Arrays and Structures

Structures & Unions

Joseph Chuang-Chieh Lin (林莊傑)

Department of Computer Science & Engineering,
National Taiwan Ocean University

Fall 2024

Outline

Outline


The Arrays as An ADT

- Arrays.
  - a set of pairs <index, value>.
  - for each index, there is a value associated with that index.
  - a consecutive set of memory locations.
  - mathematical terms: correspondence, mapping, etc.

Implementation of 1-D Array

- `int list[5]`
  - Five consecutive memory locations are allocated.
  - The address of `list[0]`: base address.
- `int list[5], *plist[5];`
  - sample code for the second.

variable	memory address
`list[0]`	base address $= b$
`list[1]`	$b + 1 \times$ `sizeof(int)`
`list[2]`	$b + 2 \times$ `sizeof(int)`
`list[3]`	$b + 3 \times$ `sizeof(int)`
`list[4]`	$b + 4 \times$ `sizeof(int)`

Array in C

- Compare `int *list1` and `int list2[5]` in C.
  - Both `list1` and `list2` are pointers.
  - `list2` reserves five memory locations.
- Some notations:
  - `list2`:
  - `(list2+i)`:
  - `*(list2+i)`:


Array in C

- Compare `int *list1` and `int list2[5]` in C.
  - Both `list1` and `list2` are pointers.
  - `list2` reserves five memory locations.
- Some notations:
  - `list2`: a pointer to `list2[0]`
  - `(list2+i)`: `&list2[i]`
  - `*(list2+i)`: `list2[i]`

Outline

Structures

- An array is a collection of data of the same type.
  - `int arr[] = { 0, 1, 2, 3, 4 };`
- A structure is a collection of data items, where each item is identified as to its type and name.

```
struct employee {
    char name[10];
    int age;
    double salary;
};
struct employee person;
```

```
struct employee {
    char name[10];
    int age;
    double salary;
} person;
```

Usage of a struct Variable

```c
struct employee {
    char name[10];
    int age;
    double salary;
} person;

strcpy(person.name, "Peter");
person.age = 10;
person.salary = 80000;
```

```c
struct employee {
    string name; // C++
    int age;
    double salary;
} person;

person.name = "Peter";
person.age = 10;
person.salary = 80000;
```


typedef

```c
typedef int COUNT;
COUNT num1, num2;
typedef struct employee HUMAN_BEING;

HUMAN_BEING person1, person2;
strcpy(person1.name, "Peter");
person.age = 10;
person.salary = 80000;
```

A structure within a structure is possible

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

typedef struct {
    char name[10];
    int age;
    float salary;
    date birthday;
} HUMAN_BEING;
```

The usage:

```
HUMAN_BEING person;
person.birthday.month = 10;
person.birthday.day = 31;
person.birthday.year = 1979;
```

Outline

Unions

- Similar to structures.
- The fields of a union must share their memory space.
- Only one field of the union is active at any given time.

```
typedef struct {
    int sex;
    union {
        int kid;
        int beard;
    } u;
} SEX_TYPE;
```

Characteristics of Unions (code)

- The size of the union is the size of the largest member.
- Only one member can contain data at the same time.

```c
union union1 {
    int x;
    int y;
} U1;

union union2 {
    int arr[10];
    char y;
} U2;
```

```c
int size1 = sizeof(U1);
int size2 = sizeof(U2);
printf("Sizeof U1: %d\n", size1);
printf("Sizeof U2: %d\n", size2);
```

An Application Example

binary tree (only leaf nodes have data)

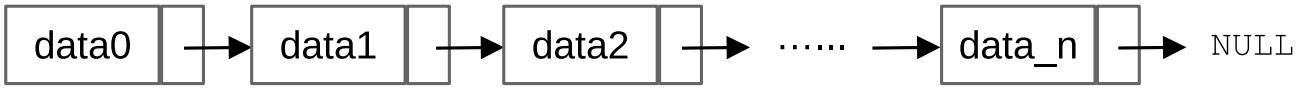
```
struct Node {
    bool is_leaf;
    struct Node* left;
    struct Node* right;
    double data;
};
```

```
struct Node {
    bool is_leaf;
    union {
        struct {
            struct Node* left;
            struct Node* right;
        } internal;
        double data;
    } info;
};
```

Reference: GeeksforGeeks

Self-Referential Structures

One or more of its components is a pointer pointing to itself.




```c
struct Node {
    int data;
    struct Node* link;
};
typedef struct Node list;
```

```c
list item0, item1, item2;
item0.data = data0;
item1.data = data1;
item2.data = data2;
item0.link = &item1;
item1.link = &item2;
item2.link = NULL;
```

Discussions