CSE1301 - Computer Programming

Advanced C Programming

Dr. Tim Ferguson

School of Computer Science and Software Engineering,
Monash University. AUSTRALIA.

Tel: +61-3-99053227 FAX: +61-3-99053574
E-Mail: timf@csse.monash.edu.au
WWW: http://www.csse.monash.edu.au/∼timf/
Clayton Office: Building 75, Room 192

Topics

- Process of compiling a C program.
- Linking multiple object files.
- Conditional expressions.
- The `switch` statement.
- The `sizeof` operator.
- Address arithmetic.
- Self-referential Structures.

NOTE: This material will not be on the exam

How is a C Program Compiled?

- Compiling a C program is broken down into four parts:
  1. **Preprocess** the `.c` file into a `.i` file.
  2. **Compile** the preprocessed `.i` file into an assembly `.s` file.
  3. **Assemble** the assembly `.s` file into an object `.o` file.
  4. **Link** one or more object `.o` files with one or more library files into an executable `.exe` file.
- Let us look an example which shows each of the above steps.
  - Use the GNU project C compiler (gcc).
  - Show the output of the above steps on `helloworld.c`.
  - Create a cut-down version of `stdio.h` called `mystdio.h` for our demonstration.

C Source Code

helloworld.c

```
#include "mystdio.h"
#define C1  1
#define C2  2+3
#define SUM(a, b)  (a + b)

int main(void)
{
int v1 = C1, v2 = C2;

      /* this is a comment */
   printf("Hello World! %d\n",
      SUM(v1, v2));
#if 0
   printf("Yeah Baby!\n");
#endif
   return 0;
}
```

mystdio.h

```
/* ... previous defns and consts ... */

typedef struct _IO_FILE FILE;

extern int fclose (FILE *__stream) ;
extern FILE *fopen (const char *__filename,
   const char *__modes) ;
extern int fscanf (FILE *__stream,
   const char *__format, ...) ;
extern int printf (const char *__format, ...) ;
extern int scanf (const char *__format, ...) ;
extern int sscanf (const char *__s,
   const char *__format, ...) ;

/* ... more defns and consts ... */
```

C Preprocessor Output

helloworld.i

```
# 1 "helloworld.c"
# 1 "mystdio.h" 1

typedef struct _IO_FILE FILE;

extern int fclose (FILE *__stream) ;
extern FILE *fopen (const char *__filename,
   const char *__modes) ;
extern int fscanf (FILE *__stream,
   const char *__format, ...) ;
extern int printf (const char *__format, ...) ;
extern int scanf (const char *__format, ...) ;
extern int sscanf (const char *__s,
const char *__format, ...) ;
# 2 "helloworld.c" 2

int main(void)
{
int v1 = 1, v2 = 2+3;

   printf("Hello World! %d\n",
      (v1 + v2));
   return 0;
}
```

C Compiler Output

helloworld.s

```
    .file "helloworld.c"
    .version "01.01"
gcc2_compiled.:
        .section .rodata
.LC0:
    .string  "Hello World! %d\n"
.text
    .align 4
.globl main
    .type  main,@function
main:
    pushl %ebp
    movl  %esp, %ebp
    subl  $8, %esp
    movl  $1, -4(%ebp)
    movl  $5, -8(%ebp)
    subl  $8, %esp
    movl  -8(%ebp), %eax
    addl  -4(%ebp), %eax
    pushl %eax
    pushl $.LC0
    call  printf
    addl  $16, %esp
    movl  $0, %eax
    leave
    ret
.Lfe1:
    .size  main,.Lfe1-main
    .ident   "GCC: (GNU) 2.96 20000731 ..."
```

Assembler Output

helloworld.o – This is a HEX dump of a binary file

```
000000  7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00   .ELF............
000010  01 00 03 00 01 00 00 00 00 00 00 00 00 00 00 00   ................
000020  0c 01 00 00 00 00 00 00 34 00 00 00 00 00 28 00   ........4.....(.
000030  0b 00 08 00 55 89 e5 83 ec 08 c7 45 fc 01 00 00   ....U......E....
000040  00 c7 45 f8 05 00 00 00 83 ec 08 8b 45 f8 03 45   ..E.........E..E
000050  fc 50 68 00 00 00 00 e8 fc ff ff ff 83 c4 10 b8   .Ph.............
000060  00 00 00 00 c9 c3 89 f6 08 00 00 00 00 00 00 00   ................
000070  01 00 00 00 30 31 2e 30 31 00 00 00 48 65 6c 6c   ....01.01...Hell
000080  6f 20 57 6f 72 6c 64 21 20 25 64 0a 00 00 47 43   o World! %d...GC
000090  43 3a 20 28 47 4e 55 29 20 32 2e 39 36 20 32 30   C: (GNU) 2.96 20
......
000340  03 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
000350  03 00 07 00 1d 00 00 00 00 00 00 00 32 00 00 00   ............2...
000360  12 00 01 00 22 00 00 00 00 00 00 00 00 00 00 00   ...."...........
000370  10 00 00 00 00 68 65 6c 6c 6f 77 6f 72 6c 64 2e   .....helloworld.
000380  63 00 67 63 63 32 5f 63 6f 6d 70 69 6c 65 64 2e   c.gcc2_compiled.
000390  00 6d 61 69 6e 00 70 72 69 6e 74 66 00 00 00 00   .main.printf....
0003a0  1f 00 00 00 01 06 00 00 24 00 00 00 02 0a 00 00   ........$.......
```

Linker Output

- The linker combines one or more object files with one or more library files.
- A library is simply a collection or an archive of many object files.
- In our example, the linker:
  - Takes the single `helloworld.o` object file;
  - Combines it with the standard C library's objects; and
  - Produces the executable `helloworld.exe`.
- The format of the executable is not very different from the previous object file.

Recall A Previous Example...

countwords.c

```
int countWords ( FILE *inpf )
{
 .....
}

FILE* openInput ( void )
{
 ....
}
```

countwords.h

```
#define MAXLEN  100
int countWords ( FILE *inpf );
FILE*  openInput ( void );
```

testcount1.c

```
#include <stdio.h>
#include <stdlib.h>
#include "countwords.h"
#include "countwords.c" /* THIS IS BAD */

int main()
{
  FILE  *inputFile = NULL;
  int   count;

  inputFile = openInput();
  count = countWords(inputFile);
  printf("%d words in file.\n", count);
  fclose(inputFile);

  return 0;
}
```

A Better Alternative (Linking Multiple Files)

countwords.c

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

int countWords ( FILE *inpf )
{
 .....
}

FILE* openInput ( void )
{
 ....
}
```

countwords.h

```
#define MAXLEN  100
int countWords ( FILE *inpf );
FILE*  openInput ( void );
```

testcount1.c

```
#include <stdio.h>
#include "countwords.h"

int main()
{
  FILE  *inputFile = NULL;
  int   count;

  inputFile = openInput();
  count = countWords(inputFile);
  printf("%d words in file.\n", count);
  fclose(inputFile);

  return 0;
}
```

A Better Alternative (Linking Multiple Files)

- Using the compiler:
  - Compile `countwords.c` into an object file `countwords.o`.
  - Compile `testcount1.c` into an object file `testcount1.o`.
- Now link `countwords.o` with `testcount1.o` and the standard C library to create our executable.

Conditional Expression

- Consider the following code:

```
if (a > b) z = a;
else z = b;
```

- That is, `z` will be the maximum of `a` and `b`.
- An alternative is the conditional expression:

```
z = (a > b) ? a : b;
```

- That is, in the following expression:

```
expr1 ? expr2 : expr3
```

- If `expr1` evaluates to true (i.e. non-zero) then `expr2` is evaluated, otherwise `expr3` is evaluated.
- Only one of `expr2` and `expr3` is evaluated, with the result being the value of the overall expression.

Switch Statement

- The `switch` statement tests if an expression matches one of a number of *constant* integers and branches accordingly.

Syntax

```
switch( expression )
{
  case const-expr : statements
  case const-expr : statements
  ...
  default: statements
}
```

```
if(c == 'a') a_cnt++;
else if(c == 'b') b_cnt++;
else if(c == 'c' || c == 'C') c_cnt++;
else other_cnt++;
```

```
switch(c)
{
  case 'a':
     a_cnt++;
     break;
  case 'b':
     b_cnt++;
     break;
  case 'c':
  case 'C':
     c_cnt++;
     break;
  default:
     other_cnt++;
     break;
}
```

Sizeof Operator

- The `sizeof` operator provides the number of bytes required to store an object and is evaluated during compilation of a program.
- For example:

```
struct tst {
  char arr[16];
  int a;
  double b;
};
char str[] = "Hi Mum";
printf("char: %d\n", sizeof(char));
printf("short: %d\n", sizeof(short));
printf("int: %d\n", sizeof(int));
printf("long long: %d\n", sizeof(long long));
printf("float: %d\n", sizeof(float));
printf("double: %d\n", sizeof(double));
printf("tst: %d\n", sizeof(struct tst));
printf("\"%s\": %d\n", str, sizeof(str));
```

Sizeof Operator

- The `sizeof` operator provides the number of bytes required to store an object and is evaluated during compilation of a program.
- For example:

```
struct tst {
  char arr[16];
  int a;
  double b;
};
char str[] = "Hi Mum";
printf("char: %d\n", sizeof(char));
printf("short: %d\n", sizeof(short));
printf("int: %d\n", sizeof(int));
printf("long long: %d\n", sizeof(long long));
printf("float: %d\n", sizeof(float));
printf("double: %d\n", sizeof(double));
printf("tst: %d\n", sizeof(struct tst));
printf("\"%s\": %d\n", str, sizeof(str));
```

On a 32-bit Pentium 3 using gcc:

```
char: 1
short: 2
int: 4
long long: 8
float: 4
double: 8
tst: 28
"Hi Mum": 7
```

Address Arithmetic

- If `p` is a pointer to an array (for example `int arr[100], *p = arr;`):
  - `p++` advances the pointer to the next element in the array.
  - `p+=i` advances it `i` elements further in the array.
- These are simple forms of address arithmetic.
- Example 1: String copy and string length.

```
void my_strcpy(char *d, char *s)
{
  while((*d = *s) != '\0')
    {
    s++;
    d++;
    }
}
```

```
int my_strlen(char *str)
{
  char *p = str;
  while(*p != '\0') p++;
  return p - str;
}
```

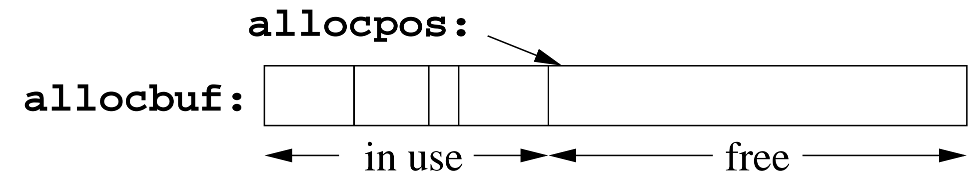
Address Arithmetic

- Example 2: Rudimentary storage allocator.
- Calls to `my_free` must be made in the opposite order to calls to `my_alloc`.

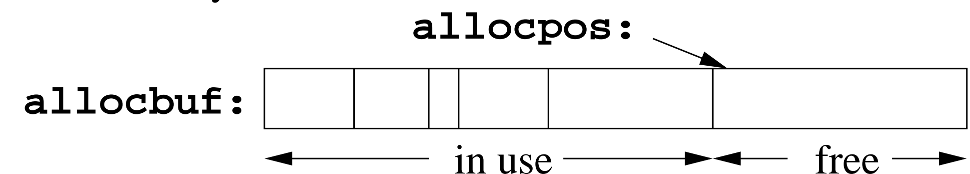
```
#define ALLOCSIZE   10000
static char allocbuf[ALLOCSIZE];
static char *allocpos = allocbuf;

char *my_alloc(int n)
{
  if(allocbuf + ALLOCSIZE - allocpos >= n)
     {
     allocpos += n;
     return  allocpos - n;
     }
  else return NULL;
}
```

```
void my_free(char *p)
{
  if(p >= allocbuf
     && p < allocbuf + ALLOCSIZE)
     allocpos = p;
}
```

Before call to my_alloc:



After call to my_alloc:



Self-referential Structures

- A structure cannot contain an instance of itself. For example,

```
struct bad_struct {
  int a, b;
  struct bad_struct right;
};
```

will not work.

- It may however contain references (pointers) to itself.
- For example:

```
struct lnode {
  char word[MAXLEN];
  int count;
  struct lnode *next;
};
```

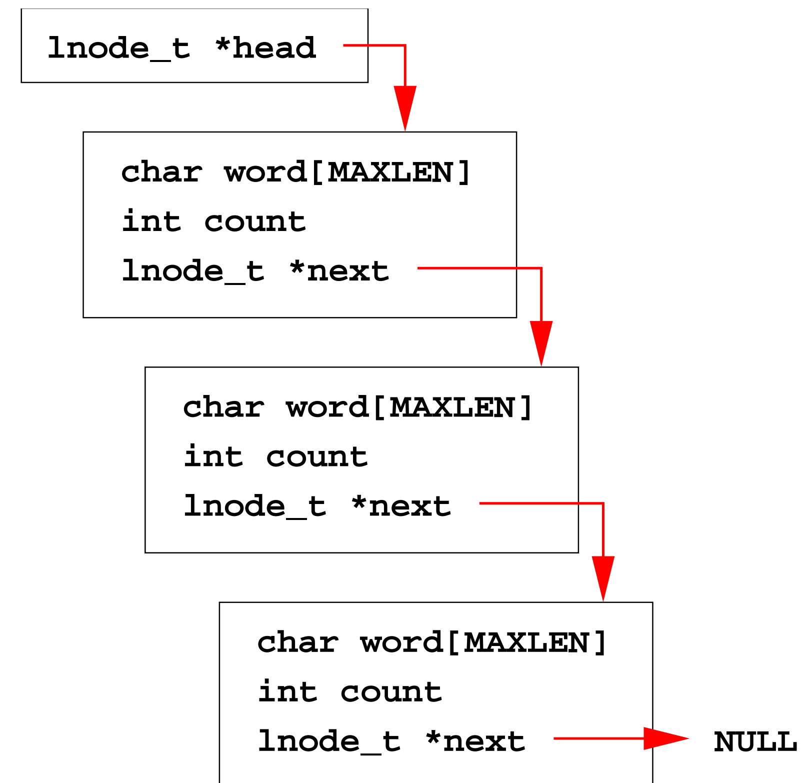
OR

```
typedef struct lnode lnode_t;
struct lnode {
  char word[MAXLEN];
  int count;
  lnode_t *next;
};
```

Self-referential Structures

- Using the previous structure, we can create a "linked list":

```
lnode_t *add_llist(lnode_t *p, char *w)
{
  if(p == NULL)
    {
    p = my_alloc(sizeof(lnode_t));
    strcpy(p->word, w);
    p->count = 1;
    p->next = NULL;
    }
  else if(strcmp(w, p->word) == 0)
    p->count++;
  else p->next = add_llist(p->next, w);
  return p;
}
```



Self-referential Structures

```
lnode_t *head = NULL;
```

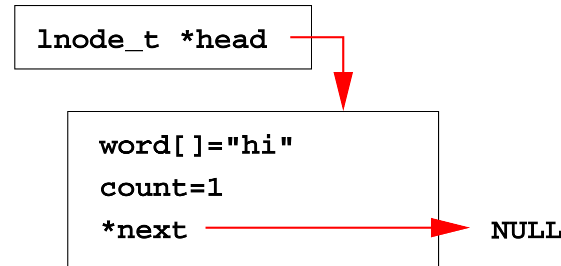
```
lnode_t *add_llist(lnode_t *p, char *w)
{
  if(p == NULL)
    {
    p = my_alloc(sizeof(lnode_t));
    strcpy(p->word, w);
    p->count = 1;
    p->next = NULL;
    }
  else if(strcmp(w, p->word) == 0)
    p->count++;
  else p->next = add_llist(p->next, w);
  return p;
}
```

lnode_t *head → NULL

Self-referential Structures

```
lnode_t *head = NULL;
   head = add_llist(head, "hi");
```

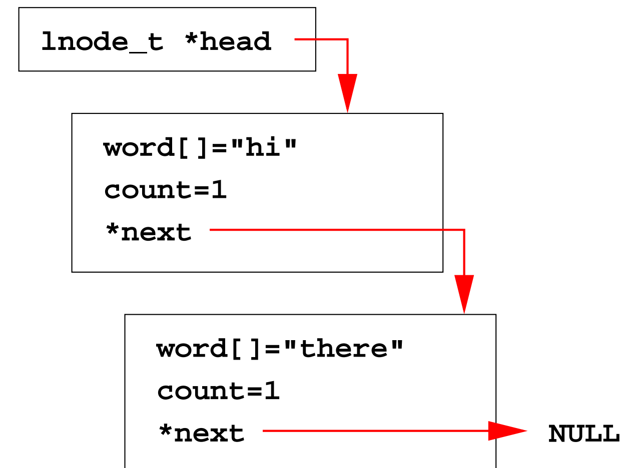
```
lnode_t *add_llist(lnode_t *p, char *w)
{
  if(p == NULL)
    {
    p = my_alloc(sizeof(lnode_t));
    strcpy(p->word, w);
    p->count = 1;
    p->next = NULL;
    }
  else if(strcmp(w, p->word) == 0)
    p->count++;
  else p->next = add_llist(p->next, w);
  return p;
}
```



Self-referential Structures

```
lnode_t *head = NULL;
   head = add_llist(head, "hi");
   head = add_llist(head, "there");
```

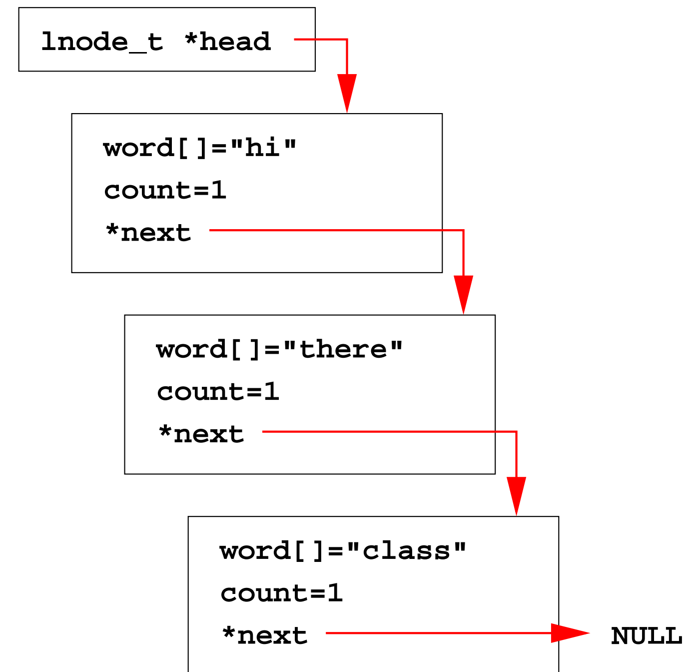
```
lnode_t *add_llist(lnode_t *p, char *w)
{
  if(p == NULL)
    {
    p = my_alloc(sizeof(lnode_t));
    strcpy(p->word, w);
    p->count = 1;
    p->next = NULL;
    }
  else if(strcmp(w, p->word) == 0)
    p->count++;
  else p->next = add_llist(p->next, w);
  return p;
}
```



Self-referential Structures

```
lnode_t *head = NULL;
   head = add_llist(head, "hi");
   head = add_llist(head, "there");
   head = add_llist(head, "class");
```

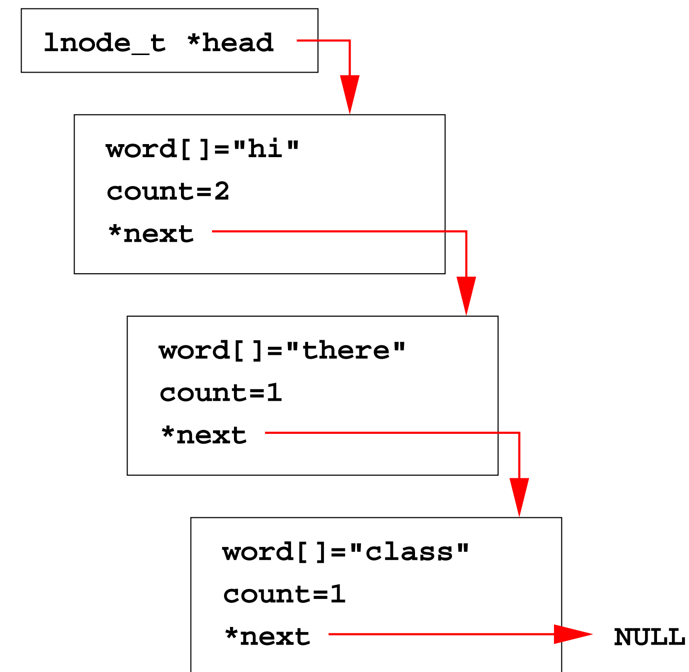
```
lnode_t *add_llist(lnode_t *p, char *w)
{
  if(p == NULL)
    {
    p = my_alloc(sizeof(lnode_t));
    strcpy(p->word, w);
    p->count = 1;
    p->next = NULL;
    }
  else if(strcmp(w, p->word) == 0)
    p->count++;
  else p->next = add_llist(p->next, w);
  return p;
}
```



Self-referential Structures

```
lnode_t *head = NULL;
   head = add_llist(head, "hi");
   head = add_llist(head, "there");
   head = add_llist(head, "class");
   head = add_llist(head, "hi");
```

```
lnode_t *add_llist(lnode_t *p, char *w)
{
  if(p == NULL)
    {
    p = my_alloc(sizeof(lnode_t));
    strcpy(p->word, w);
    p->count = 1;
    p->next = NULL;
    }
  else if(strcmp(w, p->word) == 0)
    p->count++;
  else p->next = add_llist(p->next, w);
  return p;
}
```



- Covered several advanced topics in C programming.
- **You do not need to know this for the exam.**
- You will (re-)learn this in future subjects.