

Control Flow, Function, and Array

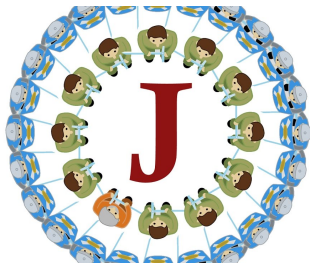
The Josephus Puzzle

魏恒峰

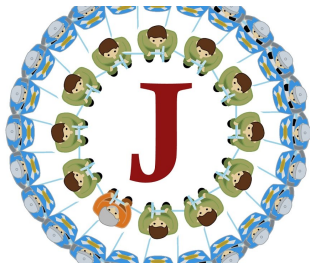
hfwei@nju.edu.cn

2017 年 10 月 27 日

The Josephus Puzzle

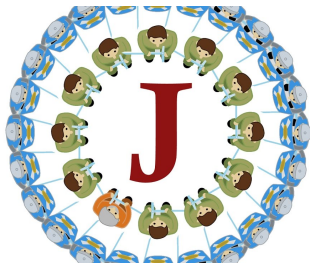


The Josephus Puzzle



$$J(n) = ?$$

The Josephus Programming Task



Input: n
Output: $J(n)$

Input: n
Output: $J(1), J(2), \dots, J(n)$

Solving the Josephus Puzzle

Josephus 编程任务分解

```
int main(void) {  
    return 0;  
}
```

1. *input the number n of soliders*
2. *find the survivor of the Josephus puzzle with n soliders*
3. *output the survivor*

Josephus 编程任务分解

```
int main(void) {  
    return 0;  
}
```

1. *input the number n of soliders*
2. *find the survivor of the Josephus puzzle with n soliders*
3. *output the survivor*

```
int main(void) {  
    int n = 0;  
    scanf("%d", &n);  
    int survivor = solve_josephus(int n);  
    printf("The survivor is %d.", survivor + 1);  
    return 0;  
}
```

```
int solve_josephus(int n);
```

1. create n soldiers with ids $1 \cdots n$
2. keep killing each other until only one soldier survives
3. return the id of the survivor


```
int solve_josephus(int n);
```

1. create n soldiers with ids $1 \dots n$
2. keep killing each other until only one soldier survives
3. return the id of the survivor

```
int solve_josephus(int n) {  
    // create  $n$  soldiers with ids  $1 \dots n$   
    int soldiers[n];  
    for (int i = 0; i < n; ++i) {  
        soldiers[i] = i + 1;  
    }  
  
    return survive(soldiers, n);  
}
```

```
int survive(int soldiers[], int n);
```

1. kill $n - 1$ soldiers
 - 1.1 identify the killer
 - 1.2 identify the killed

```
int survive(int soldiers[], int n);
```

1. kill $n - 1$ soldiers
 - 1.1 identify the killer
 - 1.2 identify the killed

```
int survive(int soldiers[], int n) {  
    int killer = 0, killed = 0;  
    // kill  $n - 1$  soldiers  
    for (int i = 0; i < n - 1; ++i) {  
        killed = next_alive(soldiers, n, killer);  
        soldiers[killed] = DEAD;    // #define DEAD 0  
        killer = next_alive(soldiers, n, killed);  
    }  
    return killer;  
}
```

```
int next_alive(int soldiers[], int n, int pos)
```

```
int next_alive(int soldiers[], int n, int pos) {  
    do {  
        pos = (pos + 1) % num;  
    } while (soldier[pos] == KILLED);  
  
    return pos;  
}
```

```
void test_josephus(int n);
```

```
void test_josephus(int n) {  
    for (int i = 1; i <= limit; ++i) {  
        printf("%d: %d\n", i, solve_josephus(i) +  
            1);  
    }  
}
```

```
void test_josephus(int n);
```

```
void test_josephus(int n) {  
    for (int i = 1; i <= limit; ++i) {  
        printf("%d: %d\n", i, solve_josephus(i) +  
            1);  
    }  
}
```

$n = 50, 100, \dots, 1000$

$n = 16, 64, 128, 1024$

```
void test_josephus(int n);
```

```
void test_josephus(int n) {  
    for (int i = 1; i <= limit; ++i) {  
        printf("%d: %d\n", i, solve_josephus(i) +  
            1);  
    }  
}
```

$n = 50, 100, \dots, 1000$

$n = 16, 64, 128, 1024$

Q: What have you found?

Functions

Prototype and Definition

```
int solve_josephus(int n);
```

```
int main(void) {  
    ...  
    // call the function  
    int survivor = solve_josephus(int n);  
    ...  
}
```

```
int solve_josephus(int n) {  
}
```

Variables and Scopes

Variables:

1. external variables
2. automatic variables
3. parameters

Pass by Value

```
void swap(int a, int b) {  
    int temp = a;  
    a = b;  
    b = tmp;  
}
```

Pass by Value

```
void swap(int a, int b) {  
    int temp = a;  
    a = b;  
    b = tmp;  
}
```

```
void swap(int *a, int *b) {  
    int temp = *a;  
    *a = *b;  
    *b = tmp;  
}
```

Control Flow

if

switch

for

while

do while

goto

Arrays

Array Declaration and Initialization

```
int soldiers[5];
```

```
int soldiers[] = {1, 2, 3, 4, 5};
```

```
int soldiers[5] = {1, 2, 3};
```

For Loop over Array

```
for (int i = 0; i < n; i++) {  
    arr[i] ...  
}
```

Array as Parameters