CSE320 System Fundamentals II

TONY MIONE

Lecture overview

More about C Types

C Enumerated Type

C Pointers

C Arrays

C Structures

C I/O

Functions

Macros

C Standard Library

C Types [II]

C provides an ‘enumerated’ type (enum)

- A group of ‘named’ constants
- Underlying representation is an integer
- Provides readability to applications

C Pointers represent the memory address of a value/variable/function

typedef allows creation of named types

enum [Declaration]

Syntax:

enum <enumTag> = {<symbol1>,<symbol2>,…};

Example:

```
enum daysOfWeek = {Sunday, Monday, Tuesday,
        Wednesday, Thursday, Friday, Saturday};
```

enum [Use]

Syntax:

Example:

```
enum daysOfWeek = {Sunday, Monday, Tuesday,
        Wednesday, Thursday, Friday, Saturday};
enum daysOfWeek today = Monday;

if ((today >= Monday) && (today <= Friday)) {
  printf("Sorry, get to work!\n");
}
```

Pointers

Pointers are just memory addresses

C tracks the type that a pointer variable ‘points to’

Pointers [Declaration]

Syntax:

<type> *<variableName>[= &<declaredVariable>];

Example:

```
int *countAddress;
char *helloString = “Hello!”;
```

Pointers [Use]

Syntax:

<variable> = *<pointer>;

<pointer> = &<variable>;

Example:

```
count = 5;
countAddress = &count;
char aString[] = “This is a string.”;
char *stringPtr;
char firstChar;
// stringPtr points to same address as aString
stringPtr = aString;
firstChar = *stringPtr;
```

C Arrays

Arrays contain multiple items of the same type

Individual items retrieved by an index (0 is always the low index value)

Arrays can have any number of dimensions

C Arrays [Declaration]

Syntax:

<type> <name>[<#elems1>]…;

<type> <name>[<#elems1>] = {init1, init2, …};

<type> <name>[] = {init1, init2, …};

Example:

```
int a[5]; // Variable ‘a’ holds 5 integers (indexed 0-4)
float f[10]; // f holds 10 32-bit floating point value
int b[] = {3, 5, 7};  // b is 3 element integer array

// Chessboard array holds 8 rows and 8 columns of int
int chessboard[8][8];
```

C Arrays [use]

Syntax:

<name>[<index>]

Example:

```
sum[4] = sum[4] + nextInput;
chessboard[2][2] = 5;
if (chessboard[2][4] == 11) {…}
```

Exercise

Write a short C program that:

Creates an array of 10 doubles [call it 'values'] and initializes them to random values (pick the values yourself and initialize them directly in the declaration)

Using a *for* or *while* loop, compute the **root mean square** for the set of values in the double array. The root mean square is the square root of the average of squares. The formula is:

$$\textbf{rmsvalue} = \sqrt{((x_1 + x_2 + \ldots + x_n) / n)}$$

Use the math library's sqrt() function:

```
#include <math.h>
…
   rmsvalue = sqrt(sumofsquares);
```

Print the resulting value using:

```
printf ("RMS result of the array is: %f\n", rmsvalue);
```

C Structures

C Structures hold fields of differing types

Used to collect related data

C structures [Declaration]

Syntax:

```
struct <structTag> {
  <type> <name>;
  <type> <name>;
  …
}
```

Example:

```
struct _employee {
  char firstName[40];
  char lastName[40];
  int employeeNumber;
  int manager;
  …
}
```

C structures [Definition]

Syntax:

struct <structTag> <instanceName>;

Example:

```
// Create an array of 1000 employee records
struct _employee persList[1000];
```

C structures [use]

Syntax:

<instanceName>.<fieldName>

Example:

```
nextEmp = 100;
strcpy(persList[nextEmp].firstName, ‘Larry’);
strcpy(persList[nextEmp].lastName, ‘Boy’);
persList[nextEmp].manager = 5;
```

C functions

Functions allow developer to:

- Subdivide large tasks
- Isolate frequently used logic

Functions may take 1 or more ‘parameters’

Functions may return a value

Functions may call other functions

Functions may call themselves (recursion)

Functions may be mutually recursive

Functions [Declaration]

Syntax:

<returnType> <name>(<parametersList>|void);

Example:

int square(int input);

Functions [Definition]

Syntax:

```
<returnType> <name>(<parametersList>|void) {
  <functionBodyStatements>
}
```

Example:

```
int square(int input) {
  return input * input;
}
```

functions [use]

Syntax:

<name>(<argumentList>)

Example:

```
int a, b;

b = 5;
a = square(b) + 1;
```

Exercise

Write a function called *f2c* which accepts a single float parameter [which is a temperature in Fahrenheit], converts it to Celsius, and returns it to the caller.

Add this main routine to the file and run it to test the function:

```
int main(int argc, char **argv)
{
   float testvals[5] = {32.0, 212.0, 98.6, 101.1, 0.0};
   int idx;

   for (idx = 0; idx <= 5; idx++) {
       printf ("%f in Celcius is: %f\n", testvals[idx], f2c(testvals[idx]);
  }
}
```

C macros

Macros are templates expanded by the C preprocessor

Blind text substitution – no understanding of C syntax or semantics

Can take arguments → Use caution with arguments!!!

macro [declaration]

Syntax:

#define <macroName> <subsText>

#define <macroName>(<parametersList>) <subsText>

Example:

```
#define  MAX_RECORDS  100
#define  SQUARE(A)  A*A
```

macro [use]

Syntax:

<macroName>

<macroName>(<argumentList>)

Example:

```
int a, b, c;
b = 5;
a = SQUARE(b);
c = SQUARE(b+1);    // Bad!
```

macro – Writing good macros

From last slide:

```
c = SQUARE(b+1);
    generates:
c = b + 1 * b + 1;    // which equals 2 * b + 1
```

Example:

```
#define SQUARE(A) ((A)*(A))
```

Now:

```
c = SQUARE(b+1);
      generates:
c = ((b+1) * (b+1));
```

C Standard Library

A collection of useful functions

Include:

- String functions
- Math functions
- I/O functions
- Time/Date functions
- Others

Most used: String, Math, and I/O functions

C I/O

Include file: <stdio.h>

Functions:

- FILE *fopen(char *filename, char *mode);
- size_t fread(void *buffer, size_t size, size_t count, FILE *fptr);
- size_t fwrite(const void *buffer, size_t size, size_t count, FILE *fptr);
- int fprintf(FILE *fptr, const char *fmtString, …);
- int fclose(FILE *fptr);
- int printf(const char *fmtString, …);
- int scanf(const char *fmtString, …);
- <lots more>

C I/O Examples

```
#include <stdio.h>
int main(int argc, char **argv) {
  FILE *infile;
  char rbuffer[85];
  size_t rcount;
  if (argc > 1) {
    infile = fopen(argv[1], "r");
    rcount = fread(rbuffer, 1, 80, infile);
    while (rcount > 0) {
       rbuffer[rcount] = '\0';
       printf ("Read: ||%s||\n", rbuffer);
       rcount = fread(rbuffer, 1, 80, infile);
     }
    fclose(infile);
  }
}
```

C I/O Examples

```
#include <stdio.h>
int main(int argc, char **argv) {
  int theNumber;
  char nameBuffer[80];

  printf("Hi, what's your name? ");
  scanf("%s", nameBuffer);
  printf("Enter a number: ");
  scanf("%d", &theNumber);

  printf ("Hello, %s, you gave me %d\n", nameBuffer, theNumber);
}
```

Exercise

Take the *f2c()* function from the previous exercise. Rewrite the main procedure to use *printf* and *scanf* to accept values from the terminal and print the conversion for the user.

String library

Include: <string.h>

Functions:

- char *strcpy(char *dest, char *src);
- char *strncpy(char *dest, char *src, size_t len);
- char *strcat(char *dest, char *src);
- char *strncat(char *dest, char *src, size_t len);
- void *memcpy(char *dest, char *src, size_t len);
- char *strchr(char *s, char c);
- char *strrchr(char *s, char c);
- <lots more>

String library examples

```
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv) {
  char sent[81];
  char theChar;
  char *foundAt = 0;
  printf("Hi, please give me a sentence (up to 80 characters): ");
  // Code to read sentence into ‘sent’ array
  printf("Enter a character to find: ");
  // Code to read character from user here.
  foundAt = strchr(sent, theChar);
  if (foundAt > 0) {
     printf ("Found character %c at %d characters from the start.\n",
           theChar,
           (int) ((int)foundAt - (int)sent));
   } else {
     printf ("Didn't find %c\n");
   }
}
```

String Library Examples

```
#include <stdio.h>
int main(int argc, char **argv) {
  int theNumber;
  char nameBuffer[80];

  printf("Hi, what's your name? ");
  scanf("%s", nameBuffer);
  printf("Enter a number: ");
  scanf("%d", &theNumber);

  printf ("Hello, %s, you gave me %d\n", nameBuffer, theNumber);
}
```

String Library Examples

```
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv) {
  char word1[20], word2[20];
  char bothwords[80];

  printf("Enter a word: ");
  scanf("%s", word1);
  printf("Enter another word: ");
  scanf("%s", word2);
  strcpy(bothwords, word1);
  strcat(bothwords, " and ");
  strcat(bothwords, word2);
  printf("You entered %s\n", bothwords);
}
```

math library

Include: <math.h>

Link options: -lm

Functions:

- double sqrt(double x);
- double exp(double x);
- double pow(double x, double y);
- double sin(double x); // Also: cos, tan, asin, acos, atan

Math library examples

```
#include <stdio.h>
#include <math.h>
int main(int argc, char **argv) {
  float s1 = 0.0, s2 = 0.0, hyp = 0.0, testhyp, eps=.01;
  float angle1, r2dfactor = 360.0/6.28;
  printf("Enter side1, side2, and hyp length: ");
  scanf("%f %f %f", &s1, &s2, &hyp);
  testhyp=sqrt(s1*s1 + s2*s2);
  if (abs(testhyp-hyp) > eps) {
     printf("Not a right trinangle!\n");
     return;
   } else {
   angle1 = (asin(s1/hyp)) * r2dfactor;
   printf ("Angle1=%f, angle2=%f\n", angle1, 90-angle1);
  }
}
```

Questions?