From C0 to C

Adapted from slides by Pat Virtue

Last time

END PART II

- Data structures

Today

Transition to C

- Code organization
- Contracts
- Compiling
- Memory management

Next

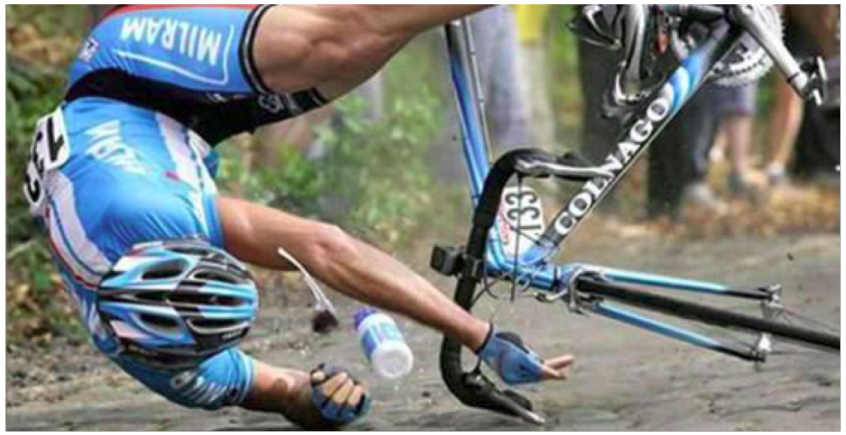
(Midterm)

C's Memory Model

C0

C

?



C0 Example for Today

simple.c0

```
#use <util>

/*** Interface ***/
int absval(int x)
/*@requires x > int_min(); @*/
/*@ensures \result >= 0; @*/ ;

struct point2d {
  int x;
  int y;
};

/*** Implementation ***/
int absval(int x)
//@requires x > int_min();
//@ensures \result >= 0;
{
  return x < 0 ? -x : x;
}
```

test.c0

```
#use <conio>

int main() {
  struct point2d* P
      = alloc(struct point2d);

  P->x = -15;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  print("x coord: ");
  printint(P->x);
  println("\n");
  print("y coord: ");
  printint(P->y);
  println("\n");
  return 0;
}
```

simple.c0 to simple.c

```
#use <util>

/*** Interface ***/
int absval(int x)
/*@requires x > int_min(); @*/
/*@ensures \result >= 0; @*/ ;

struct point2d {
  int x;
  int y;
};

/*** Implementation ***/
int absval(int x)
//@requires x > int_min();
//@ensures \result >= 0;
{
  return x < 0 ? -x : x;
}
```

```
#ifndef _SIMPLE_H_
#define _SIMPLE_H_

int absval(int x)
/*@requires x > INT_MIN; @*/
/*@ensures \result >= 0;  @*/ ;

struct point2d {
  int x;
  int y;
};
#endif
```

```
#include <limits.h>
#include "lib/contracts.h"
#include "simple.h"

int absval(int x) {
  REQUIRES(x > INT_MIN);
  int res = x < 0 ? -x : x;
  ENSURES(res >= 0);
  return res;
}
```

test.c0 to test.c

Eventually

```
#use <conio>

int main() {
  struct point2d* P
      = alloc(struct point2d);

  P->x = -15;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  print("x coord: ");
  printint(P->x);
  println("\n");
  print("y coord: ");
  printint(P->y);
  println("\n");
  return 0;
}
```

```
#include <stdlib.h>
#include "lib/xalloc.h"
#include <assert.h>
#include <stdio.h>
#include <stdbool.h>
#include "simple.h"

int main() {
  struct point2d* P
      = xmalloc(sizeof(struct point2d));

  P->x = -15;
  P->y = 0;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  printf("x coord: %d\n", P->x);
  printf("y coord: %d\n", P->y);

  free(P);

  return 0;
}
```

C Preprocessor

statements that are processed before compiling the actual C code

- #include

- #define

- #ifdef
- #else
- #endif
- #ifndef

C Preprocessor

#include

- Our code (double quotes, because we know where it is)
  - `#include "simple.h"`
- Course library code (double quotes, because we know where it is)
  - `#include "lib/hdict.h"`
- C system library code (< >, because the system knows where it is)
  - `#include <stdio.h>`

Rules (strict guidelines)

- Only include *.h files

C Preprocessor

Macro definitions

```
#define INT_MIN 0x8000000
```

Rules (strict guidelines)

- Use all CAPS for macro names
- Only #define constants

Buggy

```
#define INT_MAX INT_MIN ^ -1


INT_MAX / 2
```

C Preprocessor

Conditional compilation

```
#define DEBUG

#ifdef DEBUG
  printf(“Reached here\n”);
#endif
```

C Preprocessor

Conditional compilation

```
#ifdef TARGET_OS_MAC
  #include “arch/osx_optimizations.h”
  osx_optimize(code);
#else
  optimize(code);
#endif
```

C Preprocessor

Macro functions

```
#define MULT (x, y)  x * y


MULT(1+2, 3+4);
// Broken :(
```

Rules (strict guidelines)

- Don’t create macro functions

C Preprocessor

statements that are processed before compiling the actual C code

- #include

- #define

- #ifdef
- #else
- #endif
- #ifndef

How do we do contracts??

Contract in C

- Comments, e.g. //@requires, are just comments ☹
- We'll use macro functions for
  - REQUIRES
  - ENSURES
  - ASSERT

Sadly, no loop invariants

How do we do contracts??

Contracts in C

- lib/contracts.h

simple.c0 to simple.c

```
#use <util>

/*** Interface ***/
int absval(int x)
/*@requires x > int_min(); @*/
/*@ensures \result >= 0; @*/ ;

struct point2d {
  int x;
  int y;
};

/*** Implementation ***/
int absval(int x)
//@requires x > int_min();
//@ensures \result >= 0;
{
  return x < 0 ? -x : x;
}
```

```
#ifndef _SIMPLE_H_
#define _SIMPLE_H_

int absval(int x)
/*@requires x > INT_MIN; @*/
/*@ensures \result >= 0;  @*/ ;

struct point2d {
  int x;
  int y;
};
#endif
```

```
#include <limits.h>
#include "lib/contracts.h"
#include "simple.h"

int absval(int x) {
  REQUIRES(x > INT_MIN);
  int res = x < 0 ? -x : x;
  ENSURES(res >= 0);
  return res;
}
```

simple.c0 to simple.c

```
#use <util>

/*** Interface ***/
int absval(int x)
/*@requires x > int_min(); @*/
/*@ensures \result >= 0; @*/ ;

struct point2d {
  int x;
  int y;
};

/*** Implementation ***/
int absval(int x)
//@requires x > int_min();
//@ensures \result >= 0;
{
  return x < 0 ? -x : x;
}
```

```
#ifndef _SIMPLE_H_
#define _SIMPLE_H_

int absval(int x)
/*@requires x > INT_MIN; @*/
/*@ensures \result >= 0;  @*/ ;

struct point2d {
  int x;
  int y;
};
#endif
```

```
#include <limits.h>
#include "lib/contracts.h"
#include "simple.h"

int absval(int x) {
  REQUIRES(x > INT_MIN);
  int res = x < 0 ? -x : x;
  ENSURES(res >= 0);
  return res;
}
```

test.c0 to test.c

```
#use <conio>

int main() {
  struct point2d* P
      = alloc(struct point2d);

  P->x = -15;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  print("x coord: ");
  printint(P->x);
  println("\n");
  print("y coord: ");
  printint(P->y);
  println("\n");
  return 0;
}
```

```
#include <stdlib.h>
#include "lib/xalloc.h"
#include <assert.h>
#include <stdio.h>
#include <stdbool.h>
#include "simple.h"

int main() {
  struct point2d* P
      = xmalloc(sizeof(struct point2d));

  P->x = -15;
  P->y = 0;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  printf("x coord: %d\n", P->x);
  printf("y coord: %d\n", P->y);

  free(P);

  return 0;
}
```

Where are the files?

C0/C1 code we have been using:

- Our code
  - Usually in our current directory, e.g. exp-handout/dict.c0
- Course library code
  - Usually in lib directory, e.g. exp-handout/lib/stack_of_int.c0
- C0/C1 system library code
  - In some system directory we never see
  - We need: `#use <conio>`, `#use <util>`, etc.

Interfaces code mixed in with implementation code

- Essentially all in *.c0 files

Where are the files?

C code we will use:

- Our code
  - Usually in our current directory, e.g. exp-handout/dict.c
- Course library code
  - Usually in lib directory, e.g. exp-handout/lib/stack_of_int.c
- C system library code
  - In some system directory we never see
  - We need: `#include <stdio.h>`, `#include <stdlib.h>`, etc.

Interface and code in separate files

- Interface in header files (*.h)
- Implementation in c files (*.c)

simple.c0 to simple.c

```
#use <util>

/*** Interface ***/
int absval(int x)
/*@requires x > int_min(); @*/
/*@ensures \result >= 0; @*/ ;

struct point2d {
  int x;
  int y;
};

/*** Implementation ***/
int absval(int x)
//@requires x > int_min();
//@ensures \result >= 0;
{
  return x < 0 ? -x : x;
}
```

simple.h

```
#ifndef _SIMPLE_H_
#define _SIMPLE_H_

int absval(int x)
/*@requires x > INT_MIN; @*/
/*@ensures \result >= 0;  @*/ ;

struct point2d {
  int x;
  int y;
};
#endif
```

simple.c

```
#include <limits.h>
#include "lib/contracts.h"
#include "simple.h"

int absval(int x) {
  REQUIRES(x > INT_MIN);
  int res = x < 0 ? -x : x;
  ENSURES(res >= 0);
  return res;
}
```

Compiling our code

C0 compiles

- All code together
- Needs to know about system library code

```
$ cc0 lib/*.c0 simple.c0 test.c0
```

C compiles

- C files (*.c), separately
- Needs to know about system library code
- Compiler still need to know about interface code, i.e. header files (*.h)

```
$ gcc lib/*.c simple.c test.c
```

Note: We'll deal with compiler flags later

simple.c0 to simple.c

```
#use <util>

/*** Interface ***/
int absval(int x)
/*@requires x > int_min(); @*/
/*@ensures \result >= 0; @*/ ;

struct point2d {
  int x;
  int y;
};

/*** Implementation ***/
int absval(int x)
//@requires x > int_min();
//@ensures \result >= 0;
{
  return x < 0 ? -x : x;
}
```

simple.h

```
#ifndef _SIMPLE_H_
#define _SIMPLE_H_

int absval(int x)
/*@requires x > INT_MIN; @*/
/*@ensures \result >= 0;  @*/ ;

struct point2d {
  int x;
  int y;
};
#endif
```

simple.c

```
#include <limits.h>
#include "lib/contracts.h"
#include "simple.h"

int absval(int x) {
  REQUIRES(x > INT_MIN);
  int res = x < 0 ? -x : x;
  ENSURES(res >= 0);
  return res;
}
```

test.c0 to test.c

```
#use <conio>

int main() {
  struct point2d* P
      = alloc(struct point2d);

  P->x = -15;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  print("x coord: ");
  printint(P->x);
  println("\n");
  print("y coord: ");
  printint(P->y);
  println("\n");
  return 0;
}
```

```
#include <stdlib.h>
#include "lib/xalloc.h"
#include <assert.h>
#include <stdio.h>
#include <stdbool.h>
#include "simple.h"

int main() {
  struct point2d* P
      = xmalloc(sizeof(struct point2d));

  P->x = -15;
  P->y = 0;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  printf("x coord: %d\n", P->x);
  printf("y coord: %d\n", P->y);

  free(P);

  return 0;
}
```

simple.c0 to simple.c

```
#use <util>

/*** Interface ***/
int absval(int x)
/*@requires x >= int_min(); @*/
/*@ensures \result >= 0; @*/ ;

struct point2d {
  int x;
  int y;
};

/*** Implementation ***/
int absval(int x)
//@requires x > int_min();
//@ensures \result >= 0;
{
  return x < 0 ? -x : x;
}
```

simple.h

```
#ifndef _SIMPLE_H_
#define _SIMPLE_H_

int absval(int x)
/*@requires x > INT_MIN; @*/
/*@ensures \result >= 0;  @*/ ;

struct point2d {
  int x;
  int y;
};
#endif
```

simple.c

```
#include <limits.h>
#include "lib/contracts.h"
#include "simple.h"

int absval(int x) {
  REQUIRES(x > INT_MIN);
  int res = x < 0 ? -x : x;
  ENSURES(res >= 0);
  return res;
}
```

test.c0 to test.c

```
#use <conio>

int main() {
  struct point2d* P
      = alloc(struct point2d);

  P->x = -15;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  print("x coord: ");
  printint(P->x);
  println("\n");
  print("y coord: ");
  printint(P->y);
  println("\n");
  return 0;
}
```

```
#include <stdlib.h>
#include "lib/xalloc.h"
#include <assert.h>
#include <stdio.h>
#include <stdbool.h>
#include "simple.h"

int main() {
  struct point2d* P
      = xmalloc(sizeof(struct point2d));

  P->x = -15;
  P->y = 0;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  printf("x coord: %d\n", P->x);
  printf("y coord: %d\n", P->y);

  free(P);

  return 0;
}
```

Printing

C0

```
#use <conio>
print(“Hello x=”);
printint(x);
println(“ world, then a new line”);
```

C

```
#include <stdio.h>
printf(“Hello x=%d world, then a new line\n”, x);
```

test.c0 to test.c

```
#use <conio>

int main() {
  struct point2d* P
      = alloc(struct point2d);

  P->x = -15;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  print("x coord: ");
  printint(P->x);
  println("\n");
  print("y coord: ");
  printint(P->y);
  println("\n");
  return 0;
}
```

```
#include <stdlib.h>
#include "lib/xalloc.h"
#include <assert.h>
#include <stdio.h>
#include <stdbool.h>
#include "simple.h"

int main() {
  struct point2d* P
      = xmalloc(sizeof(struct point2d));

  P->x = -15;
  P->y = 0;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  printf("x coord: %d\n", P->x);
  printf("y coord: %d\n", P->y);

  free(P);

  return 0;
}
```

Memory Allocation

C0

```
int* x = alloc(int);
```

C

```
int* x = malloc(sizeof(int));
// Could return NULL if out of mem
```

C with #include “lib/xalloc.h”

```
int* x = xmalloc(sizeof(int));
// Assert fail if out of memory
```

test.c0 to test.c

```
#use <conio>

int main() {
  struct point2d* P
      = alloc(struct point2d);

  P->x = -15;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  print("x coord: ");
  printint(P->x);
  println("\n");
  print("y coord: ");
  printint(P->y);
  println("\n");
  return 0;
}
```

```
#include <stdlib.h>
#include "lib/xalloc.h"
#include <assert.h>
#include <stdio.h>
#include <stdbool.h>
#include "simple.h"

int main() {
  struct point2d* P
      = xmalloc(sizeof(struct point2d));

  P->x = -15;
  P->y = 0;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  printf("x coord: %d\n", P->x);
  printf("y coord: %d\n", P->y);

  free(P);

  return 0;
}
```

Let’s try to compile

We’ll add many flags to get as many warnings as we can

```
gcc -Wall -Wextra -Wshadow -Werror -std=c99 -pedantic -g -DDEBUG lib/*.c simple.c test.c
```

`gcc`	GNU compiler
`-Wall -Wextra -Wshadow`	Extra warnings
`-Werror`	Treat warnings as errors, i.e. don’t create output executable if there are any warnings
`-std=c99`	Follow 1999 C standard
`-pedantic`	Follow standard rigorously
`-g`	Add debugging info to output for use by tools like gdb and valgrind
`-DDEBUG`	#define DEBUG

Let’s try to compile

```
---in_class $ gcc -Wall -Wextra -Wshadow -Werror -std=c99 -pedantic -g -DDEBUG lib/*.c sim
ple.c test.c
test.c: In function ‘main’:
test.c:15:38: error: invalid application of ‘sizeof’ to incomplete type ‘struct point2d’
   struct point2d* P = xmalloc(sizeof(struct point2d));
                                      ^
test.c:16:4: error: dereferencing pointer to incomplete type ‘struct point2d’
   P->x = -15;
    ^
test.c:18:17: error: implicit declaration of function ‘absval’ [-Werror=implicit-function-
declaration]
   P->y = P->y + absval(P->x * 2);
                 ^
In file included from test.c:9:0:
test.c:19:25: error: ‘true’ undeclared (first use in this function)
   assert(P->y > P->x && true);   // if P->y uninitialized, may succeeed or fail
                         ^
test.c:19:25: note: each undeclared identifier is reported only once for each function it
appears in
cc1: all warnings being treated as errors
---in class $
```

test.c0 to test.c

```
#use <conio>

int main() {
  struct point2d* P
      = alloc(struct point2d);

  P->x = -15;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  print("x coord: ");
  printint(P->x);
  println("\n");
  print("y coord: ");
  printint(P->y);
  println("\n");
  return 0;
}
```

```
#include <stdlib.h>
#include "lib/xalloc.h"
#include <assert.h>
#include <stdio.h>
#include <stdbool.h>
#include "simple.h"

int main() {
  struct point2d* P
      = xmalloc(sizeof(struct point2d));

  P->x = -15;
  P->y = 0;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  printf("x coord: %d\n", P->x);
  printf("y coord: %d\n", P->y);

  free(P);

  return 0;
}
```

Preventing “double include”

Add C preprocessor conditional compile around header files to make sure you only include that code once

```
#include "simple.h"
#include "simple.h"
```

simple.h

```
#ifndef _SIMPLE_H_
#define _SIMPLE_H_

int absval(int x)
/*@requires x >= INT_MIN; @*/
/*@ensures \result >= 0;  @*/ ;

struct point2d {
  int x;
  int y;
};
#endif
```

Valgrind

Detecting memory problems

1. Use –g flag when compiling
2. Call valgrind with executable

```
valgrind ./a.out
```

```
==7308== Conditional jump or move depends on uninitialised value(s)
==7308==    at 0x4E87B83: vfprintf (vfprintf.c:1631)
==7308==    by 0x4E8F898: printf (printf.c:33)
==7308==    by 0x40089F: main (test.c:22)
==7308==
==7308== Use of uninitialised value of size 8
==7308==    at 0x4E8476B: _itoa_word (_itoa.c:179)
==7308==    by 0x4E8812C: vfprintf (vfprintf.c:1631)
==7308==    by 0x4E8F898: printf (printf.c:33)
==7308==    by 0x40089F: main (test.c:22)
```

Uninitialized values

C does not initialize values!!

- Fast!
- Dangerous!!

test.c0 to test.c

```
#use <conio>

int main() {
  struct point2d* P
      = alloc(struct point2d);

  P->x = -15;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  print("x coord: ");
  printint(P->x);
  println("\n");
  print("y coord: ");
  printint(P->y);
  println("\n");
  return 0;
}
```

```
#include <stdlib.h>
#include "lib/xalloc.h"
#include <assert.h>
#include <stdio.h>
#include <stdbool.h>
#include "simple.h"

int main() {
  struct point2d* P
      = xmalloc(sizeof(struct point2d));

  P->x = -15;
  P->y = 0;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  printf("x coord: %d\n", P->x);
  printf("y coord: %d\n", P->y);

  free(P);

  return 0;
}
```

test.c0 to test.c

```
#use <conio>

int main() {
  struct point2d* P
      = alloc(struct point2d);

  P->x = -15;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  print("x coord: ");
  printint(P->x);
  println("\n");
  print("y coord: ");
  printint(P->y);
  println("\n");
  return 0;
}
```

```
#include <stdlib.h>
#include "lib/xalloc.h"
#include <assert.h>
#include <stdio.h>
#include <stdbool.h>
#include "simple.h"

int main() {
  struct point2d* P
      = xmalloc(sizeof(struct point2d));

  P->x = -15;
  P->y = 0;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  printf("x coord: %d\n", P->x);
  printf("y coord: %d\n", P->y);

  free(P);

  return 0;
}
```

Valgrind

Detecting memory problems

1. Use –g flag when compiling
2. Call valgrind with executable

```
valgrind ./a.out
```

```
==7320== HEAP SUMMARY:
==7320==     in use at exit: 8 bytes in 1 blocks
==7320==   total heap usage: 2 allocs, 1 frees, 4,104 bytes allocated
==7320==
==7320== LEAK SUMMARY:
==7320==    definitely lost: 8 bytes in 1 blocks
==7320==    indirectly lost: 0 bytes in 0 blocks
==7320==      possibly lost: 0 bytes in 0 blocks
==7320==    still reachable: 0 bytes in 0 blocks
==7320==         suppressed: 0 bytes in 0 blocks
==7320== Rerun with --leak-check=full to see details of leaked memory
```

Memory Ownership

Memory leaks

- Allocated memory that is no longer used but taking up space on the heap
- As memory fills up, computer may slow down and eventually run out of memory

C0 would clean up (garbage collect) any allocated memory once no variables were pointing to it.

C requires you to free any memory you allocate.

Rules (strict guidelines)

- Software component that allocates memory is responsible for freeing it

test.c0 to test.c

```
#use <conio>

int main() {
  struct point2d* P
      = alloc(struct point2d);

  P->x = -15;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  print("x coord: ");
  printint(P->x);
  println("\n");
  print("y coord: ");
  printint(P->y);
  println("\n");
  return 0;
}
```

```
#include <stdlib.h>
#include "lib/xalloc.h"
#include <assert.h>
#include <stdio.h>
#include <stdbool.h>
#include "simple.h"

int main() {
  struct point2d* P
      = xmalloc(sizeof(struct point2d));

  P->x = -15;
  P->y = 0;
  P->y = P->y + absval(P->x * 2);
  assert(P->y > P->x && true);

  printf("x coord: %d\n", P->x);
  printf("y coord: %d\n", P->y);

  free(P);

  return 0;
}
```