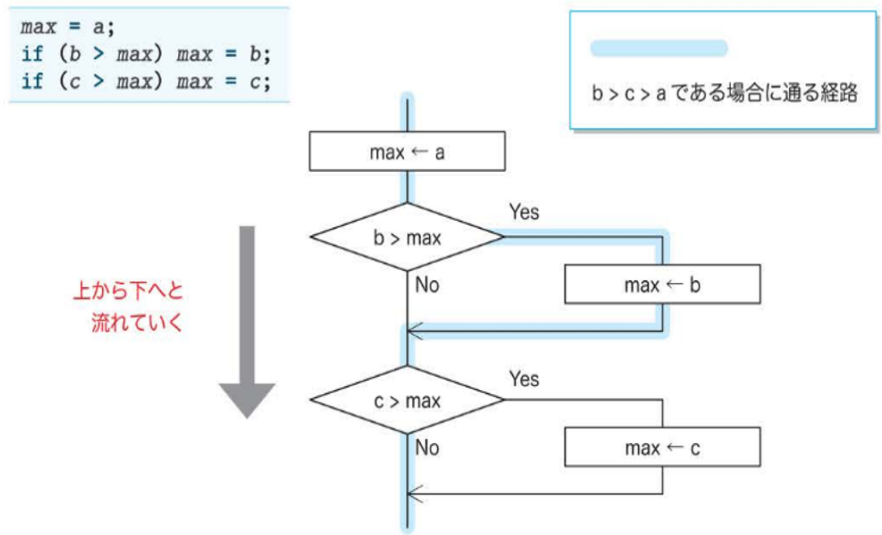



【 番号 】に合う答えを解答群から選び、記号で答えよ。

問題.1 フローチャート図の問題です



プログラムの流れは、黒線に沿って上から下へと向かい、その過程で に置かれた【 ① 】が実行されます。

ただし、  を通過する際は、その中に置かれた【 ② 】の評価結果に応じて、YesとNoのいずれか一方をたどります。そのため、条件 $b > \text{max}$ や条件 $c > \text{max}$ が成立すれば(すなわち、式 $b > \text{max}$ や式 $c > \text{max}$ を評価した値が1であれば)、【 ③ 】と書かれた右側に進み、そうでなければ0であれば【 ④ 】と書かれた下側に進みます。
if 文やwhile 文などの条件判定のために置かれる)中の式は、制御式と呼ばれます。

プログラムの流れが、二つに分岐されたルート of のいずれか一方を通ることからif 文によるプログラムの流れの分岐は、双岐選択と呼ばれます。

【解答群】

	【①】
ア	処理
イ	条件

	【②】
ア	処理
イ	条件

	【③】
ア	Yes
イ	No

	【④】
ア	Yes
イ	No

問題.2 三つの整数の中央値を求める問題です

```
#include <stdio.h>

/*--- a, b, cの中央値を求める---*/
int med3(int a, int b, int c)
{
    if (a >= b)
        if (b >= c)
            return 【 ① 】;
        else if (a <= c)
            return 【 ② 】;
        else
            return 【 ③ 】;
    else if (a > c)
        return a;
    else if (b > c)
        return c;
    else
        return b;
}

int main(void)
{
    int a, b, c;

    printf("三つの整数の中央値を求めます。\\n");
    printf("aの値:"); scanf("%d", &a);
    printf("bの値:"); scanf("%d", &b);
    printf("cの値:"); scanf("%d", &c);

    printf("中央値は%dです。\\n", med3(a, b, c));

    return 0;
}
```

実行例

```
三つの整数の中央値を求めま
aの値:1
bの値:3
cの値:2
中央値は2です。
```

【解答群】

	【①】	【②】	【③】
ア	a	c	b
イ	b	a	c

問題.3 読み込んだ整数値の符号(正／負／0)を判定

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int n;
```

```
    printf("整数:");
```

```
    scanf("%d", &n);
```

```
    if (n > 0)
```

```
        printf(【 ① 】);
```

```
    else if (n < 0)
```

```
        printf(【 ② 】);
```

```
    else
```

```
        printf(【 ③ 】);
```

```
    return 0;
```

```
}
```

実行例

整数:15

正です。

整数:-5

負です。

整数:0

0です。

【解答群】

	【①】	【②】	【③】
ア	"負です。\\n"	"正です。\\n"	"0です。\\n"
イ	"正です。\\n"	"負です。\\n"	"0です。\\n"

問題.4 1からnまでの整数の総和を求める

nが2であれば $\rightarrow 1 + 2$

nが3であれば $\rightarrow 1 + 2 + 3$

// 1, 2, ..., n の総和を求める (while文)

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int n;
```

```
    puts("1からnまでの総和を求めます。");
```

```
    printf("nの値:");
```

```
    scanf("%d", &n);
```

```
    int sum = 0; // 総和      ← ①
```

```
    int i = 1;
```

```
    while (i <= n) {          // iがn以下であれば繰り返す
```

```
        【 ① 】              // sumにiを加える
```

```
        【 ② 】              // iの値をインクリメント ← ②
```

```
    }
```

```
    printf("1から%dまでの総和は%dです.\n", n, sum);
```

```
    return 0;
```

```
}
```

実行例

1からnまでの総和を求めます。

nの値:5

1から5までの総和は15です。

【解答群】

	【①】	【②】
ア	sum = i;	i = sum;
イ	sum += i;	i++;

問題.5 関係演算子と等価演算子

関係演算子と等価演算子は、大小関係や等値関係の判定が成立すれば真であれば

【 ① 】を、成立しなければ偽であれば【 ② 】を生成する。

値【 ② 】が偽とみなされて、【 ② 】でないすべての値が真とみなされることは、必ず覚えておく必要があります。

【 ① 】でも【 ③ 】でも、とにかく【 ② 】でなければ真です。そのため、

```
if (a) printf("ABC");
```

を実行すると、変数aの値が【 ② 】でなければ「ABC」と表示されます。

【解答群】

	【①】	【②】	【③】
ア	int型の1	int 型の0	int 型の100
イ	int型の1	int 型の0	int 型の-100
ウ	int 型の0	int型の1	int 型の100
エ	int 型の0	int型の1	int 型の-100

問題.6 記憶域の確保

calloc

形 式 void *calloc(size_t nmemb, size_t size);

解説 大きさが【 ① 】であるオブジェクト【 ② 】分の配列領域を確保する。
その領域は、すべてのビットが0で初期化される。

返却値 領域確保に成功した場合は、確保した領域の【 ③ 】を返し、
失敗した場合は、【 ④ 】を返す。

【解答群】

	【①】		【②】
ア	nmemb	ア	size
イ	size	イ	nmemb個

	【③】		【④】
ア	先頭へのポインタ	ア	先頭へのポインタ
イ	空ポインタ	イ	空ポインタ

free

形式 void free(void *ptr);

解説 ptr が指す領域を【 ⑤ 】して、その後の割付けに使用できるようにする。ptrが
【 ⑥ 】の場合は、何も行わない。それ以外の場合、実引数がcalloc関数、
malloc 関数もしくはrealloc 関数によって以前に返されたポインタと一致しないとき、
またはその領域がfree関数もしくはrealloc関数の呼出しによって既に解放されて
いるときの動作は定義されない。

返却値 なし。

【解答群】

	【⑤】		【⑥】
ア	取得	ア	ヒープ
イ	解放	イ	空ポインタ

いずれの関数も、一般に【 ⑦ 】と呼ばれる、特別な“空き領域”から記憶域を確保します。確保したオブジェクトの寿命生存期間)は、割付け記憶域期間(allocated storage duration)と呼ばれます。

なお、確保した記憶域が不要になったら、その領域を解放します。そのために提供されているのが【 ⑧ 】関数です。

【解答群】

	【⑦】
ア	ヒープ
イ	スタック

	【⑧】
ア	malloc
イ	free

```
int main(void)
{
    int *x = calloc(【 ⑨ 】); // int型オブジェクトを生成
    if (x == 【 ⑩ 】)
        puts("記憶域の確保に【 ⑪ 】しました。");
    else {
        *x = 57;
        printf("*x = %d\n", *x);
        free(【 ⑫ 】); // int型オブジェクトを破棄
    }
    return 0;
}
```

【解答群】

	【⑨】
ア	int
イ	1 , sizeof(int)
ウ	sizeof(int)
エ	int

	【⑩】
ア	1
イ	0
ウ	FALSE
エ	NULL

	【⑪】
ア	成功
イ	失敗

	【⑫】
ア	*x
イ	x
ウ	1 , sizeof(int)
エ	sizeof(int)

問題.7 ポインタ

「ポインタ p がオブジェクト n を指す」ようにするには、 n のアドレスを p に代入する必要があります。それを実現するのが、以下の代入です。

```
int n = 0;
```

```
int *p;
```

【 ① 】; // n へのポインタを p に代入する (p が n を指すようにする)

ポインタが指すオブジェクトには、「間接演算子」と呼ばれる単項演算子を使ってアクセスできます。

先ほどの代入によって p が n を指しているので、 p から n へアクセスする間接式 $*p$ です。そのため、

【 ② 】= 999; // p の指す先に 999 を代入する

を実行すると、 n に 999 が代入されます。

【解答群】

	【①】		【②】
ア	$*p = n$	ア	p
イ	$p = *n$	イ	$*p$
ウ	$p = n$	ウ	$\&p$
エ	$p = \&n$	エ	$**p$

● ポインタと配列

```
int a[5] = {10, 20, 30, 40, 50};
```

```
int *p = a;
```

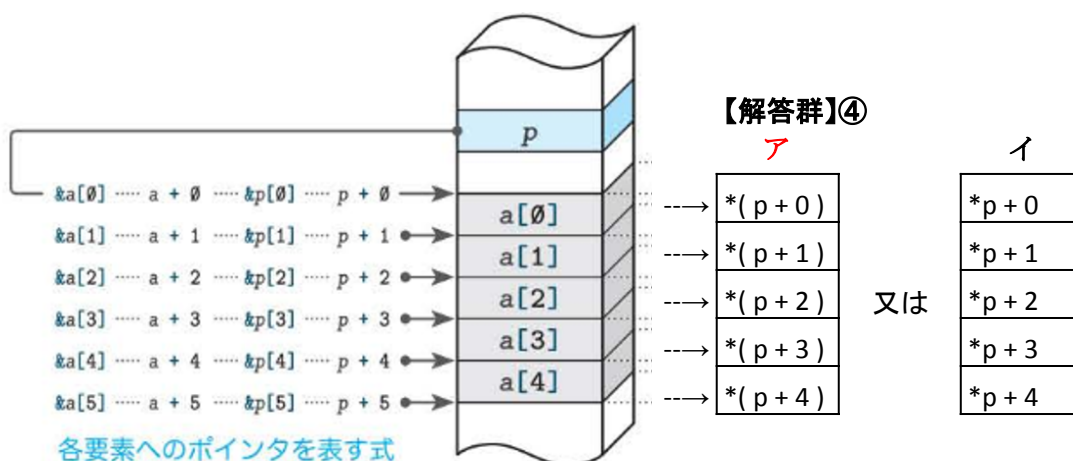
配列 a とポインタ p が宣言されており、 p の初期化子は、配列名 a です。

原則として、配列名は、その配列の先頭要素へのポインタと解釈されます。

すなわち、式 a の値は $a[0]$ のアドレスである【 ③ 】と一致します。

【解答群】

	【③】
ア	$\&a[0]$
イ	$a[0]$
ウ	$*a[0]$
エ	$\text{sizeof}(\text{int})$



ポインタ p が配列中の要素 e を指すとき、

$p + i$ は、要素 e の i 個だけ【 ⑤ 】の要素を指すポインタとなり、

$p - i$ は、要素 e の i 個だけ【 ⑥ 】の要素を指すポインタとなる。

【解答群】

	【⑤】	【⑥】
ア	後方	前方
イ	前方	後方

問題.8 乱数

100～189 の範囲の乱数を生成し、出力を行います
」

```
int main(void)
{
    // 乱数の種を初期化
    【 ① 】(time(NULL));

    // 100～189 の範囲の乱数を生成
    int value = 【 ② 】;

    printf("%d\n", value);

    return 0;
}
```

【解答群】

	【①】
ア	free
イ	rand
ウ	printf
エ	srand

	【②】
ア	rand() % 189
イ	rand() % 100
ウ	rand()
エ	100 + rand() % 90

rand 関数が生成する乱数は、【 ③ 】と呼ばれます。【 ③ 】は、乱数のように見えますが、実際には一定の法則に従って生成される数列です。

【解答群】

	【③】
ア	真の乱数
イ	連続乱数
ウ	偽乱数
エ	疑似乱数

問題.9 構造体

こういう利用を想定した構造体について

	<i>name</i>		<i>height</i>		<i>vision</i>
0	赤坂忠雄	0	162	0	0.3
1	加藤富明	1	173	1	0.7
2	斉藤正二	2	175	2	2.0
3	武田信也	3	171	3	1.5
4	長浜正樹	4	168	4	0.4
5	浜田哲明	5	174	5	1.2
6	松富明雄	6	169	6	0.8

/*--- 身体検査データ型 ---*/

typedef 【 ① 】{

 【 ② 】 // 氏名 / 身長 / 視力

} PhysCheck;

【解答群】

	【①】		【②】
ア	int	ア	char name[20];
イ	void		int height;
ウ	static		double vision;
エ	struct	イ	PhysCheck date;

問題.10 2分探索

2分探索は要素の値が【 ① 】に【 ② 】されている配列から効率よく探索を行うアルゴリズムです

【解答群】

	【①】		【②】
ア	昇順または降順	ア	ソート
イ	無作為	イ	コピー
ウ	static	ウ	分割
エ	struct	エ	整数化

```
int bin_search(const int a[], int n, int key) {  
    int pl = 【 ③ 】;  
    int pr = 【 ④ 】;  
    do {  
        int pc =【 ⑤ 】;  
        if (【 ⑥ 】)  
            return pc;  
        else if (a[pc] < key)  
            pl = pc + 1;  
        else  
            pr = pc - 1;  
    } while (pl <= pr);  
    return -1;  
}
```

```
int main(void) {  
    int x[7] = {12, 27, 39, 77, 92, 118, 121};  
    int key = 92;  
    int idx = bin_search(x, 7, key);  
    if (idx != -1)  
        printf("%d\n", idx); // 添え字だけを入力  
    else  
        printf("検索対象はありませんでした\n");  
  
    return 0;  
}
```

【解答群】

	【③】
ア	a[0]
イ	key
ウ	1
エ	0

	【④】
ア	a[0]
イ	0
ウ	1
エ	n - 1

	【⑤】
ア	(n + pr) / 2
イ	pl
ウ	pr
エ	(pl + pr) / 2

	【⑥】
ア	a[i] == key
イ	a[pc] > key
ウ	a[pc] < key
エ	a[pc] == key

問題.11 関数へのポインタ

関数へのポインタとは、名前の通り、関数を指すポインタです。

関数へのポインタの型は、指す対象となる関数の型によって異なります
次の関数はint型の引数を受け取ってdouble型の値を返却する関数です。

```
double func(int);
```

この関数を指すポインタの型、fpの宣言は次のようになります。

【 ① 】;

【解答群】

	【①】
ア	double (*fp)(int)
イ	double *fp(int)

```
// 九九の加算と乗算
```

```
#include <stdio.h>
```

```

/*--- x1とx2の和を求める---*/
int sum( int x1 , int x2 )
{
    return x1 + x2;
}

```

```

/*--- x1とx2の積を求める---*/
int mul( int x1 , int x2 )
{
    return x1 * x2;
}

```

```

/*--- 九九の表を出力---*/
void kuku(int (*calc)(int, int) )
{
    for (int i = 1; i <= 9; i++) {
        for (int j = 1; j <= 9; j++)
            printf("%3d", 【 ② 】);
        putchar('\n');
    }
}

```

```

int main(void)
{
    puts("九九の加算表");
    【 ③ 】;
    puts("\n九九の乗算表");
    【 ④ 】;

    return 0;
}

```

実行例

九九の加算表

```

2  3  4  5  6  7  8  9 10
3  4  5  6  7  8  9 10 11
4  5  6  7  8  9 10 11 12
5  6  7  8  9 10 11 12 13
6  7  8  9 10 11 12 13 14
7  8  9 10 11 12 13 14 15
8  9 10 11 12 13 14 15 16
9 10 11 12 13 14 15 16 17
10 11 12 13 14 15 16 17 18

```

九九の乗算表

```

1  2  3  4  5  6  7  8  9
2  4  6  8 10 12 14 16 18
3  6  9 12 15 18 21 24 27
4  8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81

```

【解答群】

	【②】	【③】	【④】
ア	(*calc)(i , j)	kuku(sum);	kuku(mul);
イ	(*calc)(int , int)	kuku(sum(int , int));	kuku(mul(int , int));
ウ	*calc(i , j)	kuku(sum);	kuku(mul);
エ	*calc(int , int)	kuku(sum(int , int));	kuku(mul(int , int));

問題.12 クイックソート

```
#include <stdio.h>
#include <stdlib.h>
#define swap(type, x, y) do { type t = x; x = y; y = t; } while (0)

/*--- クイックソート ---*/
void quick(int a[], int left, int right) {
    int pl = left;
    int pr = right;
    int x = a[【 ① 】];
    do {
        while (a[pl] 【②】x) pl++;
        while (a[pr] 【③】x) pr--;
        if (pl 【④】 pr) {
            swap(int, a[pl], a[pr]);
            pl++;
            pr--;
        }
    } while (pl <= pr);
    【 ⑤ 】
}

int main(void) {
    int x[9] = {5, 8, 4, 2, 6, 1, 3, 9, 7}; // 初期化された配列
    int nx = 9;

    quick(x, 0, nx - 1); // ソート実行

    for (int i = 0; i < nx; i++)
        printf("x[%d] = %d\n", i, x[i]); // ソート後の出力

    return 0;
}
```

【解答群】

	【①】
ア	$(pl - pr)$
イ	$(pl + pr)$
ウ	$(pl - pr) / 2$
エ	$(pl + pr) / 2$

	【②】
ア	$==$
イ	$+$
ウ	$>$
エ	$<$

	【③】
ア	$==$
イ	$+$
ウ	$>$
エ	$<$

	【④】
ア	$==$
イ	$+$
ウ	$>=$
エ	$<=$

	【⑤】
ア	if (left < pr) quick(a, left, pr); if (pl < right) quick(a, pl, right);
イ	if (left < pr) return 0; if (pl < right) return 0;