# NWEN 241
# Arrays and Pointers II

Qiang Fu

School of Engineering and Computer Science
Victoria University of Wellington

## This Lecture

- How arrays relate to pointers?



## How Arrays Relate to Pointers

- An array name is actually a pointer (not exactly)

```
int i[10];
/* i is a pointer to int */
/* i is &i[0], (i+1) is &i[1], ..., */
/* (i+8) is &i[8], (i+9) is &i[9] */
```



## How Arrays Relate to Pointers

- An array name is actually a pointer (not exactly)

```
int i[10];
/* i is a pointer to int */
/* i is &i[0], (i+1) is &i[1], ..., */
/* (i+8) is &i[8], (i+9) is &i[9] */
```

  - i or &i[0] is the base address of the array
  - *i is equivalent to i[0]
  - *(i+1) is equivalent to i[1], ...

## How Arrays Relate to Pointers

- An example (print an array in reverse order)

```
#define SIZE 10

void rprint(int a[]);  /* pass an array as an argument */

int main(void)
{ int i, x[SIZE];      /* x[] has 10 int elements */

  for (i=0; i<SIZE; i++)
    x[i] = i;          /* assign i to x[i] */

  rprint(x);           /* call for reverse printing */
  return 0;
}

void rprint(int a[])    /* pass by value??? */
{ int i;
  for (i=SIZE-1; i>-1; i--)   /* i starts with 9 */
    printf("x[%d]=%d, &x[%d]=%x\n", i, a[i], i, &a[i]);
}
```



## How Arrays Relate to Pointers

- An example (print an array in reverse order)

```
#define SIZE 10

void rprint(int *);    /* pass as a pointer */

int main(void)
{ int i, x[SIZE];      /* x[] has 10 int elements */

  for (i=0; i<SIZE; i++)
    x[i] = i;          /* assign i to x[i] */

  rprint(x);           /* call for reverse printing */
  return 0;
}

void rprint(int *a)    /* pass by address */
{ int i;
  for (i=SIZE-1; i>-1; i--)   /* i starts with 9 */
    printf("x[%d]=%d, &x[%d]=%x\n", i, *(a+i), i, a+i);
}
```



## How Arrays Relate to Pointers

- An example (print an array in reverse order)

```
#define SIZE 10

void rprint(int *);    /* pass as a pointer */

int main(void)
{ int i, x[SIZE];      /* x[] has 10 int elements */

  for (i=0; i<SIZE; i++)
    x[i] = i;          /* assign i to x[i] */

  rprint(x);           /* call for reverse printing */
  return 0;
}

void rprint(int *p)    /* array vs pointer notation */
{ int i;
  for (i=SIZE-1; i>-1; i--)   /* i starts with 9 */
    printf("x[%d]=%d, x[%d]=%d\n", i, *(p+i), i, p[i]);
}
```



## How Arrays Relate to Pointers

- p[i] vs a[i]

```
int a[10] = {0, 1, 2, …, 8, 9};

int *p;

p = a;

/* What are a[2] and p[2]? */


/* How do they work differently? */
```

## How Arrays Relate to Pointers

- p[i] vs a[i]

```
int a[10] = {0, 1, 2, …, 8, 9};

int *p;

p = a;

/* What are a[2] and p[2]? */


/* How do they work differently? */

– a[2]: shift a (base address) by 2, dereference it

– p[2]: get the base address from p, increment it by
  2, dereference it
```



## How Arrays Relate to Pointers

- Another example (exchange element values)



## Strings

- In Java, string is a real object
- In C, strings are one-dimensional char arrays
- There two ways to express a string
  - Use a pointer
  - Use an array

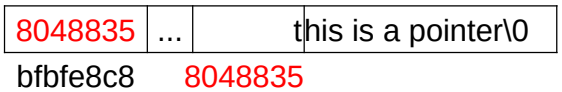


## Strings

- Use a pointer to express a string

```
char *p = “this is a pointer”;
```

In memory,




```
/* a pointer + a string literal */
```

– A string is terminated by null character \0

## Strings

- Use an array to express a string

```
char a[] = “this is an array”;
```

In memory, this is an array\0

```
char a[] = {‘t’, ‘h’, ‘i’, ‘s’, ‘ ’, ‘i’,
  ‘s’, ..., ‘\0’};
```

– A string is terminated by null character \0



## Strings

- An example (copy string)

```
... strcp(..., ...);
int main(void)
{ char source[] = "this is an array";
  char target[] = "this is another array";

  strcp(source, target); /* not strcpy */
}

Tell me the types.
```



## Strings

- An example (copy string)

```
void strcp(char *, char *);
int main(void)
{ char source[] = "this is an array";
  char target[] = "this is another array";

  strcp(source, target); /* not strcpy */
}

void strcp(char *s, char *t)
```



## Strings

- An example (copy string)

```
void strcp(char *, char *);
int main(void)
{ char source[] = "this is an array";
  char target[] = "this is another array";

  strcp(source, target); /* not strcpy */
}

void strcp(char *s, char *t)
{ *t   = *s

}
```

## Strings

- An example (copy string)

```
void strcp(char *, char *);
int main(void)
{ char source[] = "this is an array";
  char target[] = "this is another array";

  strcp(source, target); /* not strcpy */
}

void strcp(char *s, char *t)
{ *t++ = *s++

}
```



## Strings

- An example (copy string)

```
void strcp(char *, char *);
int main(void)
{ char source[] = "this is an array";
  char target[] = "this is another array";

  strcp(source, target); /* not strcpy */
}

void strcp(char *s, char *t)
{ (*t++ = *s++) != '\0'

}
```



## Strings

- An example (copy string)

```
void strcp(char *, char *);
int main(void)
{ char source[] = "this is an array";
  char target[] = "this is another array";

  strcp(source, target); /* not strcpy */
}

void strcp(char *s, char *t)
{ while ((*t++ = *s++) != '\0')
    ;
}
```



## Strings

- An example (copy string)

```
/* if you are careful .... */

void strcp(const char *, char *);

void strcp(const char *s, char *t)
{ while ((*t++ = *s++) != '\0')
    ;
}
```

## Strings

- An example (copy string)
  - Let us do this:

```
char *source = "this is a pointer";
```

  instead of this:

```
char source[] = "this is an array";
```

  this:

```
char *target = "this is another pointer";
```

  instead of this:

```
char target[] = "this is another array";
```

  - Does it work?



## Strings

- An example (copy string)

```
void strcp(char *, char *);
int main(void)
{ char *source = "this is an array";
  char *target = "this is another array";

  strcp(source, target); /* not strcpy */
}
void strcp(char *s, char *t)
{ while ((*t++ = *s++) != '\0')
    ;
}
```

  - Does it work?



## Strings

- An example (copy string)

```
void strcp(char *, char *);
int main(void)
{ char *source = "this is an array";
  char *target = "this is another array";

  strcp(source, target); /* not strcpy */
}
void strcp(char *s, char *t)
{ while ((*t++ = *s++) != '\0')
    ;
}
```

  - System dependent - it may be read-only memory.



## Strings

- p1 vs p2

```
char *p1 = "this is a pointer";

char *p2 = "this is a pointer";

/* Are p1 and p2 associated with */
/* the same string literal? */
```

  - System dependent - it may be read-only memory.

## Various Pointers (What's p?)

- Pointers to pointers

```
char **p;         /* a pointer to ... */
char ***p;        /* a pointer to a pointer ... */
```

- Pointers and arrays

```
char *p[5];       /* an array of ... to char */
char (*p)[5];     /*a pointer to ... */
```

- Pointers and functions

```
char *p(void);          /* a function ... to char */
char *p(int, int);
char (*p)(int, int);    /* a pointer to ... */
char *(*p)(void);
```

- Pointers, arrays and functions

```
char (*(*p[5])(void))(void);
  /* an array of ... that return a pointer to functions
  that return a char */
```



## Next Lecture

- More on arrays and pointers