C

Slides and captured lecture (video and sound) are available at:

`www.cim.mcgill.ca/~jer/C/`

Crash Course -- Part III

Today, we will cover:

- structures
- using linked lists
- separate compilation
- Makefiles
- debugging (using gdb on UNIX)

Structures

Use `struct` to create or define a new type:

```
struct str_data {
   char *string;
   int length;
};
struct str_data s1, s2;
```

Syntax

```
struct structure_name {
 type1 member1;
 type2 member2;
 …
};
```

A member of a structure is referred to by:

```
structure_name.member
```

Structures, cont.

Example:

```
struct str_data {
   char *string;
   int length;
};
struct str_data s1, s2;

s1.string = (char *) malloc(80);

strcpy (s1.string, "How now brown cow.");
s1.length = strlen(s1.string);
s2 = s1; /* can copy structures, pass to fcns */
```

Arrays of Structures

Example:

```
#define MAX_STUDENT 100

typedef struct {  /* define a new type */
   char name[80];
   int name_len;
   int student_number;
} student_t;

/* create list of elements of type student_t */
student_t class[MAX_STUDENT];

for (i = 0; i < MAX_STUDENT; i++) {
   gets(class[i].name);
   class[i].name_len = strlen(class[i].name);
}
```

Pointers to Structures

These are all equivalent:

```
list[0].length = ...
(*list).length = ...
list->length = ...
```

Rules:

- p->y is shorthand for (*p).y
- means "follow the pointer p into structure member y
- more efficient to pass pointers to structs rather than the structs themselves

Self-Referential Structures

Example: Linked List

This does not work:

```
typedef struct {
   int data;
   node_t *next;
} node_t;
```

Rules:

- a structure cannot refer to its own name before it is defined
- but it can contain a pointer to itself
- how?

Naming your structure

Give the structure a name:

```
typedef struct element {
   int data;
   struct element *next;
} node_t;
```

Rules:

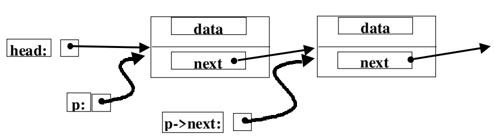
- **next** is a pointer to a structure of type `node_t`
- we can use this to create a pointer to the head of the list:

```
node_t *head = (node_t *) malloc (sizeof (node_t));
```

How do we define the end of the list?

Traversing a Linked List

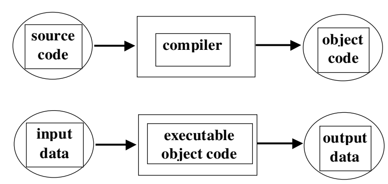
```
node_t *p;

p = head;
while (p != NULL) {
   printf ("data = %d\n", p->data);
   p = p->next;
}
```



Compiling

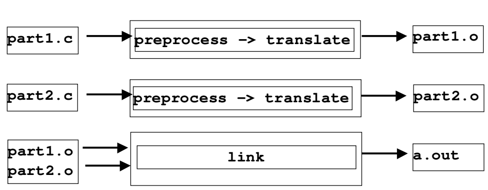
Steps to a running program:

```
% vi prog.c
% gcc prog.c
% a.out
```



Separate Compilation

```
% gcc -c part1.c
% gcc -c part2.c
% gcc part1.o part2.o
% a.out
```



Makefiles

```
# Sample Makefile
#
# The following two lines say that pgm
# depends on prog.o and sub.o.
# The action required to create pgm is
# to link prog.o and sub.o.

pgm: prog.o sub.o
 gcc -o pgm prog.o sub.o

# The next four lines specify what each of
# prog.o and sub.o depend upon, and how these
# object files are to be created.

prog.o: incl.h prog.c
 gcc -c prog.c
sub.o: incl.h sub.c
 gcc -c sub.c
```

Header Files

```
/* File: myheader.h
 * Definitions for program build.c
 */

/* Booleans */
typedef int boolean_t;
#define TRUE 1
#define FALSE 0

/* node type */
typdef struct node_t {
...
}

/* function prototypes */
node_t *NewNode(char *label, int cost);
void KillNode (node_t *node);
```

Naming Conventions

Consistency is bliss. Suggested format:

- Constants: use all upper case
  ```
  #define TRUE 1
  #define MAX_ELEMENTS 50
  ```
- Variables: use all lower case
  ```
  char *name; int num_students;
  ```
- Functions: upper case for first letter of each world
  ```
  float *GetCompressedValue(); char *GetNextWord();
  ```
- Types: all lower case with a "_t" suffix (for “type”)
  ```
  typedef struct node_t {...}
  ```

The GNU Debugger: GDB

Allows you to:

- Make your program stop on specified conditions
- Step through your program’s execution line by line
- Examine what has happened when your program has stopped or crashed.

Using GDB

Compile your program with the -g option:

```
% gcc -g -o prog prog.c
```

To run your program in GDB:

```
% gdb prog
(gdb) run
```

To examine your program after a crash (core dump):

```
% prog
Segmentation fault (core dumped)

% gdb prog core
Core was generated by `prog'.
Program terminated with signal 11, Segmentation
 fault.
(gdb) ...
```

Some GDB Commands

Set a breakpoint:

```
(gdb) break [file:]function
```

Run your program:

```
(gdb) run [arglist]
```

List 10 lines of code:

```
(gdb) list
```

Display the value of an expression:

```
(gdb) print expr
```

Display a value after each stoppage:

```
(gdb) display expr
```

More GDB Commands

Execute next line (skipping function calls):

```
(gdb) next
```

Execute next line stepping into fcn calls:

```
(gdb) step
```

Continue after a break:

```
(gdb) cont
```

Getting help:

```
(gdb) help [name]
```

Exiting:

```
(gdb) quit
```

Debugging A Program I

Example: (searching a linked list)

```
node_t *Search(node_t *head, int data)
/* search for data, return pointer to the node */
{
   node_t *p;

   p = head;
   do {
      if (p->data = data)
         return (p);
      else
         p = p->next;
   } while (p != NULL);
   return (NULL);
}
```

Debugging A Program II

There is a bug:

Input:

```
1 2 3 4 -1      (values for linked list)
17              (value to search for)
```

Output:

```
17 is found     (! but it's not in the list)
17 3 2 1        (! 4 is missing)
```

Steps to find the bug:

- start up GDB:

```
% gdb linklist
```

Debugging A Program III

- set a breakpoint at the Search function:

```
(gdb) break Search
Breakpoint 1 at 0x10ab0: file linklist.c, line 13.
(gdb)
```

- run the program and provide some input:

```
(gdb) run
Starting program: linklist
1 2 3 4 -1 17
Breakpoint 1, Search (head=0x21b50, data=17)
13          p = head;
(gdb)
```

- display some variables:

```
(gdb) display p
1: p = (node_t *) 0x20d58
(gdb) display *p
2: *p = {data = 4, next = 0x20d48}
(gdb)
```

Debugging A Program IV

- step through the code and watch the variables change:

```
(gdb) step
15            if (p->data = data)
(gdb) display p
1: p = (node_t *) 0x21b50
(gdb) display *p
2: *p = {data = 4, next = 0x21b40}
(gdb) step
16               return (p);
2: *p = {data = 17, next = 0x21b40}
1: p = (node_t *) 0x21b50
```

We found the bug:

Debugging A Program V

But there is another bug:

Input:

```
-1       (empty list)
17       (value to search for)
```

Output:

```
Segmentation fault (core dumped)
```

See if you can find the bug.