NWEN 241
Arrays and Pointers IV

Qiang Fu

School of Engineering and Computer Science
Victoria University of Wellington

This Lecture

- Various topics about arrays, pointers and functions
  - Passing arrays to functions
  - Command-line arguments
  - Passing pointers to functions
  - Passing functions to functions
  - Functions and pointers

Passing Arrays to Functions

- We have done a lot of this

```
void rprint(int a[]);
void v_exchange(int a[]);
void strcp(char *, char *);
```

  - In the argument list, an array is treated as a pointer

```
void rprint(int a[]);
   /* is equivalent to */
void rprint(int *);
   /* or */
void rprint(int *a_name);
```


Passing Arrays to Functions

- **base address** vs. **the address of array**

Passing Arrays to Functions

- **base address** vs. **the address of array**

```
int a[5] = {0, 1, 2, 3, 4};
```

a: base address

&a: what is it? Is a here a ptr to int?

Passing Arrays to Functions

- **base address** vs. **the address of array**

```
int a[5] = {0, 1, 2, 3, 4);
```

a: base address

&a: the addr of array a - a is an array, not a ptr.

Recall sizeof(a).

Passing Arrays to Functions

- What would you do if you want to avoid accidently changing the array you just want to print

```
void rprint(int a[]);
   /* is equivalent to */
void rprint(int *);
   /* or */
void rprint(int *a_name);
```


Passing Arrays to Functions

- What would you do if you want to avoid accidently changing the array you just want to print

```
void rprint(const int a[]);
   /* is equivalent to */
void rprint(const int *);
   /* or */
void rprint(const int *a_name);
```

Does the original array have to be `const`?

How about `void rprint(int * const a_name);`

Passing Arrays to Functions

- Passing an array of pointers

```
int *p[10];

/* When you pass p to a function, */
/* what are you passing? */

void func(int *p[]);

Or

void func(int **p);
```


Passing Arrays to Functions

- Passing an array of pointers

```
char *a[7] = {“Mon”, “Tue”, ...};
void func(char *a[]);
void func(char **a);
```

  – In the argument list, an array is treated as a pointer

```
void func(char *a[]);
   /* is equivalent to */
void func(char **a);
   /* or */
void func(char **);
   /* or */
void func(char *[]);
```


Command-line Arguments

- When we run a program, if the `main` has two arguments like this

```
int main(int argc, char *argv[])
{ ...
}
```

- then we can pass command-line arguments to the program.
- argc stands for argument count
  – It tells main the number of command line arguments
- argv stands for argument vector
  – What is it?

Command-line Arguments

- When we run a program, if the `main` has two arguments like this

```
int main(int argc, char *argv[])
{ ...
}
```

- then we can pass command-line arguments to the program.
- argc stands for argument count
  – It tells main the number of command line arguments
- argv stands for argument vector
  – It is an array of pointers to char (the first char of argument names)

Command-line Arguments

- An example (echo the command-line arguments)

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int i;

  for (i=0; i<argc; i++)
    printf("%s ", argv[i]);
  return 0;
}
```


Command-line Arguments

- An example (echo the command-line arguments)

```
#include <stdio.h>

int main(int argc, char **argv)
{
  int i;

  for (i=0; i<argc; i++)
    printf("%s ", argv[i]);
  return 0;
}
```


Command-line Arguments

- An example (echo the command-line arguments)

```
#include <stdio.h>

int main(int argc, char **argv)
{
  int i;

  for (i=0; i<argc; i++)
    printf("%s ", *(argv+i));
  return 0;
}
```


Command-line Arguments

- An example (echo the command-line arguments)

```
#include <stdio.h>

int main(int argc, char **argv)
{
  int i;

  for (i=0; i<argc; i++)
    printf("%s ", (*argv+i));//what's this?
  return 0;
}
```

Command-line Arguments

- An example (echo the command-line arguments)

```
#include <stdio.h>

int main(int argc, char **argv)
{
  int i;

  for (i=0; i<argc; i++)
    printf("%s ", (*argv+i));//wrong
  return 0;
}
```


Passing Pointers to Functions

- Revisit swap(): swap the values of p, q

```
int main(void)
{ ...
  int *ptrp = &p, *ptrq = &q;
  swap(ptrp, ptrq); /*the addresses of p, q*/
  return 0;         /*are passed to swap() */
}
void swap(int *ptrx, int *ptry)
{ int tmp;
  tmp = *ptrx;
  *ptrx = *ptry;    /* the values stored at */
  *ptry = tmp;      /* &p, &q are swapped */
}
```


Passing Pointers to Functions

- Revisit swap(): swap the values of ptrp, ptrq

```
void swap_ptr(int **, int **);
int main(void)
{ ...
  int *ptrp = &p, *ptrq = &q;
  int **ppp = &ptrp, **ppq  = &ptrq;

  swap_ptr(ppp,ppq);/* &ptrp, &ptrq passed */
                    /* to swap_ptr() */
  return 0;
}
```


Passing Pointers to Functions

- Revisit swap(): swap the values of p, q (c: pass by addr)

```
int main(void)
{ ...
  int *ptrp = &p, *ptrq = &q;
  swap(ptrp, ptrq); /*the addresses of p, q*/
  return 0;         /*are passed to swap() */
}
void swap(int *ptrx, int *ptry)
{ int tmp;
  tmp = *ptrx;
  *ptrx = *ptry;    /* the values stored at */
  *ptry = tmp;      /* &p, &q are swapped */
}
```

Passing Pointers to Functions

- Revisit swap(): swap the values of p, q (c++: pass by ref.)

```
int main(void)
{ ...
  int *ptrp = &p, *ptrq = &q;
  swap(p, q);       /* p and q are passed */
  return 0;         /* by reference to swap */
}
void swap(int &x, int &y)
{ int tmp;
  tmp = x;
  x = y;            /* the values stored at */
  y = tmp;          /* &p, &q are swapped */
}
```


Passing Functions to Functions

- Revisit l_minus_s(): the larger() minus the smaller()

```
int main(void)
{ ...
  l_s = l_minus_s(larger, smaller, p,q);
  ...
}

int l_minus_s(int l(int, int), int s(int,
  int), int x, int y)
{ return(l(x,y)-s(x,y));
}
```


Passing Functions to Functions

- Revisit l_minus_s(): function prototype

```
int l_minus_s(int l(int, int), int s(int,
  int), int, int);
```


Passing Functions to Functions

- Revisit l_minus_s(): function prototype

```
int l_minus_s(int l(int, int), int s(int,
  int), int, int);

int l_minus_s(int (*l)(int, int), int
  (*s)(int, int), int, int);

int l_minus_s(int (*)(int, int), int
  (*)(int, int), int, int);
```

Passing Functions to Functions

- Revisit l_minus_s(): function prototype

```
int l_minus_s(int l(int, int), int s(int,
  int), int, int);
```

– In the argument list, a function is treated as a pointer

```
int l_minus_s(int (*l)(int, int), int
  (*s)(int, int), int, int);

int l_minus_s(int (*)(int, int), int
  (*)(int, int), int, int);

        /* are equivalent to the function */
        /* prototype above */
```


Functions and Pointers

```
int (*ptrf)(void);

int (*ptrf)(int, int);

int larger(int, int);
ptrf = larger; // or ptrf = &larger;
```


Functions and Pointers

```
int (*ptrf)(void);

int (*ptrf)(int, int);

int larger(int, int);
ptrf = larger; // or ptrf = &larger;
               // both are correct
```


Functions and Pointers

```
int (*ptrf)(void);

int (*ptrf)(int, int);

int larger(int, int);
ptrf = larger; // or ptrf = &larger;
               // both are correct
               // ptrf(a,b) or (*ptrf)(a,b)?
```

Functions and Pointers

```
int (*ptrf)(void);


int (*ptrf)(int, int);


int larger(int, int);
ptrf = larger; // or ptrf = &larger;
               // both are correct
               // ptrf(a,b) or (*ptrf)(a,b)?
               // both are correct
```


Functions and Pointers

```
int (*ptrf)(void);


int (*ptrf)(int, int);


int larger(int, int);
ptrf = larger; // or ptrf = &larger;
               // both are correct
               // ptrf(a,b) or (*ptrf)(a,b)?
               // both are correct
int *ptrf(void);
```


Functions and Pointers

```
int (*ptrf)(void);
  /* ptrf is a pointer to a function that */
  /* returns an int */

int (*ptrf)(int, int);
  /* ptrf is a pointer to a function that */
  /* has two int arguments and returns an int */

int larger(int, int);
ptrf = larger; /* &larger, ptrf(a,b), (*ptrf)(a,b) */
  /* ptrf is a pointer to larger(). larger() */
  /* has two int arguments and returns an int */

int *ptrf(void);
  /* this is NOT a pointer to a function */
  /* ptrf() returns a ptr that points to an int */
```


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 Week/Lecture

- User defined types