
UNSW
SYDNEY
Australia's
Global
University
COMP1911 - Computing 1A

Array Representation

We can even make a pointer point to the middle of an array:

```
int nums[] = {1, 2, 3, 4, 5};
int *p = &nums[2];
printf("%d %d\n", *p,nums[0]);
```

So is there a difference between an array variable and a pointer?

```
int i = 5;
p = &i;     // this is OK
nums = &i; // this is an error
```

Unlike a regular pointer, an array variable is defined to point to the beginning of the array, it is constant and may not be modified.

pointer to point to the array

```
int nums[] = {1, 2, 3, 4, 5};
```

```
int *p = {1,2,3,4,5};
```

```
int *p;
p = nums;
p = &nums[0];
```

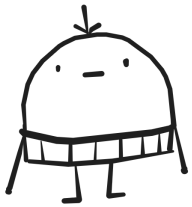
```
printf("%d %d %d %d\n", p[0],
p[1], p[2], p[3]);
```

```
char str[]="hello!";
//char *str="hello!"
char *str1 ;
str1=str;
```

```
char str2[10];
strcpy(str2, str);
```

```
char *str2 =
malloc(sizeof(char)*20);
strcpy(str2,"I love food");
```

13. Structs



In this lecture we will cover:

- Structs

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;
```

structs

- We have seen simple types e.g. **int, char, double**
  - variables of these types hold single values
- We have seen a compound type: arrays
  - array variables hold multiple values
  - arrays are homogenous - every array element is the same type
  - array element selected using integer index
  - array size can be determined at runtime (using malloc)
- Another compound type: structs
  - structs hold multiple values (fields)
  - struct are heterogeneous - fields can be different type
  - struct field selected using name
  - struct fields fixed

structs - example 1

It's annoying to pass an x, y, and z coordinate into a function.

```
void myFunction(int x, int y, int z);

int main(int argc, char *argv[]) {
        int x = 1;
        int y = 2;
        int z = 3;
        myFunction(x, y, z);
        return 0;
}

void myFunction(int x, int y, int z) {
        printf("(%d, %d, %d)\n", x, y, z);
}
```

Defining a Structure

To define a structure, we use the **struct** statement. The struct statement defines a new data type:

```
struct structureTag {
   dataType member1;
   dataType member2;
   ...
} [one or more structure variables];
```

Each member definition is a normal variable definition.

```
struct point3D {
   int x;
   int y;
   int z;
};
```

Create struct variables

```
struct point3D {
    int x;
    int y;
    int z;
} point1, point2, points[10];
```

```
struct point3D {
    int x;
    int y;
    int z;
};
struct point3D point1, point2, points[10];
```

Access members of a structure

There are two types of operators used for accessing members of a structure.

1. Member operator: .
2. Structure pointer operator: ->

structs - example 1

It's annoying to pass an x, y, and z coordinate into a function.

```
struct point3D {
int x, y, z;
};
void myFunction(struct point3D threeD) {
printf("(%d, %d, %d)\n", threeD.x, threeD.y, threeD.z);
}
int main(int argc, char *argv[]) {
struct point3D threeD;
threeD.x = 1;
threeD.y = 2;
threeD.z = 3;
myFunction(threeD);
return 0;
}
```

structs - example 2

If we define a struct that holds COMP1911 student details:

```
#define MAX_NAME 64
#define N_LABS  9

struct student {
    int zid;
    char name[MAX_NAME];
    double labMarks[N_LABS]
    double assignment1Mark;
    double assignment2Mark;
}
```

We can declare an array to hold the details of all students:

```
struct student comp1911Students[400];
```

combining structs and typedef

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

```
struct student {
    int zid;
    char name[MAX_NAME];
    double labMarks[N_LABS]
    double assignment1Mark;
    double assignment2Mark;
}

typedef struct student Student;

Student comp1911Students[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 structs

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

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

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”.

structs and functions

A structure can be passed as a parameter to a function:

```
void printStudent(Student s) {
    printf("%s z%d\n", s.name, s.zid);
}
```

Unlike arrays, a copy will be made of the entire structure, and only this copy will be passed to the function. Unlike arrays, a function can return a struct:

```
Student readStudentFromFile(char filename[]) {
    ....
}
```

Pointers to structs

If a function needs to modify a structs field or if we want to avoid the inefficiency of copying the entire struct, we can instead pass a pointer to the struct as a parameter:

```
int scanZid(Student *s) {
    return scanf("%d", &((*s).zid));
}
```

The “arrow” operator is more readable :

```
int scan_zid(Student *s) {
    return scanf("%d", &(s->zid));
}
```

If **s** is a pointer to a struct **s->fileld** is equivalent to **(*s).field**

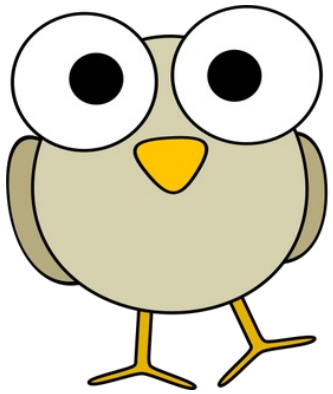
Nested Structures

One structure can be nested inside another

```
typedef struct date     Date;
typedef struct time     Time;
typedef struct speeding Speeding;
```

```
struct date {
    int day, month, year;
};
struct time {
    int hour, minute;
};
struct speeding {
    Date   date;
    Time   time;
    double speed;
    char   plate[MAX_PLATE];
};
```

Question



??