CS212

Strings and Arrays
Fall 2005

Announcements

- P3:
  - actually posted (same as 2 weeks ago)
  - P3 extra credit TBA
  - grammar/template fixes (P3 FAQ)
- ACSU meeting today 5pm!
  - **The Many Careers of a Computer Scientist**
  - Upson Lounge (111, 1st floor)
  - Free food!

Summary

- Pointer review
- Pointer arithmetic reminder
- 1-D arrays
- Strings
- Pointers to pointers (`**p`, `***p`, ...)
- Multidimensional arrays
- Functions

Pointer Reminders

- Pointers:
  - Operators:
    - `&v`: address of `v`
    - `*p`: pointer `p` and the `p`'s *pointee*
  - Pointers:
    - `*p = exp`: L-value (insert `exp` @ `p`'s pointee)
    - `blah(*p)`: R-value (retrieve value from `p`'s pointee)
- Memory:
  - `malloc(int)`:
    - allocate space on heap
    - return address (type `void*`)
    - cast with `<type*>`
  - `free(voidpointer)`:
    - must free allocated heap space

Pointer Arithmetic Reminder

- Pointer arithmetic:
  - `malloc` returns address of first heap cell
  - `*(p+i)` advances to `(i+1)`*th* cell
- Picture:

1-D Arrays

- Connection between malloc, pointers, and arrays?
  - what does `malloc(int)` do?
  - what does `*(p+int)` do?

Bali Grammar

- Syntax:
  - expression → expart `[` expression `]`
  - expression → expart `=` expression
  - expart → `(`expression`)`
  - so, `(`expart `[` expression `])` → `(`expression`)`
- Semantics:
  - `Arr[k]` is semantically equivalent to `*(Arr + k)`
  - The assignment operator can also accept as a left value a dereferenced pointer (a pointer being operated on by `*`).

Simulating Strings

- Bali has *no* type `String`!
- Characters:
  - `char` and `char*` types
  - surround chars with `''`
  - no `int`-`char` casting or promotion
  - read `char`s with `readChar`
- Simulating a string:
  - array of characters
    - `<char*> malloc(size+1)`
    - Bali *pseudostrings* must terminate with `'\0'`
  - 1st cell's address is the address of the string
  - last cell's address is a flag
- Freeing memory:
  - `free(<void*> address_of_first_char)`
  - in real C, you can have strings on the stack

Basic Bali String Example

- Use **malloc**:

```
{ char * s; int size; }
{ size = 3;
  s = <char *> malloc(size+1);
  *(s+0) = 'a';
  *(s+1) = 'b';
  *(s+2) = 'c';
  *(s+3) = '\0'; }
```

- Can you use **(s[i])=*char***?
- Freeing strings: **free(<void*> s);**

Longer Example (C)

```
char* append(char* s1,char* s2,int size);

int main(int argc, char* argv[]) {
  char* s1; char* s2; char* s;
  int L1; int L2;
  int i; int size;

  L1 = 3; L2 = 2; size=L1+L2;

  s1=(char*)malloc((L1+1)*sizeof(char));
  s2=(char*)malloc((L2+1)*sizeof(char));

  *(s1+0)='a';
  *(s1+1)='b';
  *(s1+2)='c';
  *(s1+3)='\0';
  *(s2+0)='d';
  *(s2+1)='e';
  *(s2+2)='\0';

  s=append(s1,s2, size);
  for (i = 0; *(s+i) != '\0'; i = i+1)
    printf("%c", *(s+i));

}
```

```
char* append(char* s1,char* s2,int size) {
  char* s;
  int i;
  int j;

  s = (char*)malloc((size+1)*sizeof(char));

  for (i = 0; *(s1+i) != '\0'; i = i+1)
    *(s+i) = *(s1+i);

  for (j = 0; *(s2+j) != '\0'; j = j+1)
    *(s+i+j) = *(s2+j);

  *(s+i+j) = '\0';

  return s;
}
```


Pointer to Pointers

- Grammar allows **int** p;**
- What does that mean?
  - **p** is a pointer to a pointer to an int
  - **p→*p→int**
  - also: **(int *)***
- Example:

```
int i; int* q; int **p;
i = 10;
q = &i;
p = &q;
printf("%s%i\n","**p:  ",**p);
// output? picture?
```


More Syntax

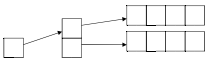
- ANSI C:

```
int main(int argc, char* argv[]) {
   int** p;
   p = (int **)malloc(sizeof(int*));
   *p = (int *)malloc(sizeof(int));
   **p = 10;
   printf("%s%i\n","**p:  ",**p);
}
```

- Arrays and Pointers:
  - array: **a[i][j][k]**
  - pointers: ***(*(*(a+i) + j) + k)**
  - in Bali? the answer is...

Bali Pointers to Pointers

Example:



```
int **p;
p = <int**>malloc(2);

*(p+0) = <int*>malloc(4);
*(p+1) = <int*>malloc(4);

*(*(p+1) + 0) = 10;
*(*(p+1) + 1) = 11;
*(*(p+1) + 2) = 12;
*(*(p+1) + 3) = 13;
// need to free *p, *(p+1), p
```


C Version

```
int main(int argc, char* argv[]) {

  int **p;
  p = (int **)malloc(2*sizeof(int*));
  *(p+0) = (int *)malloc(4*sizeof(int));
  *(p+1) = (int *)malloc(4*sizeof(int));

  // row 0:
  *(*(p+0) + 0) = 0;
  *(*(p+0) + 1) = 1;
  *(*(p+0) + 2) = 2;
  *(*(p+0) + 3) = 3;

  // row 1:
  *(*(p+1) + 0) = 10;
  *(*(p+1) + 1) = 11;
  *(*(p+1) + 2) = 12;
  *(*(p+1) + 3) = 13;

  int i, j;
  for ( i = 0 ; i < 2 ; i++ ) {
    for ( j = 0 ; j < 4 ; j++ )
      printf("%i%s",*(*(p+i)+j)," ");
    printf("\n");
  }

  free(*p);
  free(*(p+1));
  free(p);
  return 0;
}
```


Functions

- Some language notes:
  - pass by value
  - for pointers, send addresses (which are values)
- Example (see also longer string example):

```
int f(int* q);
int main( ) {
   { int* p; int i; }
   { p = &i;
     f(p);
     print i;
     return 0; }
}
int f(int* q) {
   { }
   { *q = 4;
     return 0; }
}
```