CSE 452: Programming Languages

Subprograms

Outline

- Subprograms
  - Parameter passing
  - Type checking
  - Using multidimensional arrays as parameters
  - Using subprograms as parameters
  - Overloaded subprograms
  - Generic subprograms
  - Implementation

Parameter Passing

- Pass-by-value
- Pass-by-result
- Pass-by-value-result
- Pass-by-reference
- Pass-by-name

Parameter Passing in PL

- Fortran
  - Always use inout-mode model of parameter passing
  - Before Fortran 77, mostly used pass-by-reference
  - Later implementations mostly use pass-by-value-result
- C
  - mostly pass by value
  - Pass-by-reference is achieved using pointers as parameters

```
int *p = { 1, 2, 3 };
void change( int *q) {
        q[0] = 4;
}
main() {
        change(p);      /*  p[0] = 4 after
   calling the change function */
}
```


Parameter Passing in PL

- C
  - Pass-by reference: value of pointer is copied to the called function and *nothing is copied back*

```
#include <stdio.h>
void swap (int *p, int *q)
{
        int *temp;
        temp = p;
        p = q;
        q = temp;
}
main() {
        int p[] = {1, 2, 3};
        int q[] = {4, 5, 6};
        int i;
    swap (p, q);
}
```


Parameter Passing in PL

- C++
  - Includes a special pointer type called a reference type

```
void GetData(double &Num1, const int &Num2) {
  int temp;
  for (int i=0; i<Num2; i++) {
       cout << "Enter a number: ";
       cin >> temp;
       if (temp > Num1)
            { Num1 = temp; return; }
  }
```

  - Num1 and Num2 are passed by reference
  - const modifier prevents a function from changing the values of reference parameters
  - Referenced parameters are implicitly dereferenced
  - Why do we need a constant reference parameter?

Parameter Passing in PL

- Ada
  - Reserved words: in, out, in out (in is the default mode)

```
procedure Iamp(A : in out Integer; B : in Integer; C : in Integer );
```

  - out mode can be assigned but not referenced
  - in mode can be referenced but not assigned
  - in out can be both referenced and assigned
- Fortran
  - Semantic modes are declared using Intent attribute

```
Subroutine Iamp(A, B, C)
  Integer, Intent(Inout) :: A
  Integer, Intent(In) :: B
  Integer, Intent(Out) :: C
```


Parameter Passing in PL

- Perl
  - Actual parameters are implicitly placed in a predefined array named @_

```
sub foo {
    local $i, $a=0, $b = 1;
    for ($i=0; $i<scalar(@_); $i++) {
        $a = $a + $_[$i];
        $b = $b * $_[$i];
    }
    return ($a, $b);
}
...
($a, $b) = foo(1, 2, 3);
```


Type Checking

- Ansi C: users can choose whether parameters should be type-checked

```
#include <stdio.h>

double count1(x)
    double x;                   // avoids type checking
{   return x * 2;   }           // may generate nonsense: count1(y)

double count2(double x)         // prototype method
{   return x * 2; }             // can coerce actual params: count2(y)

main() {
    double x = 30.0;
    int y = 30;
    printf("count1 : %f\n", count1(x));
    printf("count2 : %f\n", count2(x));
    printf("count1 : %f\n", count1(y));
    printf("count2 : %f\n", count2(y));
```

Output:

```
count1 : 60.000000
count2 : 60.000000
count1 : 0.000000
count2 : 60.000000
```