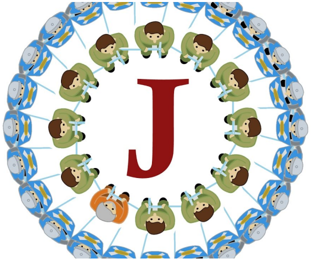
The Josephus Puzzle
— Control Flow, Function, and Array

魏恒峰

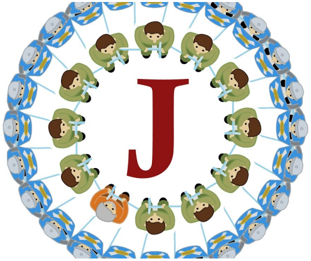
hfwei@nju.edu.cn

2017 年 10 月 27 日

The Josephus Puzzle

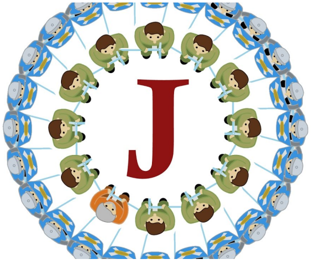


The Josephus Puzzle



$$J(n) = ?$$

The Josephus Programming Task



Input: n
Output: $J(n)$

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

Solving the Josephus Puzzle

Josephus Programming Task Decomposed

```
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 Programming Task Decomposed

```
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'', &d);
  int survivor = solve_josephus(int n);
  printf(''The survivor is %d.'', survior + 1);
  return 0;
}
```

int solve_josephus(int n);

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

```c
int solve_josephus(int n) {
  // create n soldiers with ids 1···n
  int *soldiers = malloc(sizeof(int) * 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)

```c
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);
  }
}
```

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

$$n = 16, 64, 128, 1024$$

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);
      }
    }
```

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

$$n = 16, 64, 128, 1024$$

Q: What have you found?

Generalized Josephus Puzzle

```
   int solve_generalized_josephus(int soldiers[],
      int from_position, int interval);
```

Functions

Prototype and Definition

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

Making function names verbs.

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

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

Variables and Scopes

Variables:

1. Automatic variables
2. Parameters
3. External variables

Variables and Scopes

Variables:

1. Automatic variables
2. Parameters
3. External variables

We will learn more about it in the next class.

Variables and Scopes

Variables:

1. Automatic variables
2. Parameters
3. External variables

We will learn more about it in the next class.

$\boxed{\text{You shall always initialize variables. Always. Every time.}}$

Pass by Value

```
    swap(a, b);

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

Pass by Value

```
swap(a, b);

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

```
swap(&a, &b);

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

Pass by Value

```
swap(a, b);

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

```
swap(&a, &b);

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

Control Flow

if; switch, case

```
if (condition A) {
  ...
} else if (condition
   B) {
  ...
} else { // ¬A ∨ ¬B
  ...
}
```

```
switch () {
  case c1:
    ...
    break;
  case c2:
    ...
    break;
  default:
}
```

for

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

while

```
while (condition) {
  ...
}
```

```
do {
  ...
} while (condition);
```

Arrays

Array Declaration and Initialization

```c
int soldiers[5];  // external/automatic

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

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

int soldiers[5] = {1};

int *soldiers = malloc(sizeof(int) * n);
```

For Loop over Array

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

Array as Parameters

```
f(arr, size);

void f(int arr[], int size);

void f(int *arr, int size);
```

Thank
You!