CS302-2023-24: Quiz #2

Duration : 60 minutes Max Marks : 65

- No explanations will be provided. In case of a doubt, make suitable assumptions and justify.
- You are allowed to use a single A4 sheet of hand-written (not xeroxed) notes.
- Please write only the final answers in your answer sheet. Rough work may be done at the back. Extra material (or multiple answers of the same questions) will receive negative marks.
- Answers must be in ink. Answers written using pencil will not be evaluated.

(1) Consider an improved version of a part of the SDTS for processing C declarations that we have studied in the class. Let $Item.bt$ and $Item.dt$ represent the base type and the derived type of an item (represented by $Item$) occurring in a list in a type declaration statement. Given the inherited attribute $Item.bt$, the following SDTS computes the synthesized attribute $Item.dt$. The productions for the rest of the non-terminals have the usual actions and are not shown here. For example the value of $Item.bt$ is inherited in the rules not shown here. The value of token num is represented by the attribute $num.val$.

The base cases of non-terminals $ListDim$ and $ListStar$ create type expressions in which an unknown type is represented by θ. Its value becomes known in the productions for $Item$. A call $replace(E, \theta, \tau)$ returns a type expression obtained by replacing the occurrence of θ in E by the type expression τ. **(5+5+6=16)**

$Item \rightarrow id\ ListDim$	$\{Item.dt = replace(ListDim.dt, \theta, Item.bt)\}$
$Item \rightarrow ListStar\ \ id$	$\{Item.dt = replace(ListStar.dt, \theta, Item.bt)\}$
$Item \rightarrow ListStar\ \ id\ \ ListDim$	$\{\tau = replace(ListStar.dt, \theta, Item.bt)$ $Item.dt = replace(ListDim.dt, \theta, \tau)\}$
$Item \rightarrow (\ \ ListStar\ \ id\ \ )\ \ ListDim$	$\{\tau = replace(ListDim.dt, \theta, Item.bt)$ $Item.dt = replace(ListStar.dt, \theta, \tau)\}$
$Item \rightarrow ListStar_1\ \ (\ \ ListStar_2\ \ id\ \ )\ \ ListDim$	$\{\tau_1 = replace(ListStar_1.dt, \theta, Item.bt)$ $\tau_2 = replace(ListDim.dt, \theta, \tau_1)$ $Item.dt = replace(ListStar_2.dt, \theta, \tau_2)\}$
$ListStar \rightarrow *$	$\{ListStar.dt = pointer(\theta)\}$
$ListStar_1 \rightarrow *\ \ ListStar_2$	$\{ListStar_1.dt = pointer(ListStar_2.dt)\}$
$ListDim \rightarrow [\ \ num\ \ ]$	$\{ListDim.dt = array(num.val, \theta)\}$
$ListDim_1 \rightarrow [\ \ num\ \ ]\ \ ListDim_2$	$\{ListDim_1.dt = array(num.val, ListDim_2.dt)\}$

Consider the C declaration `int **x[10][20], (**y)[10][20], ** (**z)[10][20];`

(a) For variable x, give the values of $ListStar.dt$, $ListDim.dt$, and $Item.dt$, for the non-terminals occurring in production $\boxed{Item \rightarrow ListStar\ id\ ListDim}$.

(b) For variable y, give the values of $ListStar.dt$, $ListDim.dt$, and $Item.dt$, for the non-terminals occurring in production $\boxed{Item \rightarrow (\ ListStar\ id\ )\ ListDim}$.

(c) For variable z, give the values of $ListStar_1.dt$, $ListStar_2.dt$, $ListDim.dt$, and $Item.dt$, for the non-terminals occurring in production $\boxed{Item \rightarrow ListStar_1\ (\ ListStar_2\ id\ )\ ListDim}$.

Answer:

(a) For variable x, $ListStar.dt = pointer(pointer(\theta))$,
$ListDim.dt = array(10, array(20, \theta))$,
$Item.dt = array(10, array(20, pointer(pointer(int))))$.

(b) For variable y, $ListStar.dt = pointer(pointer(\theta))$,
$ListDim.dt = array(10, array(20, \theta))$,
$Item.dt = pointer(pointer(array(10, array(20, int))))$.

(c) For variable z, $ListStar_1.dt = pointer(pointer(\theta))$,
$ListStar_2.dt = pointer(pointer(\theta))$,
$ListDim.dt = array(10, array(20, \theta))$,
$Item.dt = pointer(pointer(array(10, array(20, pointer(pointer(int))))))$.

Evaluation rubrics: 1 marks for *ListStar.dt* and *ListDim.dt*, and 3 marks for *Item.dt*.

Note: It's fine if the students have drawn the expression trees. Besides, if `int` is left as *Item.bt*, we will allow it because it was missing from the declarations. The name "replace" causes confusion in that it could mean changing the argument E. So if some students have replaced θ by *Item.bt* that should be allowed too.

---

(2) Resolve the expressions occurring in procedures R, S, and T, under static and dynamic scoping rules by identifying the instances of variables that they use. If expression $x + y$ uses variable x defined in procedure A and variable y defined in procedure B, then after scope resolution, the expression is described by $A : x + B : y$.

All declarations in any scope appear before any executable statement in the scope. **(18)**

```
void P()
{  int a, b, c, d;
   void Q()
   {  int a, c;
      void R() { int a, b; a+b; c+d;}
      void S()
      {  int b, d;
         void T() { int b, c; a+b; c+d;}
         a+b; c+d;
      }
   }
}
```

Write your answer by filling in the blank entries in this table. For dynamic scoping, use the call sequence *main* $\rightarrow \ldots \rightarrow P \rightarrow Q \rightarrow S \rightarrow T \rightarrow R$.

Procedures containing the expressions	Expressions after scope resolution			
	Static Scoping		Dynamic Scoping	
	$a+b$	$c+d$	$a+b$	$c+d$
Procedure R				
Procedure S				
Procedure T				

Answer:

The expressions are resolved as following:

Procedures containing the expressions	Expressions after scope resolution			
	Static Scoping		Dynamic Scoping	
	$a+b$	$c+d$	$a+b$	$c+d$
Procedure R	$R:a+R:b$	$Q:c+P:d$	$R:a+R:b$	$T:c+S:d$
Procedure S	$Q:a+S:b$	$Q:c+S:d$	$Q:a+S:b$	$Q:c+S:d$
Procedure T	$Q:a+T:b$	$T:c+S:d$	$Q:a+T:b$	$T:c+S:d$

Evaluation rubrics: 1.5 marks per entry.

(3) Draw a picture of the control stack for the following program showing all activation records before any activation record is popped off. Assume that the control stack grows downwards. Use the given activation records structure. For each variable in an activation record, show the value of the variable along with its name. If the value of a field is not available, leave it as a blank entry in the activation record and do not skip it. **(15)**

Program

```
1. int z;
2. int main()
3. {
4.     int w = 1;
5.     z = 15;
6.     recur (z - w);
7.     cout << z;
8.     return 0;
9. }
```

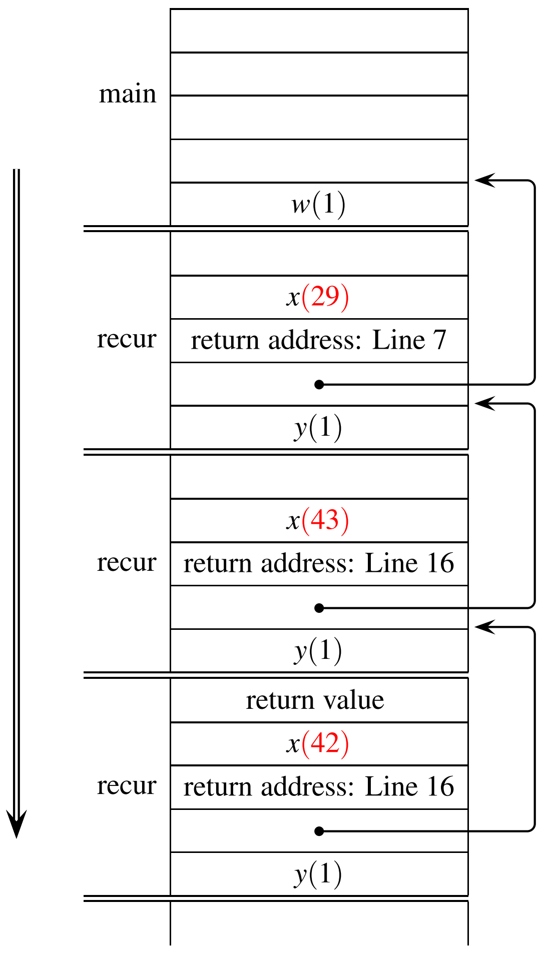
```
10. void recur(int x)
11. {  int y = 1;
12.    if (x <= 30)
13.    {
14.        x = x + z;
15.        recur (x - y);
16.        z = x * 5;
17.    }
18. }
```

Activation record structure

return value
param n
...
param 1
return address (in terms of source line)
control/dynamic link
local 1
...
local m

Answer:

The control stack just before any activation record is popped off, is as shown below.



Evaluation rubrics: 1 mark per correct entry in the three activation records for procedure `recur` (no marks for `main`). Deduct 1 mark for every additional entry.

The return value is meaningless because the procedures do not have a return statement. If the blank entries contain the names of the fields, then that is fine too. Also, if some people have written the previous line number for return address (Line 6 and Line 15), we should allow it because the line number of next instruction is ambiguous when we talk about source.

There is a suggestion that the return value of main (0) should be shown in the activation record but that is wrong because the execution has not reached the end of main before popping any activation record.

Also, some of you have optimized away some fields (such as return value). However, the question clearly says that no field should be skipped. So this is not acceptable.

---

(4) Consider the following program. Assume that you have a compiler that allows you to choose any parameter passing mechanism (and thus you are not constrained by the mechanism used by C++ although the program uses C++ syntax). Use the last digit of you roll number as the initial value of variable *a* and show the output of the program under the following parameter passing mechanisms: (a) Call by value, (b) call by reference, (c) call by value-result, and (d) call by name. **(16)**

```
int a, b;
void main( )
{  a = ...;
   f(a, a+1);
   cout << a << ", " << b << endl;
}
```

```
void f(int c, int d)
{  a = a + 15;
   a = a + d;
   c = c + d + 105;
   b = c - 1;
}
```

Answer:

The output values for different possible last digits as values of variable *a*, are as shown below

a	value	value-result	reference	name
0	16,105	106,105	122,121	168,167
1	18,107	108,107	125,124	172,171
2	20,109	110,109	128,127	176,175
3	22,111	112,111	131,130	180,179
4	24,113	114,113	134,133	184,183
5	26,115	116,115	137,136	188,187
6	28,117	118,117	140,139	192,191
7	30,119	120,119	143,142	196,195
8	32,121	122,121	146,145	200,199
9	34,123	124,123	149,148	204,203

Evaluation rubrics: 2 marks per correct entry. There is no need to give any explanation of how the answer has been arrived at.

Best Luck!