COMP1511 17s2

— Lecture 12 —

Stack and Heap

~~Andrew Bennett~~
~~<andrew.bennett@unsw.edu.au>~~
Jashank Jeremy
<jashank.jeremy@unsw.edu.au>

review: structures
the stack; stack frames
the heap; dynamic allocation

Don’t panic!

practical exam: this Friday
you must arrive early. slots 1515–1630, 1620–1645
first slot cannot leave early; second slot is *corralled*.
arriving after 1630? you won’t be able to do the exam.
time-slot swapping: form coming out later today

assignment 1:
coming very soon…

Review: Structured Data


```
typedef struct _type-name {
    type member;
    [...]
} type-name;
```

a way to group together related data of differing types
we refer to the individual pieces of data
as fields or members

we have two new operators: `.` and `->`
which retrieve a member from
a structure or a structure pointer, respectively.

Review: `struct` and `typedef`

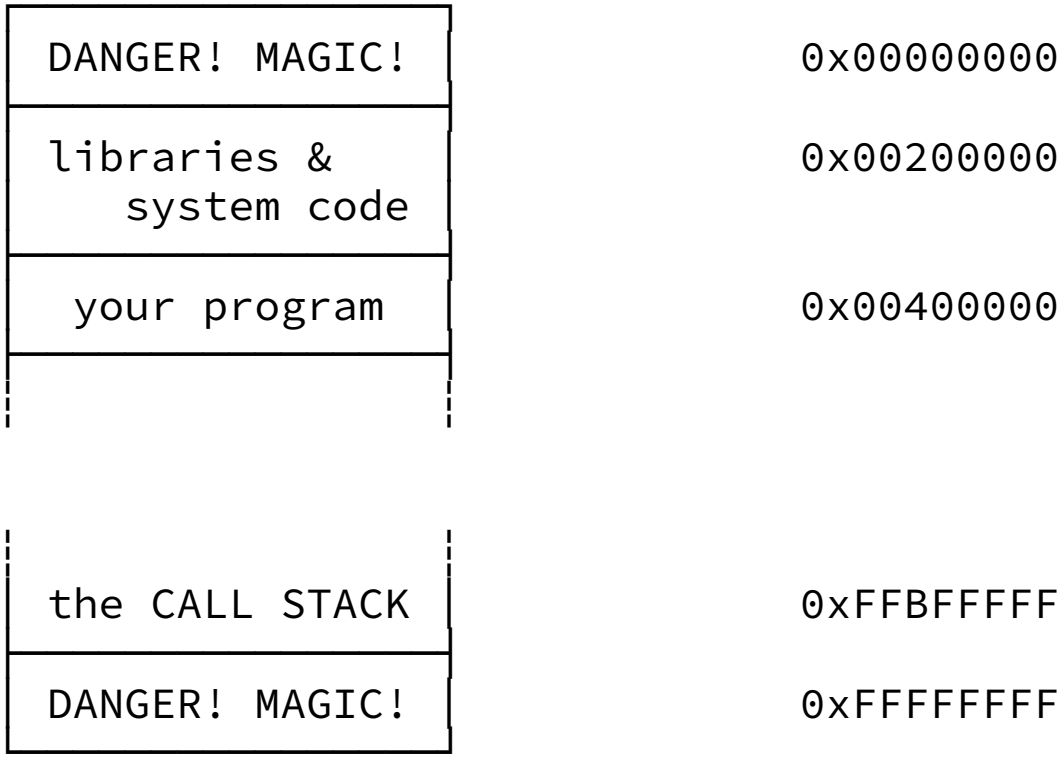
In this course, we forbid using tagged `structs`; use the `typedef`'d name.

`typedef` *existing-type new-type-name*;

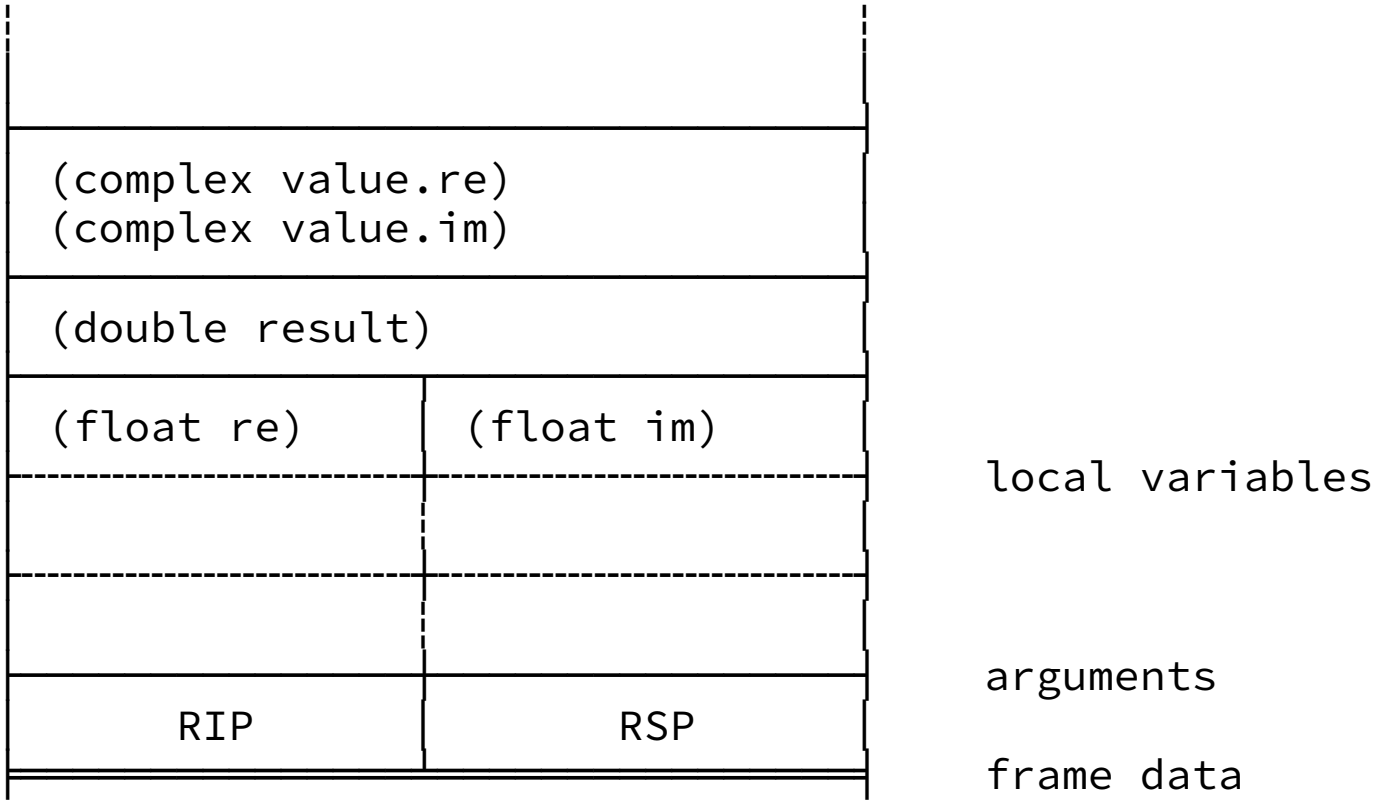
`typedef` lets us create our own types,
that can be shorter than types we already have

```
struct _student s;
student s;
```

Review: The Stack

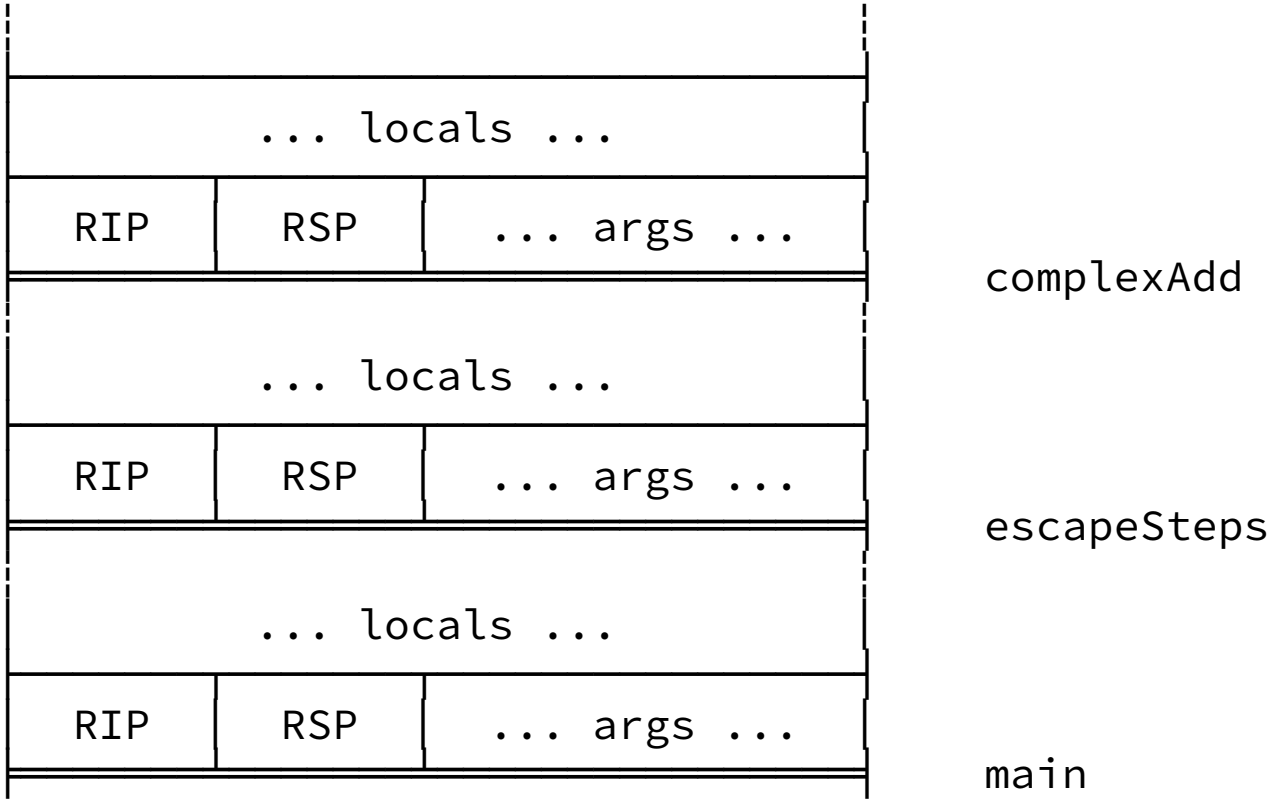


Review: The Stack



RIP: return instruction pointer
RSP: return stack pointer

Review: The Stack



The Stack

all our arguments and
local variables live here.

layout is mostly dependent
on how the compiler is feeling;
so long as the resultant code works,
we don't *really* care too much about the rules.

values here will only live
as long as the frame does
we can say a variable has a lifetime,
bounded by the stack frame

for more gory detail on how this works: COMP1521

Lifetimes


```
{
    double a = 0;                   //    \
    double b = 0;                   //    |  \
                                    //    |  |
    if (a * b > 2) {                //    |  |  \
        double result = a * b;      //    |  |  |
    }                               //    |  |  /
                                    //    /  /
}
```

Lifetimes


```
{
    double a = 0;                   //    \
    double *p;                      //    |  \
                                    //    |  |
    if (a > 2) {                    //    |  |
        p = &a;                     //    |  |
    } else {                        //    |  |
        double b = 0;               //    |  |  \
        p = &b;                     //    |  |  |
    }                               //    |  |  /
                                    //    |  |
    printf ("%lf\n", *p);           //    |  |  !!
                                    //    /  /
}
```

Lifetimes

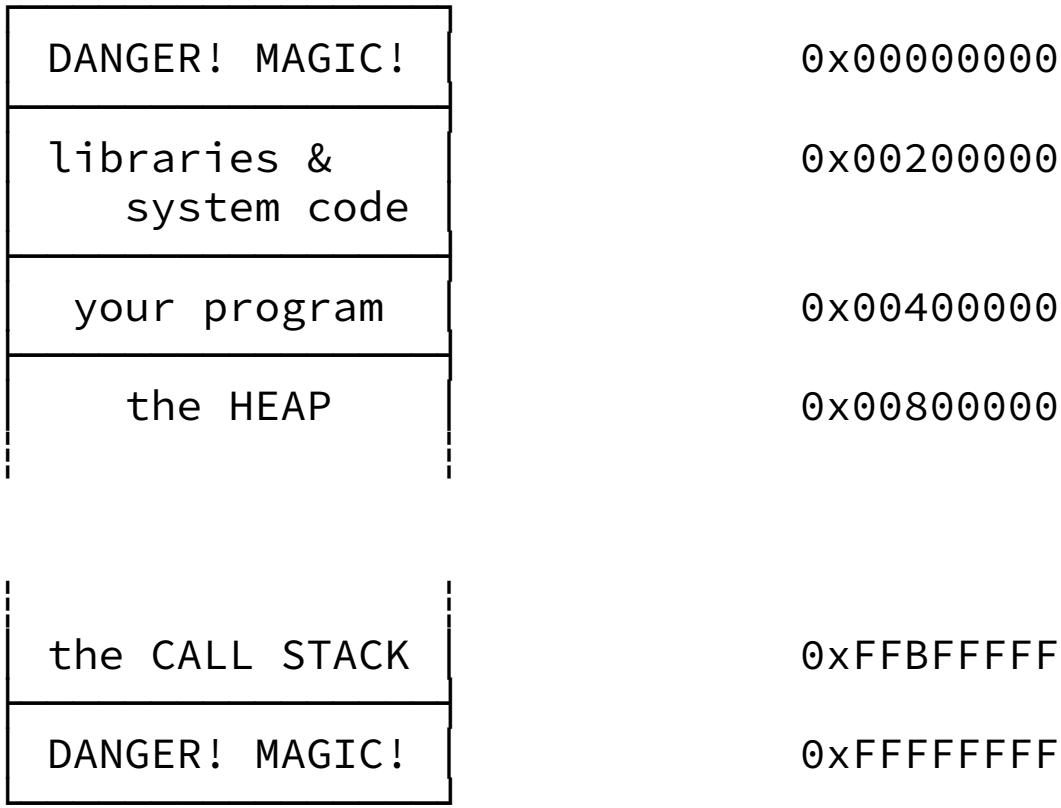

```
int main (int argc, char *argv[]) {
    double a = 0;                   //    \
    double *p;                      //    |  \
                                    //    |  |
    doSomething (&a, &p);           //    |  |
                                    //    |  |
    printf ("%lf\n", *p);           //    |  |  !!
                                    //    /  /
}

void doSomething (double *a, double **p) {
    double q = 0;                   //    |  |  \
    if (q > 2) {                    //    |  |  |
        *p = &q;                    //    |  |  |
    } else {                        //    |  |  |
        *p = a;                     //    |  |  |
    }                               //    |  |  /
}
```

Persistence

so how do we deal with this case? malloc, calloc, and free

```
void *malloc (unsigned int bytes);
void *calloc (unsigned int nThings, unsigned int thingSize);
void free (void *);
```

we prefer calloc.


DANGER! MAGIC!
0x00000000
libraries &
system code
0x00200000
your program
0x00400000
the HEAP
0x00800000
the CALL STACK
0xFFBFFFFF
DANGER! MAGIC!
0xFFFFFFFF