Lecture 21

Pointers and WLPP

CS 241: Foundations of Sequential Programs
Fall 2009

Troy Vasiga et al
University of Waterloo

WL and MIPS

- Compiling WL to MIPS means that MIPS can do everything that a WL program can do
- The reverse is not true: mips.array
- WL cannot be used to solve A2P3 or A2P4

WLPP

- Allows arrays to be declared, initialized, allocated and destroyed
- Uses C++ syntax

Arrays

Nothing more than a pointer: starting address in $1.

Size is stored in $2.

Picture

A first WLPP Program

```
int wain(int *a, int n) {
   return *a;
}
```

- signature
- declaration of the array
- dereferencing
- syntax: C++-like (in fact, see the C++ embedding)
- semantics of the above program

What mips.array is doing

```
@cpu04.student[124]% java cs241.wlppc < ex1.wlpp > ex1.mips
@cpu04.student[125]% java mips.array ex1.mips
Enter length of array: 3
Enter array element 0: 10
Enter array element 1: 14
Enter array element 2: 19
```

Allocates memory on the *heap*, then calls wain with the address and the size as parameters.

A second example

```
int wain(int *a, int n) {
   return *(a+1);
}
```

Meaning:

A word about sugar:

Returning the last element in an array

```
int wain(int *a, int n) {

}
```

Simple mistakes and their consequences:

Printing elements of an array: Version 1

```
int wain(int *a, int n) {
  int i = 0;
  while (i < n) {
    println (*(a + i));
    i = i + 1;
  }
  return n;
}
```

Printing elements of an array: Version 2

Let's use a pointer variable.

```
int wain(int *a, int n) {
  int *p = NULL;
  p = a;
  while (p < a + n) {
    println (*p);
    p = p + 1;
  }
  return n;
}
```

What this program means:

Changes to the grammar

type → INT STAR
dcls → dcls dcl BECOMES NULL SEMI
statement → DELETE LBRACK RBRACK expr SEMI
factor → NULL
factor → AMP factor
factor → STAR factor
factor → NEW INT LBRACK expr RBRACK

Semantics of operators with pointers

type of a	type of b	meaning of +	meaning of -	*, /, %	<, >, etc
int	int				
int	int*				
int*	int				
int*	int*				

A good exercise

Write a WL program that prints and returns the maximum value in a given array.

Using new and delete

```
int wain(int* a, int n) {
  int* b = NULL;
  int pos = 0;
  int size = 0;
  size = n/2+1;

  b = new int[size];

  while (pos < n) {
    *(b + pos/2) = *(a+pos);
    pos = pos + 2;
  }
  pos = 0;
  while (pos < size) {
    println (*(b+pos));
    pos = pos + 1;
  }
  delete [] b;
  return n;
}
```

Memory allocation and memory problems

```
int wain(int *a, int n) {
  int *b = NULL;
  b = new int[3];
  *b = 9;
  *(b+1) = 10;
  *(b+2) = 11;
  return *(a+n);
}
```

Memory Revisited

Address-of: &

```
int wain(int *a, int n) {
  int *x = NULL;
  int y = 7;
  x = &y;
  println (*x);
  return (*x);
}
```

Back to compilation

How do these changes affect compilation of WLPP programs?

- type checking
- heap-management

A few words on heap management

- WLPP, C, C++ have explicit memory management
- delete is a promise to not use memory again
- never trust a programmer: see example

```
int wain(int *a, int n) {
  int* b = NULL;
  b = new int[3];
  *b = 7;
  *(b+1) = 17;
  *(b+2) = 345;
  delete [] b;
  println (*(b+1));
  return 0;
}
```

- Java doesn't trust programmers: it does implicit memory management. Programmers cannot directly access memory locations in Java.

Assignment Tips and Next Lecture

A11P1: Do some pointer arithmetic to multiply two matrices.

A11P2: Sort an array of numbers efficiently. You will most likely need another array: use new (and delete).

A11P3: Manage the heap: write init and delete given an implementation of new. More on this in the next two lectures.