Quiz: C language

Студент _____ *группы* _____

Я подтверждаю что ответы ниже являются моими собственными и даны честно _____

(число, подпись)

Для ответов на вопросы, помеченные как (PA), возьмите отдельный лист.

Q1: Насколько предсказуемо чтение из полей паддинга структуры?

В архитектуре выровненной на 4 байта с sizeof(char) == 1 байт следующая структура:

```
struct Padded { int x; char y; };
```

будет иметь в конце паддинг 3 байта. Рассмотрим следующий код:

```
struct Padded p;
memset (&p, 0xDA, sizeof(p));
p.x = 2;
p.y = 'a';
/* some code not aliasing p */
char res = *((char *) p + 6);
```

Какие из утверждений ниже верны (считаем, что случайный мусор никогда не равен 0xDA)?

1) Код не будет скомпилирован без ошибок
2) В результате всегда будет считан res == 0xDA
3) В результате в res всегда будет считан случайный мусор
4) В результате в res может быть считан как 0xDA, так и случайный мусор
5) В результате чтения произойдет runtime error

Q2: Насколько предсказуемо чтение из неинициализированной переменной?

Результатом чтения неинициализированной переменной является:

1) Run-time error
2) Определенное и предсказуемое мусорное значение
3) Неопределенное, но всегда одно и то же значение для одной и той же переменной
4) Неопределенное и, возможно, всегда разное значение для одной и той же переменной
5) Результатом является тот факт, что Дед Мороз не принесет вам подарок на Новый Год

Q3: Можно ли использовать адресную арифметику для объектов, аллоцированных в разных местах?

Рассмотрите следующий код, смешивающий адресную арифметику для стека и кучи:

```
int x = 0xBEEF;
int *px = &x;
int *mx = malloc (sizeof(int));
ptrdiff_t pd = px - mx;
int res = *(mx + pd);
```

Охарактеризуйте исполнение и результат этого кода:

1) Ошибка компиляции
2) Run-time error
3) res == 0xBEEF
4) Неопределенное значение внутри res (может быть и 0xBEEF, если повезет)
5) В результате исполнения этого кода, его разработчик погибнет от удара тяжелым предметом

Q4: Достаточно ли побитового равенства для равенства?

Для двух объектов, аллоцированных в разных местах, но имеющих одинаковое численное значение указателей на них, при сравнении указателей на равенство:

1) Указатели будут считаться равными
2) Результат не определен: указатели могут считаться не равными
3) У двух указателей на объекты в разных местах не может быть одинакового значения
4) Такое сравнение просто не скомпилируется

Q5: Можно ли сериализовать указатели?

Студент Шустрый взял значение указателя некоей переменной и записал его в файл, а через некоторое время считал из файла и разыменовал, чтобы считать значение. Пусть эта переменная все это время не выходила из области видимости и не меняла своего значения, других указателей на нее не было. В этом случае, с использованием восстановленного указателя:

1) Её значение всегда будет считано верно
2) Её значение всегда будет считано неверно
3) Её значение может быть считано как верно, так и неверно
4) При попытке такого считывания обязательно произойдет run-time error
5) Студент Шустрый не получит зачета

Q6: Сохранится ли значение указателя на удаленную область памяти?

```
int *p = malloc (sizeof(int));
long pold = (long) p;
free (p);
assert (p == (int *)pold);
```

Результатом будет:

1) Возможно assertion failure, возможно нет
2) Всегда assertion failure
3) Все нормально
4) Такая программа не скомпилируется

Q7: Возможна ли арифметика указателей за пределами массива?

Рассмотрим попытку манипулировать значением указателя за пределами адресуемого им массива, строчки для удобства пронумерованы:

```
/* 1 */ int p[] = {1, 2, 3, 0xBEEF};
/* 2 */ int *px = p;
/* 3 */ px = p + 4;
/* 4 */ px += 3;
/* 5 */ px -= 7;
/* 6 */ int res = px[3];
```

Охарактеризуйте результат исполнения такого кода:

1) Все нормально, `res == 0xBEEF`
2) Неопределенное поведение на строчке 3
3) Неопределенное поведение на строчке 4, до этого все нормально
4) Неопределенное поведение на строчке 5, до этого все нормально

Q8: Насколько стабильно нулевые указатели получаются из неконстантных выражений?

На архитектуре, где null pointer представлен ненулевым 32-битным значением, выполняется следующий код:

```
void
foo (int x, int y)
{
  int *zerop = (int*)(x – y);
  assert (zerop == NULL);
  assert (x == y);
  /* more code here */
}
```

Охарактеризуйте возможное выполнение этого кода:

1) Может сработать только первый assertion
2) Может сработать только второй assertion
3) Может сработать как первый, так и второй assertion
4) Программа не скомпилируется

Q9 (PA): как нарушить типизацию, не нарушив стандарт?

Студент Шустрый хочет исследовать структуру чисел с плавающей точкой на разных архитектурах. Для этого ему нужна функция, которая принимала бы число с плавающей точкой (double) и возвращала выделенный массив из unsigned char побитового представления этого числа и количество реально использованных бит этого массива (для некоторых архитектур это может быть 64 или 80). Как бы вы написали для него эту функцию?

Q10: Насколько стабильно нулевые указатели получаются из константных выражений?

На архитектуре, где null pointer представлен ненулевым 32-битным значением, выполняется следующий код:

```
int
foo (int *x)
{
  if (x)
    {
      assert (x != NULL);
      /* some code */
    }
  return 0;
}
```

Охарактеризуйте поведение этого кода:

1) В нём возможно assertion failure
2) В нём невозможно assertion failure
3) Такая архитекутра невозможна, NULL это всегда 0
4) Для такой архитектуры этот код не скомпилируется