CS 32: Data Structures + Algorithms

Objective

Data abstraction

C++ Classes

Pointers, Dynamic Arrays, Resource Management

Linked Lists

Stacks and Queues

Inheritance and Polymorphism

OOP

Recursion

Templates, Iterators, STL

Algorithmic Efficiency

Sorting

Trees

Tree-based tables, Hash tables

Priority Queues, Heaps

Graphs

CS 31 Review

Pointers

- → Another way to implement pass by reference
- → Traverse arrays
- → Manipulate dynamic storage
- → Represent relationships in data structures

double	is a number
double &	is a reference to a double (another name for a pre-existing object)
double*	is a pointer to a double.

&x	generate a pointer to x
*p	the object that p points to

Reference parameters

→ used to make a function return multiple values

```
void polarToCartesian (double rho, double theta,
                       double xx, double yy)

{
    xx = rho * cos(theta);
    yy = rho * sin(theta);
}

int main()
{
    double r;
    double angle;

    .. .. get r and angle ...

    double x;
    double y;

    polarToCartesian (r, angle, x, y)
```

while x and y obtain the correct value, they poof when the function poofs.

What I'd like to do: change variable values OUTSIDE the function.

```
void polarToCartesian(double rho, double theta,
                      double& xx, double& yy)
```

double rho and double theta create brand new doubles that the values are copied into.

But double& xx and double& yy are references — essentially, they point to the same double as x/y.

So if I change the value of x, even xx changes.

$\boxed{\text{i.e.}}$ the parameter is just another name for the original argument. NO COPY IS MADE.

x : 5

angle : 0

xx : ??? : xx

yy : ??? : yy

5 : rho

0 : theta

Upon ending the function, xx and yy go away - but they do their job, i.e. alter the values of x and y.

Lecture 1 Part 1 — January 3

Undefined Behaviour

a[k] where k is out of bounds.

i/j where j = 0.

p→ [element undefined]

Uninitialized variable assumed to be 0.

Implementation-dependent behaviour

17/ -5	17% -5
-3 -4	2 -3

If there's no return statement in a function, (except void), it is undefined behaviour.

```
double f
{
    if ——
        return 0;
    if ——
        return 1;
}
```

probably will not give an error, although it will give a warning. Ensure return in the general function code.

int	-2 billion	to	2 billion
unsigned int (size_t)	0	to	4 billion

0 - 2 billion ⟶ treated the same way

```
for (int k = 0;  k < string.size() - 1;  k++)
```

unsigned [size_t] — signed

an expression containing a signed and an unsigned int always converts the signed to unsigned.

Therefore, if string.size = 0, the value returned is 4 billion.

```
for (int k = 0; k+1 < s.size(); k++)
```

↳ fixed version

Alternatively,

```
int ssize = string.size()
```

CONVERT BOTH POTENTIAL CASES TO INT AND ACT UPON THAT.

Lecture 1 Part 2 — January 3