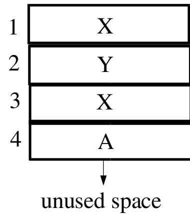
Stack



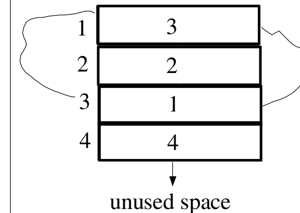
Environment = linear sequence of memory cells

what about if I call a function p many times ?
Activation records

CS330 Spring 2003
Copyright George Tzanetakis, University of Victoria

Pointers

- Is a storage location whose stored value is a reference to another object



In C: int *x;

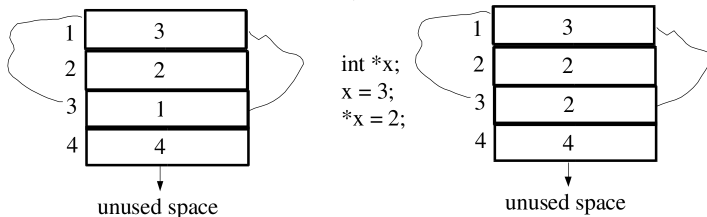
causes allocation of a pointer variable, but NOT the allocation of a object to which x points

Convention: 0 or NULL
Java: null, Pascal nil

CS330 Spring 2003
Copyright George Tzanetakis, University of Victoria

More pointers

- *x = 2;
  - the value pointed by x (a pointer variable) is 2



CS330 Spring 2003
Copyright George Tzanetakis, University of Victoria

Anonymous pointers

- void* x; (anonymous pointer variable x)
- x = (int) **malloc(**sizeof(int));
- Allocate a block of memory that fits an integer
- Dereferencing operator * (*x)
- Pointer type is also confusingly * (for example int* or float*)
- free(x);

CS330 Spring 2003
Copyright George Tzanetakis, University of Victoria

Dynamic allocation – Heap

- Memory used for calls to malloc, free is called the heap
- In C, C++ manual allocation is possible
- Java and ML don't allow allocation
- Static, Dynamic, Stack-based and Heap allocation

CS330 Spring 2003
Copyright George Tzanetakis, University of Victoria

Memory layout

static global area

stack

heap

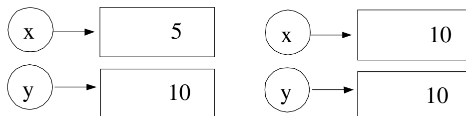
Heap storage can be released anywhere leaving "holes". Simple stack doesn't work. Functional languages automatically manage the heap. Java allows heap allocation but not deallocation.

Manual control of the heap results in very few cases in more efficient code but invites all kinds of unsafe operations.

CS330 Spring 2003
Copyright George Tzanetakis, University of Victoria

Variables – Storage semantics

- Value can be changed during execution
- name – location – value
- x = y



CS330 Spring 2003
Copyright George Tzanetakis, University of Victoria

l-value, r-value

- x = y
- x is the name of a location of a variable
- y is the value of the variable named y
- In ML distinction explicit:
  - x := !x +1;
  - x := !y;

CS330 Spring 2003
Copyright George Tzanetakis, University of Victoria

"Address of"operator in C

- int x;
- &x is the address of x and can be assigned to a pointer;
- For example:

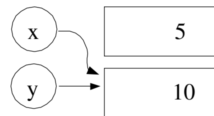
```
int x;
x = 10;
int *y = &x;
int z = *y;
int k = &y; (what does this one do ?)
```


The swamp of C

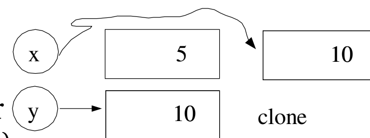
- Address arithmetic (pointers can be added subtracted like integers)
- mixing dereferencing and address of operators expressions and assignment can lead to some very confusing and complex situations

Pointer semantics

- Assignment by sharing



- Assignment by cloning done in Java by implicit pointers
  *x = *y
  (pointers under under the hood)



Value semantics – constants

- No location just a value
- Not necessarily known at compile time once computed never updated
- Examples: ML, Single assignment C
- In Java, keyword final is used for constants (gets only one final value) and static can be used when value can be computed prior to execution.

Function Definitions

- In virually all languages functions are essentially constants whose values are functions
- In ML: val square = fn(x:int) => x * x;

CS330 Spring 2003
Copyright George Tzanetakis, University of Victoria

Function Pointers in C

```
int gcd(int u, int v)
{
    if (v == 0) return u;
    else return gcd(v, u % v);
}

/* function variable – pointer syntax necessary otherwise prototype */
int (*gcdv)(int, int) = gcd;

/* can be called */

gcdv(15,10)
```


CS330 Spring 2003
Copyright George Tzanetakis, University of Victoria

Aliases

- Same thing bound to two different names at the same time

```
int *x, *y;
x = (int *) malloc(sizeof(int));
*x = 1;
y = x;
y = 2;    /* changes x although x doesn't appear in the assignment */
printf("%d\n", *x);
```


CS330 Spring 2003
Copyright George Tzanetakis, University of Victoria

Dangling references

- Location that has been dellocated from the environment but can still be accessed
- pointer to a deallocated object:

```
int *x, *y;
x = (int *) malloc(sizeof(int));
*x = 2;
y = x;
free(x);
printf("%d\n", *y);
```


CS330 Spring 2003
Copyright George Tzanetakis, University of Victoria

Garbage

- Eliminate dangling reference by never deallocating
- Garbage only wastes memory doesn't corrupt the program behavior

```
int *x;
...
x = (int *) malloc(sizeof(int));
x = NULL;
```


Copyright George Tzanetakis, University of Victoria

Garbage collection

- Lisp, Smalltalk, Java
- ML has a very efficient garbage collector
- There is a lot of interesting work in how to implement garbage collectors – some of you may learn about it when you write a Compiler

Copyright George Tzanetakis, University of Victoria

Hidden Conventions in C

A string constant is a shorthand for nameless array of characters

```
char hello[] = “hello”;
char hello[5];
hello = “hello”;
char *hello;
hello = “hello”
```

```
char hello[] = “hello”
char world[] = “world”
```

Try to write concatenate:

```
char * helloword;
helloworld = concatenate(hello, world);
```

(dynamically allocate memory when do we free it ?)

Copyright George Tzanetakis, University of Victoria

Storage semantics are conventions of the programmer

some string functions might allocate new memory for each call sometimes this can be inefficient so calling the function deletes the old memory and allocates new memory however wrong use of such function can result in the disaster

USING STRINGS IN C++ IS A BOOKKEEPING NIGHTMARE

Copyright George Tzanetakis, University of Victoria

Malloc

- No C program of any significance doesn't use malloc one way or another
- Know how much memory to allocate
- Note use any more memory than allocated
- Free memory when not required
- Not free memory before necessary
- Free only memory that's allocated
- Remember to check each allocation request

CS330 Spring 2003
Copyright George Tzanetakis, University of Victoria

Machines should work for people

(From A.Koening, B.Moo "Ruminations on C++")

Why do I care about language and abstraction ? Because I think that huge programs are ineffcieint, uncomfortable to work on, and impossible manage. I have neither seen, nor can imagine, a way to cope with a huge projects that attacks all of these problems. But if I can help point the way toward breaking up huge projects into bunches of little ones, I will be advancing the cause of the individual over the anonymous mass, and that of the human over the machine. We must be masters of our tools, not the other way aroun.

CS330 Spring 2003
Copyright George Tzanetakis, University of Victoria