

typedef

We can use the keyword `typedef` to give a name to a type:

```
typedef double real;
```

This means variables can be declared as **real** but they will actually be of type **double**.

Do not overuse typedef - it can make programs harder to read, e.g.:

```
typedef int darthVader;
```

```
darthVader main(void) {  
    darthVader i,j;  
    ....  
}
```

Using typedef to make programs portable

Suppose have a program that does floating-point calculations. If we use a typedef'ed name for all variable, e.g.:

```
typedef double real;  
  
real matrix[1000][1000][1000];  
real myAtanh(real x) {  
    real u = (1.0 - x)/(1.0 + x);  
    return -0.5 * log(u);  
}
```

If we move to a platform with little RAM, we can save memory (and lose precision) just by changing the typedef:

```
typedef float real;
```

enums

- ENUMS (enumerations) is a custom data type, which describes set of possible values in a programmer-defined category
- For example, days of the week

```
#include <stdio.h>  
enum weekdays {Monday, Tuesday, Wednesday,  
               Thursday, Friday, Saturday, Sunday};  
  
int main() {  
    enum weekdays day;  
    day = tuesday;  
    if (day == Tuesday) {  
        printf("Lecture day\n");  
    }  
    return 0;  
}
```

Using typedef to make programs portable

```
#include <stdio.h>  
enum weekdays {Monday, Tuesday, Wednesday,  
               Thursday, Friday, Saturday, Sunday};  
typedef enum weekdays week;  
int main() {  
    week day; // make a new variable called day,  
    ....  
}
```

structs

- We have seen simple types e.g. **int**, **char**, **double**
 - ▶ variables of these types hold single values
- A compound type: structs
 - ▶ structs hold multiple values (fields)
 - ▶ structs are heterogeneous - fields can be different type
 - ▶ struct field selected using name
 - ▶ struct fields are fixed

structs - example

If we define a struct that holds CP1511 student details:

```
#define MAX_NAME 64
#define N_LABS 12
struct student {
    int zid;
    double totallabMarks;
    double assignment1Mark;
    double assignment2Mark;
}
```

We can declare an array to hold the details of all students: (We will learn about it later)

```
struct student cp1511Students[400];
```

Combining structs and typedef

Common to use typedef to give name to a struct type.

```
struct student {
    int zid;
    double totallabMarks;
    double assignment1Mark;
    double assignment2Mark;
}
typedef struct student Student;
Student cp1511Students[400];
```

We use the convention that for the typedef we use should be the same as the tag, but starting with a capital letter.

Assigning values to structs

```
int main(void){
    Student s;
    s.zid = 12345678;
    s.totallabMarks = 14;
    s.assignment1Mark = 10;
    //etc
}
```

Assigning structs to structs

Unlike arrays, it is possible to copy all components of a structure in a single assignment:

```
Student student1, student2;  
...  
student2 = student1;
```

Comparing structs

It is *not* possible to compare all components with a single comparison:

```
if (student1 == student2) // NOT allowed!
```

If you want to compare two structures, you need to write a function to compare them component-by-component and decide whether they are “the same”.

Nested Structures

One structure can be nested inside another

```
typedef struct date    Date;  
typedef struct time    Time;  
typedef struct parkingTicket ParkingTicket;  
struct date {  
    int day, month, year;  
};  
struct time {  
    int hour, minute;  
};  
struct parkingTicket {  
    Date    date;  
    Time    time;  
    char    plate[MAX_PLATE];  
};
```