CSC 347 - Concepts of Programming Languages

Safety

Instructor: Stefan Mitsch

Learning Objectives

How do we make sure that we accesss memory correctly?

- Understand memory access through pointers
- Understand examples of unsafe behavior in C

Types in C

What do you make of this program?

```
int main() {
    int *p = (int*) malloc (sizeof(int));
    *p = 2123456789;

    printf ("(float)*p = %f\n", (float)*p);   /* loss of precision */
    printf ("*(float*)p = %f\n", *(float*)p); /* rubbish */

    int i = 2;
    char s[] = "three";

    printf ("i*s = %ld\n", i*(long)s);
}
```

```
$ clang -m32 typing-00.c && ./a.out
(float)*p = 2123456768.000000
*(float*)p = 96621069057346178268049192388430659584.000000
i*s = -1047484
```

Unsafe Memory Access

- Memory location contains data written at a given type (such as character array)
- The same memory location is read without permission or interpreted at an incompatible type (such as int)
- This is an *unsafe access*
- Scheme prevents unsafe access by *throwing an exception*

```
#;> (+ "dog" 1)
Error in +: expected type number, got '"dog"'.
```

Array Bounds

What do you make of the following program?

```c
int main () {
    float f = 10;
    int a[] = { 10 };
    short i = 10;
                        printf ("f=%f, a[0]=%d i=%d\n", f, a[0], i);
    a[-1] = 2123456789; printf ("f=%f, a[0]=%d i=%d\n", f, a[0], i);
    a[1]  = 2123456789; printf ("f=%f, a[0]=%d i=%d\n", f, a[0], i);
}
```

```
$ clang -m32 typing-03.c && ./a.out
f=10.000000, a[0]=10 i=10
f=10.000000, a[0]=10 i=32401
f=96621069057346178268049192388430659584.000000, a[0]=10 i=32401
```

Pointer Aliasing on the Stack

```c
int main() {
    int x = 2123456789;
    double y = x;
    printf ("x=%d, y=%f\n", x, y);
    double *p = &x;
    double z = *p;
    printf ("x=%d, z=%f\n", x, z);
}
```

```
$ gcc typing-06.c && ./a.out
x=2123456789, y=2123456789.000000
x=2123456789, z=38685644468023060038942720.000000
```

Pointer Aliasing on the Heap

```c
int main() {
    int* ip = (int*) malloc (sizeof(int));
    *ip = 10;
    free(ip);
    float* fp = (float*) malloc (sizeof(float));
    *fp = 10;
    printf ("*fp=%f, *ip=%d\n", *fp, *ip);
    printf (" fp=%p,  ip=%p\n", fp, ip);
}
```

```
$ gcc typing-07.c && ./a.out
*fp=10.000000, *ip=1092616192
 fp=0x1063010,  ip=0x1063010
```

Function Pointers

```c
void unsafeCommand () { printf ("ouch!\n"); }
void safeCommand ()   { printf ("hurray!\n"); }
int main () {
    int diff = &unsafeCommand - &safeCommand;
    void (*c) () = &safeCommand;
    c();
    c += diff;
    c();
}
```

```
$ clang -m32 typing-04.c && ./a.out
hurray!
ouch!
```

Function Pointers with Arguments

```c
void floatCommand (float f) { printf ("f=%f\n", f); }
void intCommand (int i)     { printf ("i=%d\n", i); }
int main () {
    int diff = (void*)&intCommand - (void*)&floatCommand;
    void (*c) (int) = &intCommand;
    int j = 2123456789;
    c(j);
    c -= diff;
    c(j);
}
```

```
$ clang -m32 typing-05.c && ./a.out
i=2123456789
f=96621069057346178268049192388430659584.000000
```

- C types are difficult to read, there are tools to help

Unsafety and Security

- Unsafety causes security problems

Safe Languages: Java

To be safe, Java does the following

- Disallows pointers into the stack
- Disallows pointer arithmetic
- Disallows explicit `free`
- Checks array bounds
- Checks potentially unsafe casts

Bounds Checking

```
class Typing04 {
  public static void main (String[] args) {
    Object[] bs = new Object[4];
    Object b = bs[-1];
  }
}
```

```
$ javac Typing04.java && java Typing04
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: -1
```

Checked Casts

```java
class A { int x; }
class B extends A { float y; }
class C extends A { char c; }

class Typing05 {
  static void f (B b) {
    A a = b;       /* upcast always safe */
  }
  static void g (A a) {
    B b = (B) a;   /* downcast must be checked */
  }
  public static void main (String[] args) {
      f (new B());
      g (new C());
  }
}
```

```
$ javac Typing05.java && java Typing05
Exception in thread "main" java.lang.ClassCastException: C cannot be cast to B
```

- Compare with dynamic cast in C++

Checked Array Assignment

```java
class A { int x; }
class B extends A { float y; }
class C extends A { char c; }

class Typing03 {
  public static void main (String[] args) {
      B[] bs = new B[1];
      A[] as = bs;
      as[0] = new C(); /* write must be checked */
      B b = bs[0];     /* reading always safe */
  }
}
```

```
$ javac Typing03.java && java Typing03
Exception in thread "main" java.lang.ArrayStoreException: C
```

- This is a design flaw; more in a later lecture

Summary

- Traditional systems languages are purposefully unsafe: Assembly, C, C++, Objective-C, C#-unmanaged, ...
- Recent application languages are meant to be safe: Java, Scheme, Javascript, Python, C#-managed, ...
- Recent systems languages attempt to isolate the unsafe bits: Rust, Go
- Even in safe languages, the overuse of dynamic checks allows program flaws to make it into production
- Overuse of `null` is considered a billion dollar mistake
- Which is better?
    - Security hole due to lack of null checks
    - Program crashes on null pointer reference