Xinfeng Ye

Room: 589

Office hour: by appointment, after class, or whenever I am around

11-1

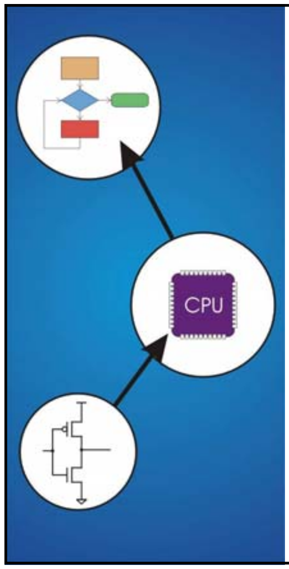
Acknowledgment

These slides are based on the slides provided by McGraw-Hill

11-3

- **Basic C programming**
- **Convert C program statements to LC-3**
- **Cache (how to bridge the speed gap between memory and CPU)**
- **Virtual memory (how to run multiple programs on one machine)**

11-2



Chapter 11 Introduction to Programming in C

Agenda

- An overview of high-level programming language
- Some C program examples

High-Level Language

Gives symbolic names to values

- don't need to know which register or memory location

Provides abstraction of underlying hardware

- operations do not depend on instruction set
- example: can write "a = b * c", even though LC-3 doesn't have a multiply instruction

Provides expressiveness

- use meaningful symbols that convey meaning
- simple expressions for common control patterns (if-then-else)

Enhances code readability

Safeguards against bugs

- can enforce rules or conditions at compile-time or run-time

Compilation vs. Interpretation

Different ways of translating high-level language

Interpretation

- interpreter = program that executes program statements
- generally one line/command at a time
- limited processing
- easy to debug, make changes, view intermediate results
- languages: Perl, Java, Bash

Compilation

- translates statements into machine language
  - ➢ does not execute, but creates executable program
- performs optimization over multiple statements
- change requires recompilation
  - ➢ can be harder to debug, since executed code may be different
- languages: C, C++

Compilation vs. Interpretation

Consider the following algorithm:

- `Get W from the keyboard.`
- `X = W + W`
- `Y = X + X`
- `Z = Y + Y`
- `Print Z to screen.`

If <u>interpreting</u>, how many arithmetic operations occur?

If <u>compiling</u>, we can analyze the entire program and possibly reduce the number of operations. Can we simplify the above algorithm to use a single arithmetic operation?

Compiling a C Program

Entire mechanism is usually called the "compiler"

Preprocessor

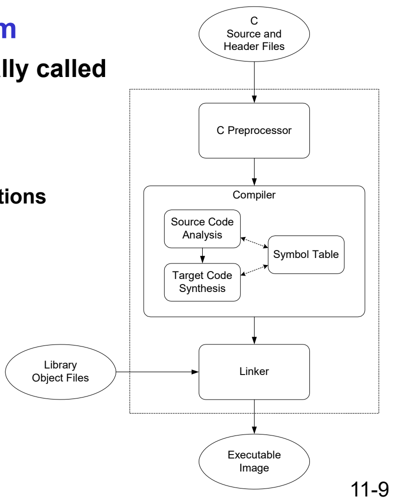
- **macro substitution**
- **"source-level" transformations**
  - ➢ **output is still C**

Compiler

- **generates object file**
  - ➢ **machine instructions**

Linker

- **combine object files (including libraries) into executable image**



11-9

Compiler

Source Code Analysis

- **"front end"**
- **parses programs to identify its pieces**
  - ➢ **variables, expressions, statements, functions, etc.**
- **depends on language (not on target machine)**

Code Generation

- **"back end"**
- **generates machine code from analyzed source**
- **may optimize machine code to make it run more efficiently**
- **dependent on target machine**

Symbol Table

- **map between symbolic names and items**
- **like assembler, but more information**

11-10

Agenda

- **An overview of high-level programming language**
- **Some C program examples**
  - **A counting program**
  - **Passing parameters to functions**
  - **Command line arguments**
  - **Handling files**
  - **Manipulating strings**
  - **Bit operations**

11-11

A Simple C Program

- **Ask user to enter a positive number**
- **Display the numbers from the given number to 0**

```
Enter a positive number: 6
6
5
4
3
2
1
0
```

11-12

A Simple C Program

```
#include <stdio.h>
#define STOP 0

/* Function: main                               */
/* Description: counts down from user input to STOP */
int main()
{
  /* variable declarations */
  int counter;  /* an integer to hold count values */
  int startPoint; /* starting point for countdown */
  /* prompt user for input */
  printf("Enter a positive number: ");
  scanf("%d", &startPoint);  /* read into startPoint */
  /* count down and print count */
  for (counter=startPoint; counter >= STOP; counter--)
    printf("%d\n", counter);
}
```

11-13

Comments

Begins with `/*` and ends with `*/`

or a line starting with //

Can span multiple lines

As before, use comments to help reader, not to confuse or to restate the obvious

11-15

Preprocessor Directives

`#include <stdio.h>`

- Before compiling, copy contents of header file (stdio.h) into source code.
- Header files typically contain descriptions of functions and variables needed by the program.
  - no restrictions -- could be any C source code

`#define STOP 0`

- Before compiling, replace all instances of the string "STOP" with the string "0"
- Called a *macro*
- Used for values that won't change during execution, but might change if the program is reused. (Must recompile.)

11-14

`main` Function

Every C program must have a function called `main()`.

This is the code that is executed when the program is run.

The code for the function lives within brackets:

```
int main()
{
  /* code goes here */
}
```

11-16

Variable Declarations

Variables are used as names for data items.

Each variable has a *type*,
which tells the compiler how the data is to be interpreted
(and how much space it needs, etc.).

```
int counter;
int startPoint;
```

`int` is a predefined integer type in C.

Input and Output

Variety of I/O functions in *C Standard Library*.
Must include `<stdio.h>` to use them.

```
printf("%d\n", counter);
```

- String contains characters to print and formatting directives for variables.
- This call says to print the variable `counter` as a decimal integer, followed by a linefeed (`\n`).

```
scanf("%d", &startPoint);
```

- String contains formatting directives for looking at input.
- This call says to read a decimal integer and assign it to the variable `startPoint`. (Don't worry about the `&` yet.)

More About Output

Can print arbitrary expressions, not just variables

```
printf("%d\n", startPoint - counter);
```

Print multiple expressions with a single statement

```
printf("%d %d\n", counter,
       startPoint - counter);
```

Different formatting options:

`%d`	decimal integer
`%x`	hexadecimal integer
`%c`	ASCII character

Examples

This code:

```
printf("%d is a prime number.\n", 43);
printf("43 plus 59 in decimal is %d.\n", 43+59);
printf("43 plus 59 in hex is %x.\n", 43+59);
printf("43 plus 59 as a character is %c.\n", 43+59);
```

produces this output:

```
43 is a prime number.
43 plus 59 in decimal is 102.
43 plus 59 in hex is 66.
43 plus 59 as a character is f.
```

Examples of Input

Many of the same formatting characters are available for user input.

```
scanf("%c", &nextChar);
```

- **reads a single character and stores it in nextChar**

```
scanf("%f", &radius);
```

- **reads a floating point number and stores it in radius**

```
scanf("%d %d", &length, &width);
```

- **reads two decimal integers (separated by whitespace), stores the first one in length and the second in width**

Must use ampersand (&) for variables being modified.
(Explained in Chapter 16.)

11-21

A Simple C Program

- **Ask user to enter a positive number**
- **Display the numbers from the given number to 0**

```
Enter a positive number: 6
6
5
4
3
2
1
0
```

11-22

A Simple C Program

```
/* Function: main                                 */
/* Description: counts down from user input to STOP */

int main()
{




}
```

A program must have a main function

11-23

A Simple C Program

```
/* Function: main                                 */
/* Description: counts down from user input to STOP */
int main()
{
  /* variable declarations */
  int counter;   /* an integer to hold count values */
  int startPoint; /* starting point for countdown */



}
```

11-24

A Simple C Program

```
#define STOP 0
```

Value for stop

```
/* Function: main                                */
/* Description: counts down from user input to STOP */
int main()
{
  /* variable declarations */
  int counter;  /* an integer to hold count values */
  int startPoint; /* starting point for countdown */




}
```

11-25

A Simple C Program

```
#include <stdio.h>
#define STOP 0

/* Function: main                                */
/* Description: counts down from user input to STOP */
int main()
{
  /* variable declarations */
  int counter;  /* an integer to hold count values */
  int startPoint; /* starting point for countdown */
  /* prompt user for input */
  printf("Enter a positive number: ");



}
```

11-26

A Simple C Program

```
#include <stdio.h>
#define STOP 0

/* Function: main                                */
/* Description: counts down from user input to STOP */
int main()
{
  /* variable declarations */
  int counter;  /* an integer to hold count values */
  int startPoint; /* starting point for countdown */
  /* prompt user for input */
  printf("Enter a positive number: ");
  scanf("%d", &startPoint);  /* read into startPoint */


}
```

11-27

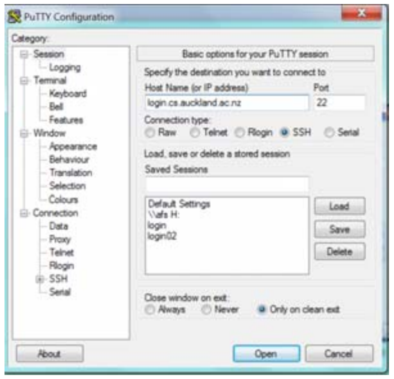
A Simple C Program

```
#include <stdio.h>
#define STOP 0

/* Function: main                                */
/* Description: counts down from user input to STOP */
int main()
{
  /* variable declarations */
  int counter;  /* an integer to hold count values */
  int startPoint; /* starting point for countdown */
  /* prompt user for input */
  printf("Enter a positive number: ");
  scanf("%d", &startPoint);  /* read into startPoint */
  /* count down and print count */
  for (counter=startPoint; counter >= STOP; counter--)
    printf("%d\n", counter);
}
```

11-28

Using University UNIX Account

- Connect from MS Windows using Putty
- All programs → Putty
- Under-grad server is login.cs.auckland.ac.nz
- Login using your UPI and password
- You can access the machine from home if you have Putty installed on your home machine


PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): login.cs.auckland.ac.nz

Port: 22

Connection type: ( ) Raw ( ) Telnet ( ) Rlogin [✓ SSH] ( ) Serial

Load, save or delete a stored session

Saved Sessions: Default Settings, \\afs H:, login, login02

Close window on exit: ( ) Always ( ) Never [✓ Only on clean exit]

Compile and Run the Program

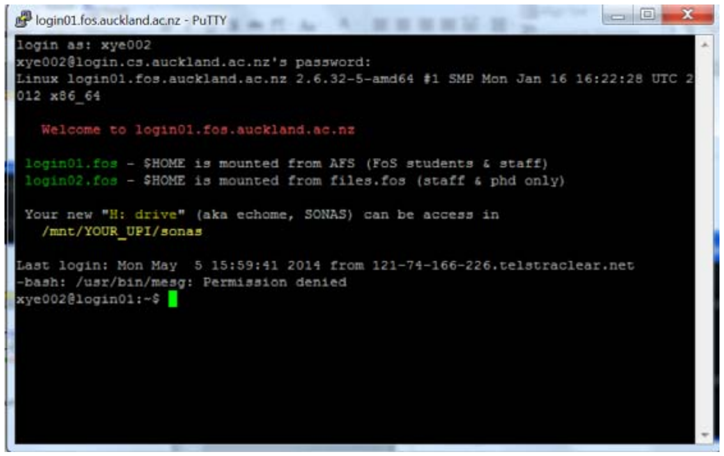
- **Compile**

  gcc -o countdown countdown.c

  name of executatble

  name of source code
- **Run**

```
$ ./countdown
Enter a positive number: 6
6
5
4
3
2
1
0
```


login01.fos.auckland.ac.nz - PuTTY

```
login as: xye002
xye002@login.cs.auckland.ac.nz's password:
Linux login01.fos.auckland.ac.nz 2.6.32-5-amd64 #1 SMP Mon Jan 16 16:22:28 UTC 2
012 x86_64

   Welcome to login01.fos.auckland.ac.nz

 login01.fos - $HOME is mounted from AFS (FoS students & staff)
 login02.fos - $HOME is mounted from files.fos (staff & phd only)

 Your new "H: drive" (aka echome, SONAS) can be access in
   /mnt/YOUR_UPI/sonas

Last login: Mon May  5 15:59:41 2014 from 121-74-166-226.telstraclear.net
-bash: /usr/bin/mesg: Permission denied
xye002@login01:~$
```

Compiling and Linking

Various compilers available

- **gcc**
- **includes preprocessor, compiler, and linker**

- **'gcc -S countdown.c' will create no executable but simply an assembly file 'countdown.s'. See if you are able to understand the generated assembly.**

- You can download examples from Canvas under "file" panel (examples.tar.gz).
- Unpack the file
  - tar xvf examples.tar.gz
- Compile and run the programs

11-33

Pointers

- A pointer is the address of an object. If you have an integer called 'n' then, '&n' will give you the address of 'n'.
- A pointer type is declared with a '*' suffix attached to the type. An 'int' is an integer, and an 'int*' is a pointer to an integer.

```
int n = 36;
int* an = &n;          // address of 'n' or "pointer" to 'n'
*an = 32;      // set the contents of 'an' to 32 - this is equivalent
               // to saying 'n = 32', since 'an' is pointing to 'n'
```

11-35

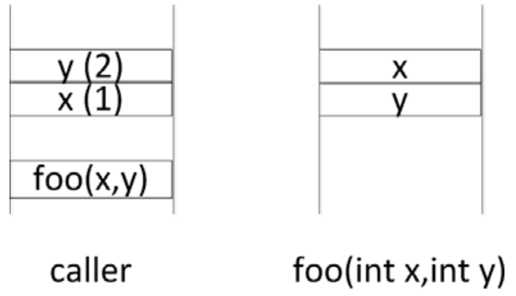
Agenda

- An overview of high-level programming language
- Some C program examples
  - A counting program
  - Passing parameters to functions
  - Command line arguments
  - Handling files
  - Manipulating strings
  - Bit operations

11-34

Passing parameters to a function

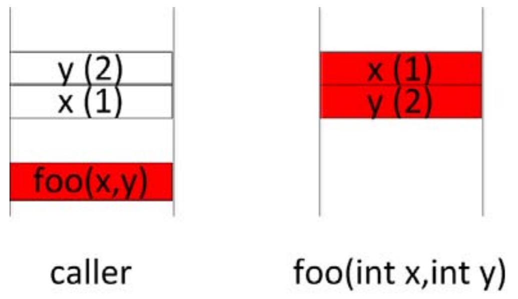
- Call-by-value: the caller passes a value to the function. The function holds its own copy of the parameter. If the function changes the value of the parameter in the function, the caller does not see the change.



11-36

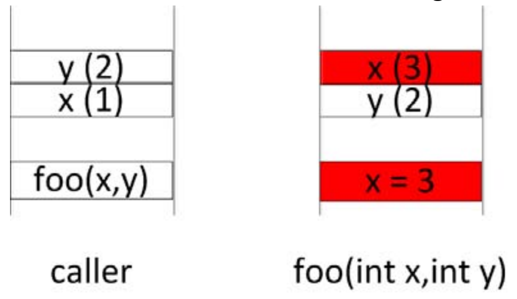
Passing parameters to a function

- **Call-by-value: the caller passes a value to the function. The function holds its own copy of the parameter. If the function changes the value of the parameter in the function, the caller does not see the change.**



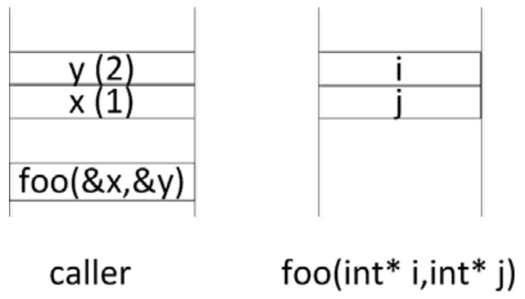
Passing parameters to a function

- **Call-by-value: the caller passes a value to the function. The function holds its own copy of the parameter. If the function changes the value of the parameter in the function, the caller does not see the change.**



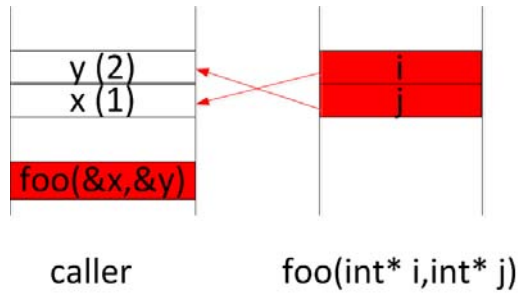
Passing parameters to a function

- **Call-by-reference: the caller passes the address of a variable to the function. The caller and the function operates on the same variable. If the function changes the value of the variable the caller can see the changes.**



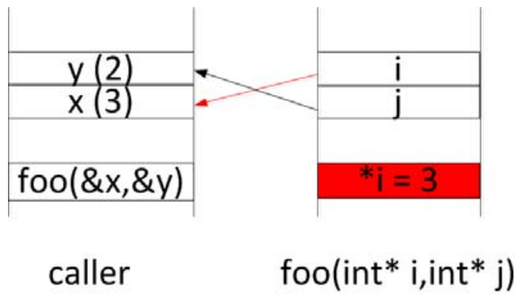
Passing parameters to a function

- **Call-by-reference: the caller passes the address of a variable to the function. The caller and the function operates on the same variable. If the function changes the value of the variable the caller can see the changes.**



Passing parameters to a function

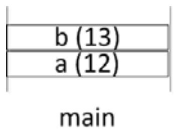
- **Call-by-reference: the caller passes the address of a variable to the function. The caller and the function operates on the same variable. If the function changes the value of the variable the caller can see the changes.**



Call-by-value example

Swap the values of two variables using a function

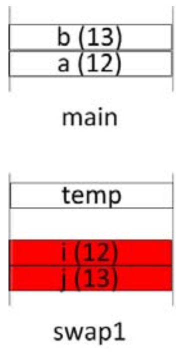
```
int main() {
  int a = 12, b = 13;
  printf("Initial values:\n");
  printf("a = %d  b = %d\n\n", a, b);
  swap1(a, b);
  printf("After swap1:\n");
  printf("a = %d  b = %d\n\n", a, b);
}
```



Call-by-value example

```
int main() {
  int a = 12, b = 13;
  printf("Initial values:\n");
  printf("a = %d  b = %d\n\n", a, b);
  swap1(a, b);
  printf("After swap1:\n");
  printf("a = %d  b = %d\n\n", a, b);
}
```

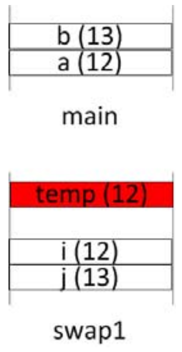
```
void swap1(int i, int j) {
  int temp;
  temp = i;
  i = j;
  j = temp;
}
```



Call-by-value example

```
int main() {
  int a = 12, b = 13;
  printf("Initial values:\n");
  printf("a = %d  b = %d\n\n", a, b);
  swap1(a, b);
  printf("After swap1:\n");
  printf("a = %d  b = %d\n\n", a, b);
}
```

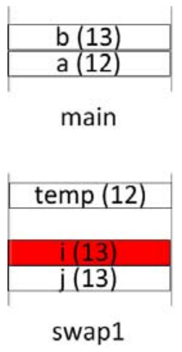
```
void swap1(int i, int j) {
  int temp;
  temp = i;
  i = j;
  j = temp;
}
```



Call-by-value example

```
int main() {
  int a = 12, b = 13;
  printf("Initial values:\n");
  printf("a = %d  b = %d\n\n", a, b);
  swap1(a, b);
  printf("After swap1:\n");
  printf("a = %d  b = %d\n\n", a, b);
}
```

```
void swap1(int i, int j) {
  int temp;
  temp = i;
  i = j;
  j = temp;
}
```

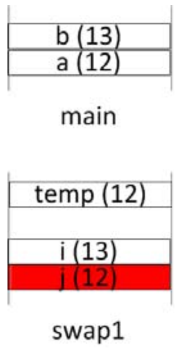


11-45

Call-by-value example

```
int main() {
  int a = 12, b = 13;
  printf("Initial values:\n");
  printf("a = %d  b = %d\n\n", a, b);
  swap1(a, b);
  printf("After swap1:\n");
  printf("a = %d  b = %d\n\n", a, b);
}
```

```
void swap1(int i, int j) {
  int temp;
  temp = i;
  i = j;
  j = temp;
}
```



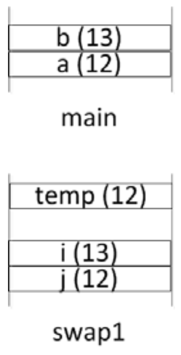
11-46

Call-by-value example

```
int main() {
  int a = 12, b = 13;
  printf("Initial values:\n");
  printf("a = %d  b = %d\n\n", a, b);
  swap1(a, b);
  printf("After swap1:\n");
  printf("a = %d  b = %d\n\n", a, b);
}
```

```
void swap1(int i, int j) {
  int temp;
  temp = i;
  i = j;
  j = temp;
}
```

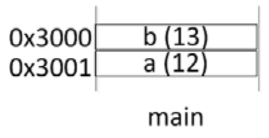
```
Initial values:
a = 12  b = 13

After swap1:
a = 12  b = 13
```



11-47

Call-by-reference example

```
int main() {
  int a = 12, b = 13;
  printf("Initial values:\n");
  printf("a = %d  b = %d\n\n", a, b);
  swap2(&a, &b);
  printf("After swap2:\n");
  printf("a = %d  b = %d\n", a, b);
}
```

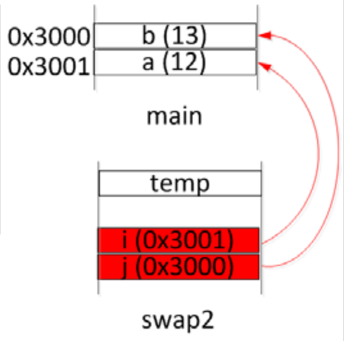


11-48

Call-by-reference example

```
int main() {
  int a = 12, b = 13;
  printf("Initial values:\n");
  printf("a = %d  b = %d\n\n", a, b);
  swap2(&a, &b);
  printf("After swap2:\n");
  printf("a = %d  b = %d\n", a, b);
}
```

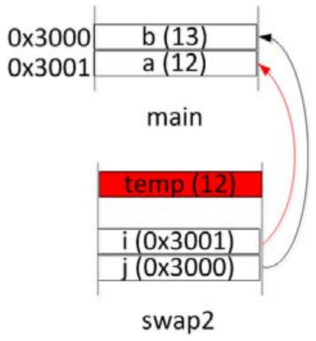
```
void swap2(int* i, int* j) {
  int temp;
  temp = *i;
  *i = *j;
  *j = temp;
}
```



Call-by-reference example

```
int main() {
  int a = 12, b = 13;
  printf("Initial values:\n");
  printf("a = %d  b = %d\n\n", a, b);
  swap2(&a, &b);
  printf("After swap2:\n");
  printf("a = %d  b = %d\n", a, b);
}
```

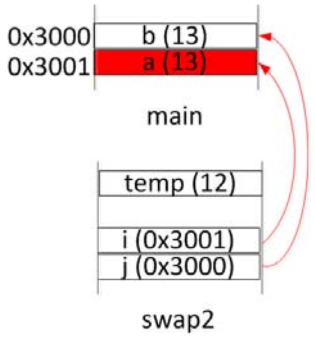
```
void swap2(int* i, int* j) {
  int temp;
  temp = *i;
  *i = *j;
  *j = temp;
}
```



Call-by-reference example

```
int main() {
  int a = 12, b = 13;
  printf("Initial values:\n");
  printf("a = %d  b = %d\n\n", a, b);
  swap2(&a, &b);
  printf("After swap2:\n");
  printf("a = %d  b = %d\n", a, b);
}
```

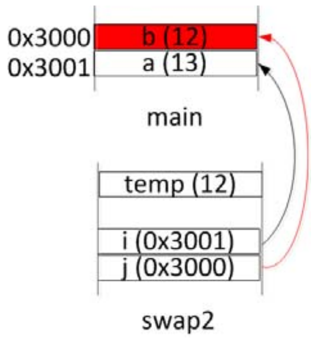
```
void swap2(int* i, int* j) {
  int temp;
  temp = *i;
  *i = *j;
  *j = temp;
}
```



Call-by-reference example

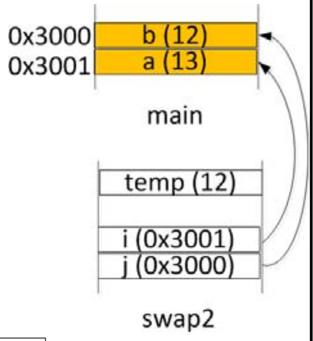
```
int main() {
  int a = 12, b = 13;
  printf("Initial values:\n");
  printf("a = %d  b = %d\n\n", a, b);
  swap2(&a, &b);
  printf("After swap2:\n");
  printf("a = %d  b = %d\n", a, b);
}
```

```
void swap2(int* i, int* j) {
  int temp;
  temp = *i;
  *i = *j;
  *j = temp;
}
```



Call-by-reference example

```
int main() {
  int a = 12, b = 13;
  printf("Initial values:\n");
  printf("a = %d  b = %d\n\n", a, b);
  swap2(&a, &b);
  printf("After swap2:\n");
  printf("a = %d  b = %d\n", a, b);
}
```



```
void swap2(int* i, int* j) {
  int temp;
  temp = *i;
  *i = *j;
  *j = temp;
}
```

```
Initial values:
a = 12  b = 13

After swap2:
a = 13  b = 12
```

A Simple C Program

```
#include <stdio.h>
#define STOP 0

/* Function: main                                  */
/* Description: counts down from user input to STOP */
main()
{
  /* variable declarations */
  int counter;  /* an integer to hold count values */
  int startPoint; /* starting point for countdown */
  /* prompt user for input */
  printf("Enter a positive number: ");
  scanf("%d", &startPoint);  /* read into startPoint */
  /* count down and print count */
  for (counter=startPoint; counter >= STOP; counter--)
    printf("%d\n", counter);
}
```

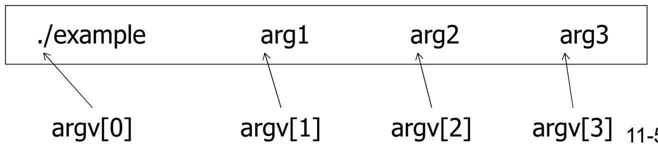
Agenda

- An overview of high-level programming language
- Some C program examples
  - A counting program
  - Passing parameters to functions
  - Command line arguments
  - Handling files
  - Manipulating strings
  - Bit operations

Command line arguments

int main (int argc, char *argv[])

- **argc is the argument count. It is the number of arguments passed into the program from the command line, including the name of the program.**
- **argv[] is the listing of all the arguments.**
  - **Each element of argv[] is a pointer that holds the address of the first character of an argument.**
  - **argv[0] points to the name of the program.**
  - **argv[1] points to the first argument, argv[2] points to the second argument, …**



An example of handling command line arguments

The program takes two integers as inputs and prints out the sum of the two integers.

```
$ ./sum
usage: ./sum operand1 operand2
$ ./sum 12 34
12 + 34 = 46
```

11-57

```
#include <stdio.h>

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

The main function has parameters for holding command line arguments

11-58

```
#include <stdio.h>

int main(int argc, char *argv[]) {
  if (argc != 3)
    printf("usage: %s operand1 operand2\n", argv[0]);
```

argc is the number of arguments

```
$ ./sum
usage: ./sum operand1 operand2
```

11-59

```
#include <stdio.h>

int main(int argc, char *argv[]) {
  if (argc != 3)
    printf("usage: %s operand1 operand2\n", argv[0]);
  else {
    long operand1, operand2;
    char *ptr;




  }
}
```

11-60

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  if (argc != 3)
    printf("usage: %s operand1 operand2\n", argv[0]);
  else {
    long operand1, operand2;
    char *ptr;
    operand1 = strtol(argv[1], &ptr, 10);
    operand2 = strtol(argv[2], &ptr, 10);


  }
}         convert string to number
```

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  if (argc != 3)
    printf("usage: %s operand1 operand2\n", argv[0]);
  else {
    long operand1, operand2;
    char *ptr;
    operand1 = strtol(argv[1], &ptr, 10);
    operand2 = strtol(argv[2], &ptr, 10);
    printf("%d + %d = %d\n", operand1, operand2,
                                   operand1+operand2);
  }
}
```

Agenda

- An overview of high-level programming language
- Some C program examples
  - A counting program
  - Passing parameters to functions
  - Command line arguments
  - Handling files
  - Manipulating strings
  - Bit operations

An example on file IO

- The program reads each line of a file and displays the line.
- The file consists of several rows.
  - Each row has three fields.
  - The fields are separated by a space character.
  - Two fields are strings while one is an integer.

```
Goofy BE 105
Jerry BA 65
Mickey BE 101
Minnie BE 120
Pooh BSc 100
Tigger BSc 70
Tom BA 75
```

```
int main() {
  FILE *file;
  char name[10], degree[5];
  int mark;
```

A string is an array of type char

```
#include <stdio.h>

int main() {
  FILE *file;
  char name[10], degree[5];
  int mark;
  file = fopen("fileIO", "r");
```

Open a file for read

```
#include <stdio.h>

int main() {
  FILE *file;
  char name[10], degree[5];
  int mark;
  file = fopen("fileIO", "r");
  while
    (fscanf(file, "%s %s %d", name, degree, &mark)!= EOF)
    printf("%s %s %d\n", name, degree, mark);

}
```

```
#include <stdio.h>

int main() {
  FILE *file;
  char name[10], degree[5];
  int mark;
  file = fopen("fileIO", "r");
  while
    (fscanf(file, "%s %s %d", name, degree, &mark)!= EOF)
    printf("%s %s %d\n", name, degree, mark);
  fclose(file);
}
```

More example on file IO

- **The program reads each line of a file, counts the number of characters in each line (excluding the new line character), prints each line and the number of characters in the line**

```
Mickey,BCom,101
Tigger,BSc,70
Goofy,BE,105
Jerry,BA,65
Tom,BA,75
```

```
Mickey,BCom,101  15
Tigger,BSc,70  13
Goofy,BE,105  12
Jerry,BA,65  11
Tom,BA,75  9
```

```
#include <stdio.h>


int main() {
  FILE *file;
  char line[20];
  int counter=0;
  file = fopen("fileIO2", "r");
```

The contents of the memory are not initialised. They could be anything

```
#include <stdio.h>
#include <stdbool.h>


int main() {
  FILE *file;
  char line[20];
  int counter=0;
  file = fopen("fileIO2", "r");
  do {
    // process each line
  } while (true);
```

```
#include <stdio.h>
#include <stdbool.h>


int main() {
  FILE *file;
  char line[20];
  int counter=0;
  file = fopen("fileIO2", "r");
  do {
    char c = fgetc(file);
    if (c == EOF) break;


  } while (true);
```

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

int main() {
  FILE *file;
  char line[20];
  int counter=0;
  file = fopen("fileIO2", "r");
  do {
    char c = fgetc(file);
    if (c == EOF) break;
    memset(line, 0, 20);

  } while (true);
```

Set the memory to 0s

11-73

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

int main() {
  FILE *file;
  char line[20];
  int counter=0;
  file = fopen("fileIO2", "r");
  do {
    char c = fgetc(file);
    if (c == EOF) break;
    memset(line, 0, 20);
    while ((c != '\n')&&(c != EOF)) {
      // read the characters in one line

    }

  } while (true);
```

11-74

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

int main() {
  FILE *file;
  char line[20];
  int counter=0;
  file = fopen("fileIO2", "r");
  do {
    char c = fgetc(file);
    if (c == EOF) break;
    memset(line, 0, 20);
    while ((c != '\n')&&(c != EOF)) {
      line[counter++] = c;
      c = fgetc(file);
    }

  } while (true);
```

11-75

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
int main() {
  FILE *file;
  char line[20];
  int counter=0;
  file = fopen("fileIO2", "r");
  do {
    char c = fgetc(file);
    if (c == EOF) break;
    memset(line, 0, 20);
    while ((c != '\n')&&(c != EOF)) {
      line[counter++] = c;
      c = fgetc(file);
    }
    printf("%s  %d\n", line, counter);
    counter=0;
  } while (true);

}
```

11-76

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
int main() {
  FILE *file;
  char line[20];
  int counter=0;
  file = fopen("fileIO2", "r");
  do {
    char c = fgetc(file);
    if (c == EOF) break;
    memset(line, 0, 20);
    while ((c != '\n')&&(c != EOF)) {
      line[counter++] = c;
      c = fgetc(file);
    }
    printf("%s  %d\n", line, counter);
    counter=0;
  } while (true);
  fclose(file);
}
```

Agenda

- An overview of high-level programming language
- Some C program examples
  - A counting program
  - Passing parameters to functions
  - Command line arguments
  - Handling files
  - Manipulating strings
  - Bit operations

Break a string into tokens

The program breaks a string into tokens according to the specified delimiters, and prints out each of the tokens.

```
String
This    is-an|example-program.
```

```
delimiters
 -|.
```

```
This


is
an
example
program
```

```
int main()
{
  char str[80];
  const char delimiters[5] = " -|.";
  char *token, *pos;


}
```

```
#include <string.h>


int main()
{
  char str[80];
  const char delimiters[5] = " -|.";
  char *token, *pos;
  strcpy(str, "This   is-an|example-program.");

}
```

Copy a string

There are many other useful functions defined in <string.h>, e.g. strcmp, strstr, strcat. Have a look of them when doing your assignment.

11-81

```
#include <string.h>


int main()
{
  char str[80];
  const char delimiters[5] = " -|.";
  char *token, *pos;
  strcpy(str, "This is-an|example-program.");
  pos = str;
  token = strsep(&pos, delimiters);

}
```

Extract the first token

pos is updated to point to the character that follows the first delimiter

11-82

```
#include <string.h>
#include <stdio.h>

int main()
{
  char str[80];
  const char delimiters[5] = " -|.";
  char *token, *pos;
  strcpy(str, "This is-an|example-program.");
  pos = str;
  token = strsep(&pos, delimiters);
  while( token != NULL ) {
    printf( " %s\n", token );
    token = strsep(&pos, delimiters);
  }
}
```

11-83

Agenda

- **An overview of high-level programming language**
- **Some C program examples**
  - **A counting program**
  - **Passing parameters to functions**
  - **Command line arguments**
  - **Handling files**
  - **Manipulating strings**
  - **Bit operations**

11-84

Bitwise Operations

Operation	Operator	Examples
AND	&	a & b
OR	|	a | b
NOT	~	~a
XOR	^	a ^ b
LEFT SHIFT	<<	a << 2
RIGHT SHIFT	>>	a >> 2

11-85

Bitwise operations apply the operations to the bits of variables

#	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A	1	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0
B	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0	0

11-86

Bitwise Operations (AND): A & B

#	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A	1	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0
B	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0	0

11-87

#	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A	1	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0
B	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0	0
	1										1	1				

11-88

Mask out value (preserve interesting value)

X AND 1 = X

Find out the value of bit 0 to 3 of variable A
B = A & 0xf

How can we preserve the value of bit 4 of variable A?

11-89

Bitwise Operations (OR): A | B

#	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A	1	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0
B	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0	0

11-90

#	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A	1	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0
B	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0	0
					0			0	0							0

11-91

Set value for a variable A (assume A = 0)

X OR 1 = 1

Set bit 2 of A to 1
A = A | 0x4

How can we set the value of bit 4 and 5 of A to 1?

11-92

Bitwise Operations (NOT): ~A

#	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A	1	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0

#	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A	1	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0
	0	0	1	0	1	1	0	1	1	1	0	0	0	1	0	1

Bitwise Operations: A << n

#	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A	1	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0

Bitwise Operations: A >> n

#	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A	1	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0

Bitwise Operations (XOR): A ^ B

#	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A	1	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0
B	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0	0

11-97

#	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A	1	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0
B	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0	0
		1	1	1		1	1			1			1		1	

11-98

0 XOR 0 =

1 XOR 0 =

11-99

- 0 XOR 0 = 0
- 1 XOR 0 = 1

X XOR 0 = X

11-100

XOR has associative property

(X XOR Y) XOR Z = X XOR (Y XOR Z)

11-101

XOR has associative property

(X XOR Y) XOR Z = X XOR (Y XOR Z)

If Y = Z
(X XOR Y) XOR Z = X

11-102

(X XOR Y) XOR Y = X XOR (Y XOR Y)
= X

Many encryption/decryption algorithms use the above property to encrypt/decrypt information.

original file XOR password $\longrightarrow$ encrypted file

encrypted file XOR password $\longrightarrow$ original file

11-103

A Simple Encryption/Decryption Program

- **The program encrypts/decrypts an image file.**
- **The name of the file to be encrypted/decrypted and the name of file holding the result of the encryption/decryption are given as command line arguments.**

```
./xor    auckland.jpg      new.jpg
```

11-104

- **The password is “compsci210” (10 bytes)**
- **Repeatedly read 10 bytes from the file and XOR the 10 bytes with the password and write the result to a file.**
- **The above process is repeated until the end of the file is reached.**

```
#include <stdio.h>

int main(int argc, char *argv[]) {
  FILE *file, *newFile;
```

```
#include <stdio.h>
#define blockSize 10
int main(int argc, char *argv[]) {
  FILE *file, *newFile;
  char password[blockSize]="compsci210";
  char block[blockSize];
  int i, numBytes;
```

```
#include <stdio.h>
#define blockSize 10
int main(int argc, char *argv[]) {
  FILE *file, *newFile;
  char password[blockSize]="compsci210";
  char block[blockSize];
  int i, numBytes;

  file = fopen(argv[1], "rb");
  newFile = fopen(argv[2], "wb");
```

```
./xor    auckland.jpg      new.jpg
```

```
#include <stdio.h>
#define blockSize 10
int main(int argc, char *argv[]) {
  FILE *file, *newFile;
  char password[blockSize]="compsci210";
  char block[blockSize];
  int i, numBytes;

  file = fopen(argv[1], "rb");
  newFile = fopen(argv[2], "wb");

    numBytes = fread(block, 1, blockSize, file);
```

file

read binary data

location for storing the data being read

size of one element

number of element

```
#include <stdio.h>
#define blockSize 10
int main(int argc, char *argv[]) {
  FILE *file, *newFile;
  char password[blockSize]="compsci210";
  char block[blockSize];
  int i, numBytes;

  file = fopen(argv[1], "rb");
  newFile = fopen(argv[2], "wb");

    numBytes = fread(block, 1, blockSize, file);
    for (i=0; i<numBytes; i++)
      block[i] = block[i]^password[i];
```

XOR the block with the password

```
#include <stdio.h>
#define blockSize 10
int main(int argc, char *argv[]) {
  FILE *file, *newFile;
  char password[blockSize]="compsci210";
  char block[blockSize];
  int i, numBytes;

  file = fopen(argv[1], "rb");
  newFile = fopen(argv[2], "wb");

    numBytes = fread(block, 1, blockSize, file);
    for (i=0; i<numBytes; i++)
      block[i] = block[i]^password[i];
    fwrite(block, 1, numBytes, newFile);
```

write binary data

location of the data

size of one element

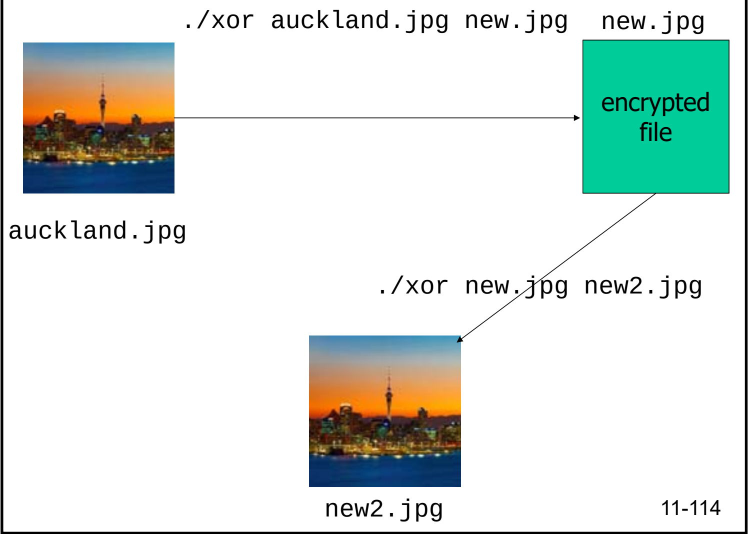
number of element

file

```
#include <stdio.h>
#define blockSize 10
int main(int argc, char *argv[]) {
  FILE *file, *newFile;
  char password[blockSize]="compsci210";
  char block[blockSize];
  int i, numBytes;

  file = fopen(argv[1], "rb");
  newFile = fopen(argv[2], "wb");
  do {
    numBytes = fread(block, 1, blockSize, file);
    for (i=0; i<numBytes; i++)
      block[i] = block[i]^password[i];
    fwrite(block, 1, numBytes, newFile);
  } while (numBytes == blockSize);
```

```
#include <stdio.h>
#define blockSize 10
int main(int argc, char *argv[]) {
  FILE *file, *newFile;
  char password[blockSize]="compsci210";
  char block[blockSize];
  int i, numBytes;

  file = fopen(argv[1], "rb");
  newFile = fopen(argv[2], "wb");
  do {
    numBytes = fread(block, 1, blockSize, file);
    for (i=0; i<numBytes; i++)
      block[i] = block[i]^password[i];
    fwrite(block, 1, numBytes, newFile);
  } while (numBytes == blockSize);
  fclose(newFile); fclose(file);
}
```

11-113



11-114

on-line C resources

FreeBSD Man Pages: http://www.freebsd.org/cgi/man.cgi

Many on-line tutorials on C, e.g.
http://www.tutorialspoint.com/c_standard_library/
http://www.cprogramming.com/tutorial/c-tutorial.html

11-115