NWEN 241
Arrays and Pointers I

Qiang Fu

School of Engineering and Computer Science
Victoria University of Wellington

Victoria
UNIVERSITY OF WELLINGTON
Te Whare Wānanga
o te Ūpoko o te Ika a Māui
CAPITAL CITY UNIVERSITY

This Lecture

- Arrays and pointers

Arrays

- An Array is a collection of data items
- All the array elements must have the same type

```
int i[10];   /* the array has 10 elements */
float f[20];
char c[30];
```

- We number array elements from 0

```
int i[10] = {0,1,2,3,4,5,6,7,8,9};
/* i[0]=0, i[1]=1, ..., i[9]=9 */
```

Arrays

- An Array is a collection of data items
- All the array elements must have the same type

```
int i[10];   /* the array has 10 elements */
float f[20];
char c[30];
```

- We number array elements from 0

```
int i[10] = {0,1,2,3,4,5,6,7,8,9};
        /* i[0]=0, i[1]=1, ..., i[9]=9 */
```

- How about this

```
int[] i = {0,1,2,3,4,5,6,7,8,9};
```

Arrays

- An Array is a collection of data items
- All the array elements must have the same type

```
int i[10];   /* the array has 10 elements */
float f[20];
char c[30];
```

- We number array elements from 0

```
int i[10] = {0,1,2,3,4,5,6,7,8,9};
        /* i[0]=0, i[1]=1, ..., i[9]=9 */
```

- How about this

```
int[] i = {0,1,2,3,4,5,6,7,8,9};
        /* I know you did Java.... */
```

Arrays

- Initialisation

```
int i[10];
i[10] = {0,1,2,3,4,5,6,7,8,9};


int i[10] = {0,1,2,3,4,5,6,7,8,9};

float f[20] = {1.7,2.0,5.9,31.2, …};

char c[30] = {‘a’, ’b’, ‘c’, ‘d’, …};

int a[10] = b[10];
```

Arrays

- Initialisation
  - Arrays can be initialised but cannot be assigned

```
int i[10];
i[10] = {0,1,2,3,4,5,6,7,8,9};
/* assignment – this is wrong */

int i[10] = {0,1,2,3,4,5,6,7,8,9};

float f[20] = {1.7,2.0,5.9,31.2, …};

char c[30] = {‘a’, ’b’, ‘c’, ‘d’, …};

int a[10] = b[10];
```

Arrays

- Initialisation
  - Arrays can be initialised but cannot be assigned

```
int i[10];
i[10] = {0,1,2,3,4,5,6,7,8,9};
/* assignment – this is wrong */

int i[10] = {0,1,2,3,4,5,6,7,8,9};

float f[20] = {1.7,2.0,5.9,31.2, …};

char c[30] = {‘a’, ’b’, ‘c’, ‘d’, …};
```

  - We cannot initialise an array using another array

```
int a[10] = b[10]; /* this is wrong */
```

Arrays

- Does Java do bound checking?
- Does C++ do bound checking?
- Does C do bound checking?

Arrays

- C does not do bound checking
  - An example

```
#define SIZE 4

int main(void)
{ int i, x[SIZE];

  for (i = 0; i<2*SIZE; i++)
    x[i] = i;
  return 0;
}
```

Arrays

- C does not do bound checking
  - An example (segmentation fault)

```
/* bad memory access */
/* segmentation fault */

#define SIZE 4

int main(void)
{ int i, x[SIZE]; /* x has 4 elements */

  for (i = 0; i<2*SIZE; i++)
    x[i] = i;     /* x has 8 elements */
  return 0;
}
```

Arrays

- Another example

```
#define SIZE 4

int main(void)
{ int i, x[SIZE];

  for (i = 0; i<=SIZE; i++)
    x[i] = i;
  return 0;
}
```

Arrays

– Another example (no segmentation fault)

```
/* bad memory access */
/* no segmentation fault */

#define SIZE 4

int main(void)
{ int i, x[SIZE];

  for (i = 0; i<=SIZE; i++)
    x[i] = i;
  return 0;
}
```

Arrays

– One more example

```
#define SIZE 4

int main(void)
{ int i, x[SIZE];
  int y=66, z=99;

  for (i = 0; i<SIZE+3; i++)
    x[i] = i;
  return 0;
}
```

Arrays

– One more example (change values by accident)

```
/* bad memory access */
/* change the values of other variables */

#define SIZE 4

int main(void)
{ int i, x[SIZE];
  int y=66, z=99;

  for (i = 0; i<SIZE+3; i++)
    x[i] = i;
  return 0;
}
```

Pointers

- Every variable occupies a memory block

```
char c; /* sizeof(c) = 1 byte */
int i;  /* sizeof(i) = 4 bytes */
```

- Each occupied block has an address

```
  /* c’s memory address gets printed */

  /* i’s memory address gets printed */
```

Pointers

- Every variable occupies a memory block

```
char c; /* sizeof(c) = 1 byte */
int i;  /* sizeof(i) = 4 bytes */
```

- Each occupied block has an address

```
printf(“&c=%x”, &c);
   /* c’s memory address gets printed */
printf(“&i=%x”, &i);
   /* i’s memory address gets printed */
```

- Can we use a variable to store c’s or i’s address?

Pointers

- Every variable occupies a memory block

```
char c; /* sizeof(c) = 1 byte */
int i;  /* sizeof(i) = 4 bytes */
```

- Each occupied block has an address

```
printf(“&c=%x”, &c);
   /* c’s memory address gets printed */
printf(“&i=%x”, &i);
   /* i’s memory address gets printed */
```

- Can we use a variable to store c’s or i’s address?

```
char *ptrc; ptrc = &c;
int *ptri; ptri = &i;
```

Pointers

- Can we use a variable to store c’s or i’s address?

```
char *ptrc = &c;
/* char *ptrc; ptrc=&c; */

int *ptri = &i;
/* int *ptri; ptri=&i; */
```

- ptrc and ptri are called pointers
  - A pointer is used to store the address of another variable
  - Pointers allow a programmer to play with memory addresses
    - Access to memory to do powerful things (dynamic data structures)
    - Access to memory that does not belong to you

Pointers

- To declare a pointer

```
char *pc;
```

Pointers

- To declare a pointer

```
char *pc;
/* pc (NOT *pc) is a pointer that points to a char.
 * Or, pc WILL be used to store some memory
 * address. The memory at that address is
 * expected to store a char.
 */

/* pc points to a “virtual” char */
```

Pointers

- Let pc point to a real char

```
char c = ‘A’;
```

  – Version 1

```
char *pc = &c;     /* pc points to c */
```

  – Version 2

```
char *pc;
pc = &c;           /* pc stores &c */
```

  – If we want to know the value stored in the memory that pc points to (that is, the value of c), we can do this:

Pointers

- Let pc point to a real char

```
char c = ‘A’;
```

  – Version 1

```
char *pc = &c;     /* pc points to c */
```

  – Version 2

```
char *pc;
pc = &c;           /* pc stores &c */
```

  – If we want to know the value stored in the memory that pc points to (that is, the value of c), we can do this:

```
printf("c=%d\n", *pc);      /* output? */
```

    • * is called dereference operator
    • *pc means dereference pointer pc
    • *pc gives us the variable pc points to

Pointers

- Let pc point to a real char
  – A simple example

```
char c = 'A';
char *pc = &c;
```

A	bfbfe8e3
bfbfe8e3	bfbfe8dc

```
printf("c=%c, &c=%x\n", c, &c);
/* c= , &c= */

printf("pc=%x, &pc=%x\n", pc, &pc);
/* pc= , &pc= */
```

Pointers

- Let pc point to a real char
  - A simple example

```
char c = 'A';
char *pc = &c;
```

A	bfbfe8e3
bfbfe8e3	bfbfe8dc

```
printf("c=%c, &c=%x\n", c, &c);
/* c=A, &c=bfbfe8e3 */

printf("pc=%x, &pc=%x\n", pc, &pc);
/* pc=bfbfe8e3, &pc=bfbfe8dc */
```

Pointers

- Let pc point to a real char
  - A simple example

```
char c = 'A';
char *pc = &c;
```

A	bfbfe8e3
bfbfe8e3	bfbfe8dc

```
printf("c=%c, &c=%x\n", c, &c);
/* c=A, &c=bfbfe8e3 */

printf("*pc=%c\n", *pc);
/* *pc=A */
```

Pointers

- Let pc point to a real char
  - A simple example

```
char c = 'A';
char *pc = &c;
```

A	bfbfe8e3
bfbfe8e3	bfbfe8dc

```
printf("c=%c, &c=%x\n", c, &c);
/* c=A, &c=bfbfe8e3 */

printf("*&pc=%x, &*pc=%x\n", *&pc, &*pc);
/* *&pc=???, &*pc=??? */
```

Pointers

- Let pc point to a real char
  - A simple example

```
char c = 'A';
char *pc = &c;
```

A	bfbfe8e3
bfbfe8e3	bfbfe8dc

```
printf("c=%c, &c=%x\n", c, &c);
/* c=A, &c=bfbfe8e3 */

printf("pc=%x, &c=%x\n", *&pc, &*pc);
/* pc=bfbfe8e3, &c=bfbfe8e3 */
```

Pointers

- Be aware …

```
char c = ‘A’, d = ‘B’;

/* char *pc;
 * *pc = &c;          /* is this correct? */
 * char *pc = c;      /* is this correct? */
 */
```

Pointers

- Be aware …

```
char c = ‘A’, d = ‘B’;

/* char *pc;
 * *pc = &c;          /* wrong! */
 * char *pc = c;      /* wrong! */
 */


char *pc = &c;
*pc = d;

/* What is c’s value now ... */
```

Pointers

- Let pi point to an int

```
int i = 65;
int *pi = &i;     /* pi points to i */
```
or
```
int *pi;
pi = &i;          /* pi stores &i */
```
  – How about this
```
pi = i;
pi = 0;
pi = NULL;
pi = (int *)238435;
```

Pointers

- Let pi point to an int

```
int i = 65;
int *pi = &i;     /* pi points to i */
```
or
```
int *pi;
pi = &i;          /* pi stores &i */
```
  – How about this
```
pi = i; /*wrong, reason mentioned earlier*/
pi = 0; /*special case/the only exception*/
pi = NULL; /* same to pi = 0. use
        /* NULL instead of 0 in practice*/
pi = (int *)238435;
        /* an absolute address in memory */
```

Pointers

- Pointer's size: all pointer types have the same size (check it out by yourself)

char * size = 4	char size = 1
int * size = 4	int size = 4
double * size = 4	double size = 8

Next Lecture

- How arrays relate to pointers?