

Quiz-1 (September 23, 2020)

Total points 12/18 ?

Write your name and ID correctly.

There are 18 questions in this quiz.

Answer the questions and submit your responses.

The respondent's email address (**f20181119@pilani.bits-pilani.ac.in**) was recorded on submission of this form.

0 of 0 points

Name *

Shreyas Bhat Kera

ID *

2018A7PS1119P

Questions 1-18

12 of 18 points

The array variables in Perl and Javascript programming language

1/1

- ☒ can grow
- ☐ support negative subscript
- ☐ none of these
- ☐ are sparse



Consider the array variables A, B, C and D declared in C programming language as shown below. The arrays A, B, C and D are respectively categorized as

1/1

```
#include <stdio.h>
#include<stdlib.h>
static int A[10];
int main()
{
    int n, B[10];
    int C[n];
    int *D;
    D=(int*) malloc(sizeof(int)*n);
    return 0;
}
```

- ☒ static array, fixed stack-dynamic array, stack-dynamic array and fixed heap-dynamic array
- ☐ fixed heap-dynamic array, static array, stack-dynamic array and fixed stack-dynamic array
- ☐ None of these
- ☐ static array, stack-dynamic array, fixed stack-dynamic array and fixed heap-dynamic array
- ☐ static array, fixed heap-dynamic array, stack-dynamic array and fixed stack-dynamic array



Consider the statements for an array (say A) (1) the subscript ranges are bound at compile time (2)The actual physical location is allocated only if the function containing definition of A is called (3) The size of the array is not required to be known until the array is used. Which statements define the fixed stack-dynamic array? 0/1

- ☐ None of these
- ☐ Statements 2 and 3
- ☐ All statements 1, 2 and 3
- ☐ Statements 1 and 2
- ☒ Statements 1 and 3

Correct answer

- ☒ Statements 1 and 2

What is the purpose of bound checking? 1/1

- ☐ Bound checking is done for preventing users from accessing data stored in an array.
- ☐ Bound checking is done for storing the type information in the symbol table
- ☒ Bound checking is done for preventing users from accessing array elements data from locations which do not contain the array data.
- ☐ None of these



Consider a two dimensional array in Pascal -like language, declared as var B: array [23..78][5..33] of integer; If the compile time layout implemented by the language is COLUMN-MAJOR, then the offset of element B[45][27] from the base address is computed as

2680

Correct answers

1254

2508

5016

10032

Feedback

Answer key revised: Multiples of 1254 considered for sizeof (int) taken as 2, 4, 8. Also, 1254 is considered.



Consider the following C program and its output. What is the number of bytes required for each string in character array B?

```
#include <stdio.h>
int main()
{
    int A[]={1,2,3,4,5,6,45,33};
    char *B[]={ "abc", "def", "pqr"};
    printf("%d %d\n", sizeof(A), sizeof(B));
}
output
32 24
```

8

Correct answers

4 Bytes

4

4 bytes

4 B

4B

Feedback

B is an array of addresses, where each address is the starting address of each string. 4 bytes are for each address and 4 bytes for each string.



Consider the following C program and write the line number (1, 2 or 3) where the user has suffered the precision loss. 1/1

```
#include <stdio.h>
int main()
{
    int A[]={1,2,3,4,5,6};
    float B[]={1.2, 3.4, 5.4, 7.8};
    int C[10];
    float D[10];
    C[3]= (float) A[4]+ B[3]; //Line 1
    D[2]= (float) A[3]+ B[2]; //Line 2
    D[3]=(float) C[3]+D[2]; //Line 3
}
```

1

Consider an array in Pascal language declared as var a: array [12..30] of integers; The data values accessed through the following three elements are not spurious in the context of array element access. 1/1

- ☐ A[5], A[8], A[10]
- ☐ A[0], A[12], A[18]
- ☒ A[18], A[12], A[30]
- ☐ None of these



The heap-dynamic arrays are different from stack dynamic arrays because 0/1

- ☐ binding of the subscript and storage allocation for heap-dynamic is at compile time while that of stack dynamic is at run time
- ☒ binding of the subscript and storage allocation for heap-dynamic is at run time while that of stack dynamic is at compile time
- ☐ none of these
- ☐ Once the subscript ranges are bound and the storage is allocated, they can be changed in heap dynamic arrays while that cannot be changed in stack-dynamic arrays
- ☐ Once the subscript ranges are bound and the storage is allocated, they cannot be changed in heap dynamic arrays while that can change in stack-dynamic arrays

Correct answer

- ☒ Once the subscript ranges are bound and the storage is allocated, they can be changed in heap dynamic arrays while that cannot be changed in stack-dynamic arrays



Consider the following C program and its output. What is the output of the statement `printf("%d", sizeof(B));` before the completion of program execution? 0/1

```
#include <stdio.h>
#include<stdlib.h>
static int A[10];
int main()
{
    int n, B[10];
    int C[n];
    int *D;
    D=(int*) malloc(sizeof(int)*n);
    printf("%u %u", B, &B[7]);
    return 0;
}

Output:
2861304704 2861304760|
```

8

Correct answers

80

80 bytes

80 B

80 Bytes




Static keyword used in the declaration of a array variable as `Static int A[20];` 1/1
Which logical segment of the memory is allocated to A?

- ☒ global data section
- ☐ call stack
- ☐ code area
- ☐ heap
- ☐ None of these

Consider the following call stack storing array A of integers. The variable names are bound to locations. What is the value stored in the shaded cell in the following figure? 1/1

A[6]	23	145
A[5]	54	147
A[4]	56	149
A[3]	14	151
A[2]	11	153
A[1]	-10	155
A[0]	19	157
A		



157



Type expression for multidimensional array contains essentially the

0/1

- ☐ None of these
- ☐ array keyword, array sizes in all dimensions, base address, element type
- ☐ array subranges in all dimensions, element type, information indicating 'array' behavior, call stack size
- ☒ array keyword, array subranges in all dimensions, base address, element type
- ☐ array subranges in all dimensions, element type, information indicating 'array' behavior
- ☐ array subranges in all dimensions, information indicating 'array' behavior, call stack size, base address

Correct answer

- ☒ array subranges in all dimensions, element type, information indicating 'array' behavior



The negative subscripts for accessing array elements are supported in

1/1

- ☐ Ruby and Lua
- ☐ Lua
- ☐ Python and Ruby
- ☐ Pascal
- ☒ Ruby, Lua and Python
- ☐ Ruby
- ☐ Lua and Pascal
- ☐ Java, Ruby and Pascal
- ☐ Python
- ☐ Pascal and Ruby
- ☐ C, Java and Ruby

The bound checking for array elements $A[5]$, $A[m]$, $A[5*m]$ used in an expression $A[5] + A[m]*A[5*m]$ is done at

1/1

- ☐ None of these
- ☒ compile time, run time and run time respectively for $A[5]$, $A[m]$ and $A[5*m]$
- ☐ compile time, compile time and run time respectively for $A[5]$, $A[m]$ and $A[5*m]$
- ☐ All at run time
- ☐ compile time, run time and compile time respectively for $A[5]$, $A[m]$ and $A[5*m]$
- ☐ All at compile time



What is meant by static allocation of storage to an array variable?

1/1

- ☐ None of these
- ☐ The array variable uses static keyword and memory is allocated at compile time
- ☒ The array variable's constructed elements' relative addresses are computed at compile time
- ☐ The variable values are populated in the memory locations
- ☐ The array variable is allocated temporary memory locations for all its elements
- ☐ The storage locations of array elements are computed at run time

Consider a two dimensional array in Pascal -like language, declared as var B: 1/1
array [23..78][5..33] of integer; If the compile time layout implemented by the
language is ROW-MAJOR, then the offset of element B[45][27] from the base
address is computed as

2640

Feedback

Answer key revised: Multiples of 660 for sizeof(int) taken as 2, 4 and 8 are considered correct. Also, 660 is considered correct.



Consider the following C-like code in a language that prevents users from accessing wrong data values stored in an array. Which minimum value of U should be initialized beyond which the user gets a semantic error. [Assume that the array elements are populated in Array1 before entering the loop]

1/1

```
int main()
{
    int i, x, U;
    int Array1[200];
    x=12;
    for(i<0; i<U; i++)
        Array1[(i-1)*3*x]= x*i+1;
    return 0;
}
```

6

Feedback

Answer key revised: In view of lack of proper initialization of i, the question stands incorrect, hence all answers are given credit.

This form was created inside BITS Pilani University.

Google Forms

