Assignment 2 of CE2004, Principles of Programming Languages

Due day: 21st June 2007

P.S.:

(1) You need to type your answers and print them out in paper. And then hand the answer sheets to TAs.

(2) Late submission will not be accepted.

(3) Copying other student's answers is strictly prohibited.

---

(1) (5 points) Dynamic type binding is closely related to implicit heap-dynamic variables. Explain this relationship.

(2) (10 points) Consider the following program:

```
procedure Main is
  X, Y, Z : Integer;
  procedure Subl is
     A,Y,Z : Integer;
     procedure Sub2 is
     A,B,Z : Integer;
     begin  -- of Sub2
       . . .
     end;   -- of Sub2
  begin – of Subl
    . . .
  end;  -- of Subl
  procedure Sub3 is
     A,X,W : Integer;
     begin  -- of Sub3
       . . .
     end;   -- of Sub3
begin  -- of Main
 . . .
end;   -- of Main
```

List all the variables, along with the program units where they are declared, that are visible in the bodies of `Subl`, `Sub2`, and `Sub3`, assuming static scoping is used.

(3) (10 points) Consider the following skeletal C program:

```
void funl(void); /* prototype */
void fun2(void); /* prototype */
void fun3(void); /* prototype */
void main() {
int a, b, c;
. . .
}
void funl(void) {
int b, c, d;
. . .
}
void fun2(void) {
int c, d, e;
. . .
}
void fun3(void) {
int  d, e, f;
. . .
}
```

Given the following calling sequences and assuming that dynamic scoping is used, what variables are visible during execution of the last function called? Include with each, visible variable the name of the function in which it was defined.

(a) `main` calls `funl; funl` calls `fun2; fun2` calls `fun3.`

(b) `main` calls `funl; funl` calls `fun3.`

(c) `main` calls `fun2; fun2` calls `fun3; fun3` calls `funl.`

(d) `main` calls `fun3; fun3` calls `funl.`

(e) `main` calls `funl; funl` calls `fun3; fun3` calls `fun2.`

(f) `main` calls `fun3; fun3` calls `fun2; fun2` calls `funl.`

(4) (10 points) Consider the following program:

```
procedure Main is
   X, Y, Z : Integer;
   procedure Subl is
     A, Y, Z : Integer;
   begin -- of Subl
    . . .
   end;   -- of Subl
   procedure Sub2 is
     A, B, Z : Integer;
   begin -- of Sub2
    . . .
   end;   -- of Sub2
   procedure Sub3 is
     A, X, W : Integer;
   begin -- of Sub3
    . . .
   end;   -- of Sub3
begin  -- of Main
 . . .
end;   --  of Main
```

Given the following calling sequences and assuming that dynamic scoping is used, what variables are visible during execution of the last subprogram activated? Include with each visible variable the name of the unit where it is declared.

(a) `Main` calls `Subl; Subl` calls `Sub2; Sub2` calls `Sub3.`

(b) `Main` calls `Subl; Subl` calls `Sub3.`

(c) `Main` calls `Sub2; Sub2` calls `Sub3; Sub3` calls `Subl.`

(d) `Main` calls `Sub3; Sub3` calls `Subl.`

(e) `Main` calls `Subl; Subl` calls `Sub3; Sub3` calls `Sub2.`

(f) `Main` calls `Sub3; Sub3` calls `Sub2; Sub2` calls `Subl.`

(5) (5 points) What significant justification is there for the `->` operator in **C** and **C++**?

(6) (10 points) Multidimensional arrays can be stored in row major order, as in C++, or in column major order, as in FORTRAN. Develop the access functions for both of these arrangements for three-dimensional arrays.

P.S.: Let the subscript ranges of the three dimensions be named `min(1)`, `min(2)`, `min(3)`, `max(1)`, `max(2)`, and `max(3).` Let the sizes of the subscript ranges be `size(1)`, `size(2)`, and `size(3).` Assume the element size is 1.

(7) (10 points) In the Burroughs Extended ALGOL language, matrixes are stored as a single-dimensioned array of pointers to the rows of the matrix, which are treated as single-dimensioned arrays of values, what are the advantages and disadvantages of such a scheme?

(8) (10 points) In the following **C** program excerpt,

(a) what is the result printed out by the statement commented as ``Location 6''?

(b) what is the result printed out by the statement commented as ``Location 11''?

(c) what mistake does it make?

```
void bar()
{  char *p,*q;              /*Location 1*/


   q=malloc(1);             /*Location 2*/
   p=q;                      /*Location 3*/
   *q='h';                  /*Location 4*/
   *p='e';                  /*Location 5*/
   printf("%c",*q);         /*Location 6*/
   q=malloc(1);             /*Location 7*/
   p=q;                      /*Location 8*/
   *q='r';                  /*Location 9*/
   *p='o';                  /*Location 10*/
   printf("%c",*q);         /*Location 11*/
}
```

(9) (10 points) In the following **C** program,

(a) Right after the execution of the statement commented as ``Location 2'' is finished, do variable `f` and `i` have the same value?

(b) Right after the execution of the statement commented as ``Location 4'' is finished, do variable `f` and `i` have the same value?

(c) Explain the results of (a) and (b).(P.S.: You can execute this program and observe the results to get your answers.)

```
#include<stdio.h>

main()

{

  float f;

  int   i;

  f=0;

  i=0.01;


  printf("%d\n",i);         /*Location 1*/

  f=i;                      /*Location 2*/

  printf("%f\n",f);         /*Location 3*/

  f=1.234;

  i=f;                      /*Location 4*/

  printf("%f\n",f);         /*Location 5*/

  printf("%d\n",i);         /*Location 6*/

}
```

(10) (10 points) In the following **C** program excerpt,

(a) what security problem does it have?

(b) why does this problem happen?

```
#define BufferSize 60
char *poi=”I will never miss a PPL class.”;
char sentence[BufferSize];
int ppp(char *s, char *d, unsigned len)
{ unsigned i;
  for( i=0 ;i<len; ++i )
  {
    *(d+i)=*(s+i);
  }
}

void goo(char *s, char *d, int length)
{
   if(length<BufferSize)
     ppp(poi,sentence,length);
}
```

(11) (10 points) In the following **C** program excerpt,

(a) what security problem does it have?

(b) why does this problem happen?

```
void candy()
{
    char CharArray[100];
    char *p;

    p=CharArray;
    while((*(p)=getchar())!=EOF)
    {
        ++p;
    }
}
```