Uncovered Introductory topics of C

Xiaoqiang Wang

The Preprocessor

- The C preprocessor permits you to define simple macros that are evaluated and expanded prior to compilation.
- Commands begin with a ‘#’. Abbreviated list:
  - `#define` : defines a macro
  - `#undef` : removes a macro definition
  - `#include` : insert text from file
  - `#if` : conditional based on value of expression
  - `#ifdef` : conditional based on whether macro defined
  - `#ifndef` : conditional based on whether macro is not defined
  - `#else` : alternative
  - `#elif` : conditional alternative
  - `defined()` : preprocessor function: 1 if name defined, else 0

```
#if defined(__NetBSD__)
```

Preprocessor: Macros

- Using macros as functions, exercise caution:
  - flawed example: `#define mymult(a,b) a*b`
    - Source: `k = mymult(i-1, j+5);`
    - Post preprocessing: `k = i – 1 * j + 5;`
  - better: `#define mymult(a,b) (a)*(b)`
    - Source: `k = mymult(i-1, j+5);`
    - Post preprocessing: `k = (i – 1)*(j + 5);`
- Be careful of *side effects*, for example what if we did the following
  - Macro: `#define mysq(a) (a)*(a)`
  - flawed usage:
    - Source: `k = mysq(i++)`
    - Post preprocessing: `k = (i++)*(i++)`
- Alternative is to use inline'ed functions
  - `inline int mysq(int a) {return a*a};`
  - `mysq(i++)` works as expected in this case.

Preprocessor: Conditional Compilation

- Its generally better to use inline'ed functions
- Typically you will use the preprocessor to define constants, perform conditional code inclusion, include header files or to create shortcuts
- `#define DEFAULT_SAMPLES  100`
- `#ifdef __linux`

  ```
   static inline int64_t
     gettime(void) {...}
  ```
- `#elif defined(sun)`

  ```
   static inline int64_t
     gettime(void) {return (int64_t)gethrtime()}
  ```
- `#else`

  ```
   static inline int64_t
     gettime(void) {... gettimeofday()...}
  ```
- `#endif`

Signed and unsigned data types

- int
- unsigned int
- short, (short int)
- unsigned short, (unsigned short int)
- long, (long int)
- unsigned long, (unsigned long int)

Type conversion

- float -> double
- int -> unsigned int -> long int -> unsigned long int
- long -> float
- 1234L is long integer
- 1234 is integer
- 12.34 is float
- 12.34L is long float

Type Conversion

```
char c;
short int s;
int i;
unsigned int u;
long int l;
unsigned long int ul;
float f;
double d;
long double ld;
i = i + c;   /* c is converted to int  */
i = i + s;   /* s is converted to int  */
u = u +i;  /* i is converted to unsigned int  */
l = l + u;  /* u is converted to long int  */
ul =ul + l; /* l is converted to unsigned long int  */
f = f + ul;  /* ul is converted to float  */
d = d + f;  /* f is converted to double  */
ld = ld + d;  /* d is converted to long double  */
```

Bitwise Operations

- Applied to char, int, short, long
  - And &
  - Or |
  - Exclusive Or ^
  - Left-shift <<
  - Right-shift >>
  - one's complement ~

Example: Bit Count

```
/*
   count the 1 bits in a number
   e.g. bitcount(0x45) (01000101 binary) returns 3
*/

int bitcount (unsigned int x) {
   int b;

   for (b=0; x != 0; x = x >> 1)#
      if (x & 01)     /* octal 1 = 000000001 */
          b++;

   return b;
}
```

Conditional Expressions

- Conditional expressions

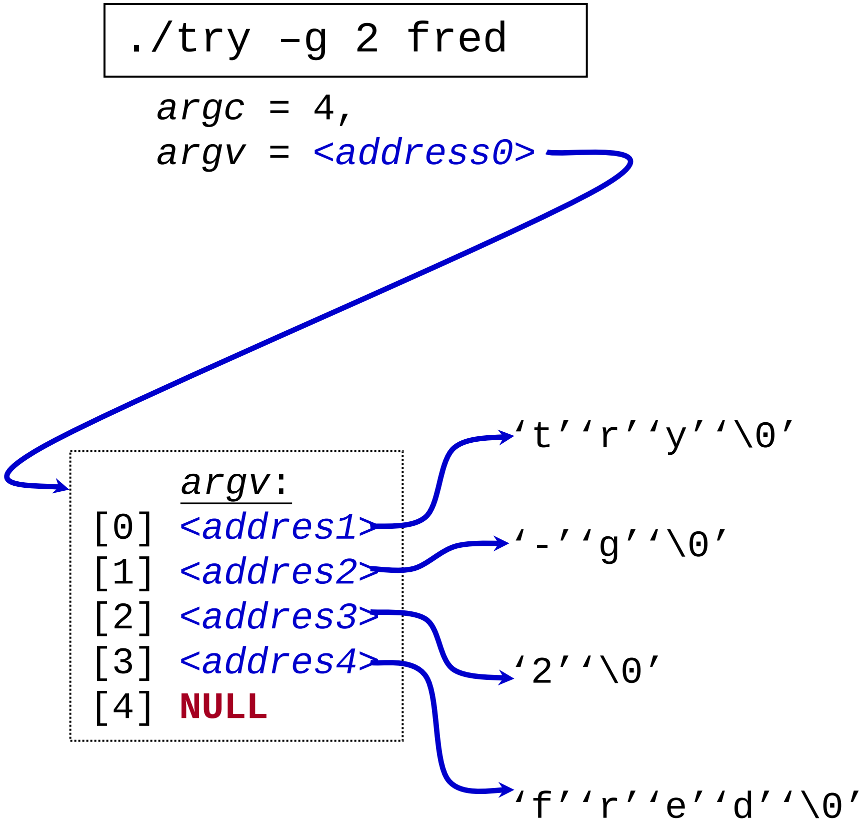
  expr1? expr2:expr3;
- if expr1 is true then expr2 else expr3

```
for (i=0; i<n; i++)#
   printf("%6d %c",a[i],(i%10==9||i==(n-1))?'\n':' ');
```

Passing Command Line Arguments

- When you execute a program you can include arguments on the command line.
- The run time environment will create an argument vector.
  - `argv` is the argument vector
  - `argc` is the number of arguments
- Argument vector is an array of pointers to strings.
- a *string* is an array of characters terminated by a binary 0 (NULL or ‘\0’).
- *argv[0]* is always the program name, so *argc* is at least 1.

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



```
./try -g 2 fred
```

```
argc = 4,
argv = <address0>
```

```
    argv:
[0] <addres1>
[1] <addres2>
[2] <addres3>
[3] <addres4>
[4] NULL
```

```
‘t’‘r’‘y’‘\0’
‘-’‘g’‘\0’
‘2’‘\0’
‘f’‘r’‘e’‘d’‘\0’
```

Time functions

- http://en.wikipedia.org/wiki/C_date_and_time_functions

	Identifier	Description
Time manipulation	difftime	computes the difference between times
	time	returns the current time of the system as time since the epoch (which is usually the Unix epoch)
	clock	returns a processor tick count associated with the process
Format conversions	asctime	converts a tm object to a textual representation
	ctime	converts a time_t object to a textual representation
	strftime	converts a tm object to custom textual representation
	wcsftime	converts a tm object to custom wide string textual representation
	gmtime	converts time since the epoch to calendar time expressed as Coordinated Universal Time
	localtime	converts time since the epoch to calendar time expressed as local time
	mktime	converts calendar time to time since the epoch
Constants	CLOCKS_PER_SEC	number of processor clock ticks per second
Types	tm	calendar time type
	time_t	time since the epoch type
	clock_t	process running time type

Structs and Unions

- structures

  ```
  struct MyPoint {int x, int y};
  typedef struct MyPoint MyPoint_t;
  MyPoint_t point, *ptr;
  point.x = 0;point.y = 10;
  ptr = &point; ptr->x = 12; ptr->y = 40;
  ```

- unions

  ```
  union MyUnion {int x; MyPoint_t pt;
  struct {int 3; char c[4]} S;};
  union MyUnion x;
  ```

  - Can only use one of the elements. Memory will be allocated for the largest element

Conditional Statements (switch)

```
int c = 10;
switch (c) {
  case 0:
    printf(“c is 0\n”);
    break;
  ...
  default:
    printf(“Unknown value of c\n”);
    break;
}
```

- What if we leave the break statement out?
- Do we need the final break statement on the default case?

Project Documentation

- README file structure
  - ***Section A***: *Introduction*
    describe the project, paraphrase the requirements and state your understanding of the assignments value.
  - ***Section B***: *Design and Implementation*
    List all files turned in with a brief description for each. Explain your design and provide simple psuedo-code for your project. Provide a simple flow chart of you code and note any constraints, invariants, assumptions or sources for reused code or ideas.
  - ***Section C***: *Results*
    For each project you will be given a list of questions to answer, this is where you do it. If you are not satisfied with your results explain why here.
  - ***Section D***: *Conclusions*
    What did you learn, or not learn during this assignment. What would you do differently or what did you do well.