to a function by value or by pointer
  - Returning a structure from a function

Define an alias to an existing data type

- We can introduce an alias (别名) to an existing data type using `typedef`.

- Syntax

```
typedef existing_type_name alias ;
```

- After the declaration, both `alias` and `existing_type_name` refer to the same data type.

Define an alias to an existing data type

```
struct date {
    int day , month , year ;
};

int main( void )
{
    struct date d1 , d2 ;
    ...
}
```

Common convention:
When we define a new data type, its 1st character is usually in uppercase

```
struct date {
    int day , month , year ;
};

typedef struct date Date ;
  // From this point, "Date" is
  // an alias of "struct date"

int main( void )
{
    Date d1 , d2 ;
    struct date d3 ;
    // Variables d1, d2 and d3
    // have the same data type
    ...
}
```

Different ways to combine typedef with struct

1

```
struct date {
    int day , month , year ;
};

typedef struct date Date ;
```

In two separate declarations:

First define a struct type named "struct date"; then define the alias

2

```
typedef struct date {
    int day , month , year ;
} Date ;
```

In one declaration:

Define "struct date" and the alias

3

```
typedef struct {
    int day , month , year ;
} Date ;
```

In one declaration:

Define a struct type with no name and define the alias

With the first two approaches, the type can be referred in the program as "struct date" or "Date". With the 3rd approach, the type can only be referred in the program as "Date".

Outline

- ~~Introduction~~

- ~~Basic Syntax~~
  - ~~Defining a structure~~
  - ~~Accessing members of a structure~~

- ~~typedef~~

- More Syntax
  - Initialize the data members in a structure
  - Copy a structure
  - Passing structures to a function by value or by pointer
  - Returning a structure from a function

Syntax #1: Initialize a structure

```
typedef struct date
{
    int day , month , year ;

} Date ;

int main( void )
{
    Date xmas = { 25 , 12 , 2022 };
... ...
}
```

- The **order** of the values in the initializer { … } must match the order of the members in the structure definition.

Syntax #2: Copy a structure variable

```
Date d1 , d2 = { 1 , 1 , 2022 };

d1 = d2 ;  // Copy d2 to d1 (byte by byte)
```

- A struct value can be copied using the assignment operator.

Syntax #3: Pass a structure to a func. (by value)

```
void printDate( Date d )   // A structure copy here!
{
    printf( "%d-%d-%d" , d.day , d.month , d.year );
}

int main( void )
{
    Date xmas = { 25 , 12 , 2022 };
    printDate( xmas );  // Implicitly d = xmas ;
    return 0 ;
}
```

- A structure can be passed by value to a function

Syntax #3: Pass a structure to a func. (by value)

```
void printDate( Date d )   // A structure copy here!
{
    printf( "%d-%d-%d" , d.day , d.month , d.year );
    d.month = 11 ;
}

int main( void )
{
    Date xmas = { 25 , 12 , 2022 };
    printDate( xmas ); printf( "%d\n", xmas.month );
    return 0 ;
}
```

- d and xmas are independent variables (different memory)

Syntax #4: Pass a structure to a func. (by pointer)

```
void printDate( Date * d )   // no structure copy here!
{
    printf( "%d-%d-%d" ,
            (*d).day , (*d).month , (*d).year );
}

int main( void )
{
    Date xmas = { 25 , 12 , 2022 };
    printDate( & xmas );  // Implicitly d = & xmas ;
    return 0 ;
}
```

Let’s revisit this page after Lecture 11: Pointers

- More efficient to pass a structure by pointer; this saves the effort to copy the entire structure

Syntax #4: Pass a structure to a func. (by pointer)

```
void printDate( Date * d )   // no structure copy here!
{
    printf( "%d-%d-%d" ,
            d->day , d->month , d->year );
    (*d).day = 10 ;
}

int main( void )
{
    Date xmas = { 25 , 12 , 2022 };
    printDate( & xmas );
    return 0 ;
}
```

Let’s revisit this page after Lecture 11: Pointers

- But there's a **risk**! Modifying d will modify xmas!

Syntax #5: Return a structure from a function

```
Date readDate()
{
    Date d ;   // local variable
    scanf( "%d%d%d" , & d.day , & d.month , & d.year );
    return d ;
}

int main( void )
{
    struct date d ;
    d = readDate() ;   // The returned value is
                       // copied to d.
    return 0 ;
}
```

Syntax #5: Return a structure from a function

```
Date * readDate()
{
    Date d ;   // local variable
    scanf( "%d%d%d" , & d.day , & d.month , & d.year );
    return & d ;
}

int main( void )
{
    struct date * d ;  // d is a pointer to a structure
    d = readDate() ;   // The returned address is
                       // copied to d.
    return 0 ;
}
```

What is the problem here?

What is the lifetime of d in readDate()?

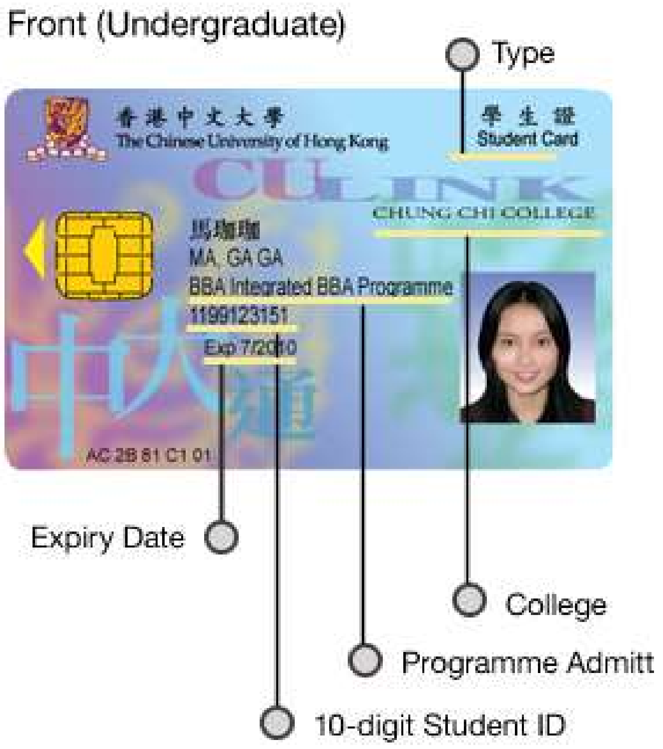
Let's revisit this page after Lecture 11: Pointers

Back to the simple example…

If you want to write a program to process “student data” in CUHK

- When we have “struct”, we can define a Student structure:
  - Student name -> string
  - Student ID -> string or integer
  - Student age -> integer or char
  - courses -> an array of strings

  etc.
- And create one array of Student
- To compute mean student age for all students in our class?



More example: How about our Project?

```
#ifndef _GAME_HH_
#define _GAME_HH_

#define BOARD_SIZE ...
typedef struct {
  int board[ BOARD_SIZE ] ;
  int step  ;  // how many steps we have done
  int score ;  // current score
  ......
} GameState ;

GameState init_game() ;   // return the whole board
void print_board( GameState * state ) ;  // pointer *
......

#endif
```

Let’s revisit these in Lecture 10: multi-file compilation

Let’s revisit this page after Lecture 11: Pointers

More example: How about our Project?

```
void print_board( GameState * state )  // pointer *
{
    printf( "Current Score is %d\n" , state->score );
    ... // print out the game board
}
...

void main ( ... )
{
    GameState state = init_game() ;
    ...
    print_board( & state );
    ...
}
```

Let’s revisit this page after Lecture 11: Pointers

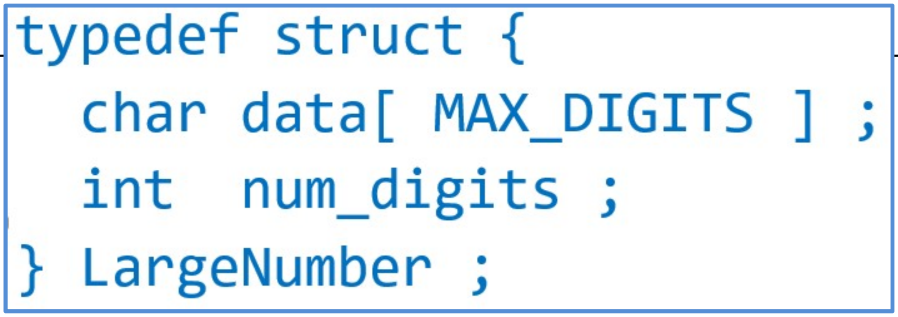
Another example: LargeNumber

```
#ifndef _LARGE_NUMBER_HH_
#define _LARGE_NUMBER_HH_

#define MAX_DIGITS    10000

typedef struct {
  char data[ MAX_DIGITS ] ;
  int  num_digits ;
} LargeNumber ;

LargeNumber add_Large_Number( LargeNumber * a ,
                              LargeNumber * b ) ;
void print_Large_Number( LargeNumber * num ) ;
......

#endif
```

Let’s revisit this page after Lecture 11: Pointers

Array of structure data

- We may create a **static** array of structure data
- No new syntax here!!!

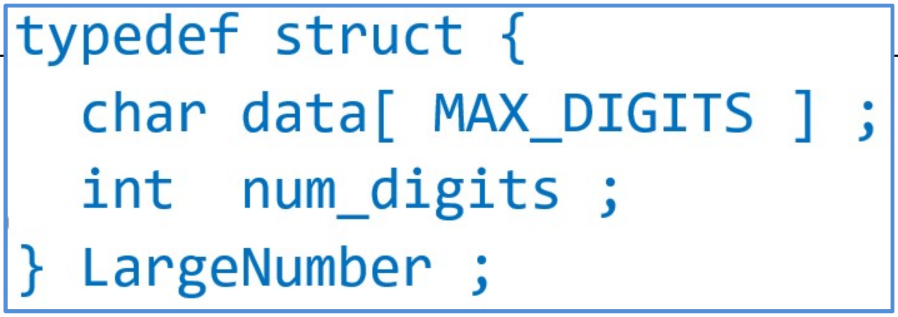


```
typedef struct {
  char data[ MAX_DIGITS ] ;
  int  num_digits ;
} LargeNumber ;
```

```
#include "large_number.h"
...
int main( void )
{
    LargeNumber num[10];
    for ( int i = 0 ; i < 10 ; i ++ ) {
        num[i] . num_digits = 1  ;
        strcpy( num[i] . data , "0" );
    }
    ...
}
```

Array of structure data

- We may create a **dynamic** array of structure data
- No new syntax here!!!



```
typedef struct {
  char data[ MAX_DIGITS ] ;
  int  num_digits ;
} LargeNumber ;
```

```
#include "large_number.h"
...
int main( void )
{
   LargeNumber * num =
      ( LargeNumber * ) malloc( sizeof(LargeNumber) * 10 );
   for ( int i = 0 ; i < 10 ; i ++ ) {
      num[i] . num_digits = 1  ;
      strcpy( num[i] . data , "0" );
   }
   ...   // remember to free(num)
}
```

Let's revisit this page after Lecture 11: Pointers

Summary

- A structure is a mean for programmers to group related variables inside one "**container**" – Kind of a **user-defined "composite" data type**!!!
- Each member of a structure is like a regular variable. Their main difference is in the syntax.
- Syntax that you should remember:
  - Define a structure (with and without `typedef`)
  - How to initialize a structure, access data members of a structure, pass structure to a func., and return a structure from a func.
- In the future, when you learn C++, struct -> class