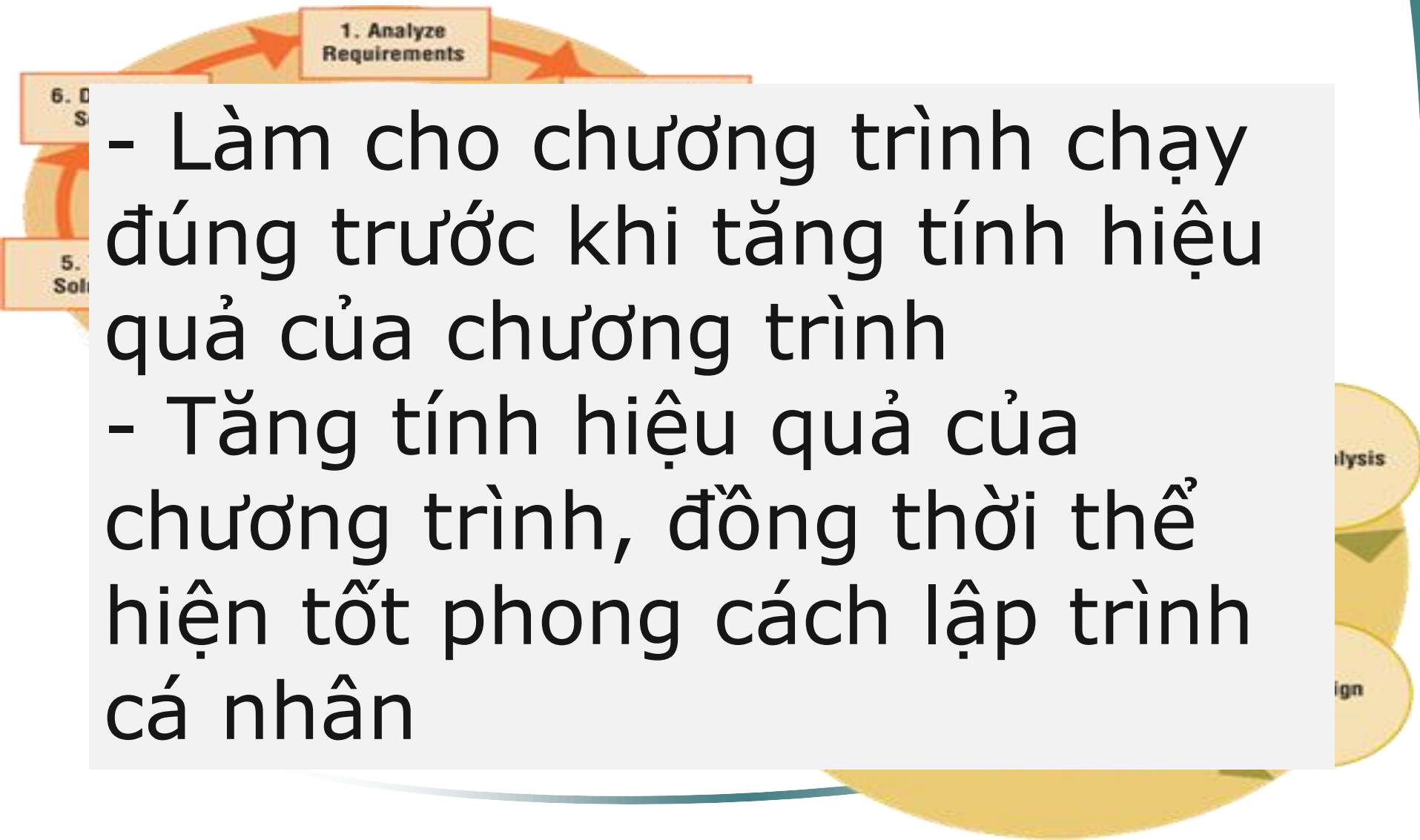



- Với mỗi bài toán, làm thế nào để:
 - Thiết kế giải thuật nhằm giải quyết bài toán đó
 - Cài đặt giải thuật bằng một chương trình máy tính



- Làm cho chương trình chạy đúng trước khi tăng tính hiệu quả của chương trình
- Tăng tính hiệu quả của chương trình, đồng thời thể hiện tốt phong cách lập trình cá nhân




CHƯƠNG III. CÁC KỸ THUẬT XÂY DỰNG CHƯƠNG TRÌNH PHẦN MỀM

- I. Mở đầu
 - II. Làm việc với biến
 - III. Viết mã chương trình hiệu quả
 - IV. Thiết kế chương trình
 - V. Xây dựng hàm/thủ tục
- 



IV. CÁC KỸ THUẬT THIẾT KẾ CHƯƠNG TRÌNH

1. Nguyên tắc chung
 2. Thiết kế giải thuật
 3. Thiết kế dữ liệu
- 



Mở đầu

- Phẩm chất của 1 chương trình tốt
 - Cấu trúc tốt
 - Logic chương trình + các biểu thức được diễn đạt theo cách thông thường
 - Tên dùng trong chương trình có tính chất miêu tả
 - Chú thích hợp lý
 - Tôn trọng chiến lược divide/conquer/association
- Làm thế nào để tạo ra chương trình có phẩm chất tốt
 - Thiết kế top-down
 - Tinh chỉnh từng bước



1. Nguyên tắc chung

- Đơn giản:
 - Thể hiện giải thuật như nó vốn có, đừng quá kỳ bí
 - Lựa chọn cấu trúc dữ liệu sao cho việc viết giải thuật bằng NNLT cụ thể là đơn giản nhất
 - Tìm cách đơn giản hóa các biểu thức
 - Thay những biểu thức lặp đi lặp lại bằng CTC tương ứng
- Trực tiếp:
 - Sử dụng thư viện mọi lúc có thể
 - Tránh việc kiểm tra điều kiện không cần thiết
- Rõ ràng:
 - Dùng các cặp dấu đánh dấu khối lệnh để tránh nhập nhằng
 - Đặt tên biến, hàm, .. sao cho tránh được nhầm lẫn
 - Không chắp vá các đoạn mã khó hiểu mà nên viết lại



1. Nguyên tắc chung

- Có cấu trúc tốt:
 - Tôn trọng tính cấu trúc của chương trình theo từng mô thức lập trình:
 - Modul: hàm/ thủ tục
 - Hướng đối tượng: lớp
 - Hướng thành phần: thành phần
 - Hướng dịch vụ: dịch vụ
 - Viết và kiểm thử dựa trên cấu trúc phân cấp của chương trình
 - Tránh hoàn toàn việc dùng goto
 - Nếu cần thì nên viết giải thuật bằng giả ngữ, rồi mới viết bằng 1 NNLT cụ thể

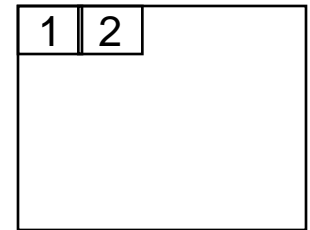


2. Thiết kế giải thuật

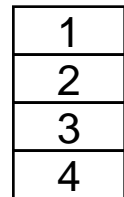
- Chia bài toán ra thành nhiều bài toán nhỏ hơn
 - Tìm giải pháp cho từng bài toán nhỏ
 - Gộp các giải pháp cho các bài toán nhỏ thành giải pháp tổng thể cho bài toán ban đầu
- Đơn giản hóa bài toán bằng cách trừu tượng hóa: làm cái gì thay vì làm như thế nào
- Ví dụ: các hàm ở mức trừu tượng
 - Hàm sắp xếp 1 mảng các số nguyên
 - Hàm nhập vào / xuất ra các ký tự: `getchar()` , `putchar()`
 - Hàm toán học : `sin(x)`, `sqrt(x)`

Bottom-Up Design is Bad

- Bottom-up design ☹️
 - Thiết kế chi tiết 1 phần
 - Thiết kế chi tiết 1 phần khác
 - Lặp lại cho đến hết
- Bottom-up design in programming
 - Viết phần đầu tiên của CT 1 cách chi tiết cho đến hết
 - Viết phần tiếp theo của CT 1 cách chi tiết cho đến hết
 - Lặp lại cho đến hết



...



...

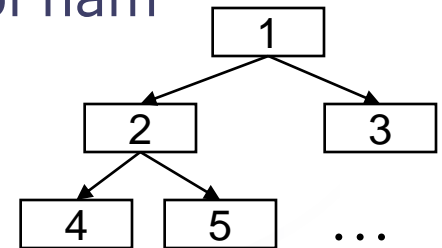
Top-Down Design is Good

- Top-down design 😊

- Thiết kế toàn bộ sản phẩm một cách sơ bộ, tổng thể
- Tinh chỉnh cho đến khi hoàn thiện

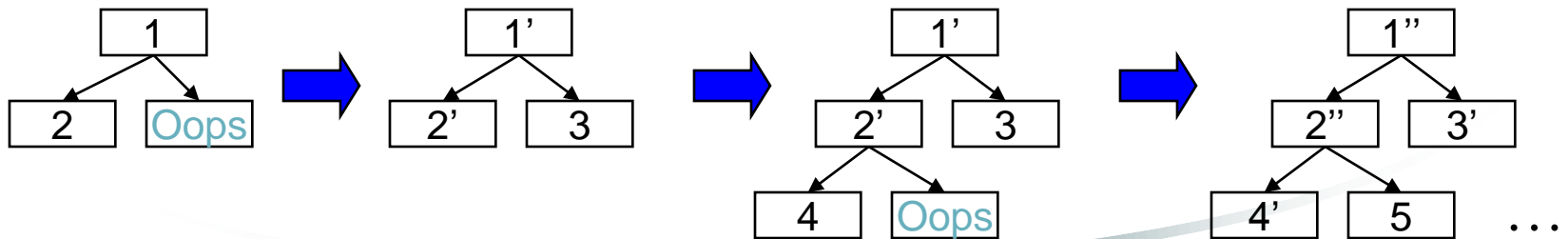
- Top-down design in **programming**

- Phác họa hàm main() (bằng các lệnh giả ngữ - pseudocode)
- Tinh chỉnh từng lệnh giả ngữ
 - Công việc đơn giản => thay bằng real code
 - Công việc phức tạp => thay bằng lời gọi hàm
- Lặp lại sâu hơn, cụ thể, chi tiết hơn
- Kết quả: Sản phẩm có cấu trúc phân cấp tự nhiên



Top-Down Design in Reality

- Thiết kế CT Top-down trong thực tiễn :
 - Định nghĩa hàm main() = pseudocode
 - Tinh chỉnh từng lệnh pseudocode
 - Nếu gặp sự cố Oops! Xem lại thiết kế, và...
 - Quay lại để tinh chỉnh pseudocode đã có, và tiếp tục
 - Lặp lại (mostly) ở mức sâu hơn, cụ thể hơn, cho đến khi các hàm đc định nghĩa xong





Ví dụ: Text Formatting

- Mục tiêu :
 - Minh họa good program và programming style
 - Đặc biệt là modul hóa mức hàm và top-down design
 - Minh họa cách đi từ vấn đề đến viết code
 - Ôn lại và mô tả cách xây dựng CTC
- Text formatting
 - Đầu vào: ASCII text, với hàng loạt dấu cách và phân dòng
 - Đầu ra: Cùng nội dung, nhưng căn trái và căn phải
 - Dồn các từ tối đa có thể trên 1 dòng 50 ký tự
 - Thêm các dấu cách cần thiết giữa các từ để căn phải
 - Không cần căn phải dòng cuối cùng
 - Để đơn giản hóa, giả định rằng :
 - 1 từ kết thúc bằng dấu cách space, tab, newline, hoặc end-of-file
 - Không có từ nào quá 20 ký tự

Ví dụ về Input and Output

I Tune every heart and every voice.
N Bid every bank withdrawal.
P Let's all with our accounts rejoice.
U In funding Old Nassau.
T In funding Old Nassau we spend more money every year.
Our banks shall give, while we shall live.
We're funding Old Nassau.

O
U
T
P
U
T
Tune every heart and every voice. Bid every bank
withdrawal. Let's all with our accounts rejoice.
In funding Old Nassau. In funding Old Nassau we
spend more money every year. Our banks shall give,
while we shall live. We're funding Old Nassau.



Nghiên cứu bài toán

- Khái niệm “từ”
 - Chuỗi các ký tự không có khoảng trắng, tab xuống dòng, hoặc EOF
 - Tất cả các ký tự trong 1 từ phải đc in trên cùng 1 dòng
- Làm sao để đọc và in đc các từ
 - Đọc các ký tự từ stdin cho đến khi gặp space, tab, newline, or EOF
 - In các ký tự ra stdout tiếp theo bởi các dấu space(s) or newline
- Nếu đầu vào lộn xộn thì thế nào?
 - Cần loại bỏ các dấu spaces thừa, các dấu tabs, và newlines từ input
- Làm sao có thể căn trái - phải ?
 - Ta không biết được số dấu spaces cần thiết cho đến khi đọc hết các từ
 - Cần phải lưu lại các từ cho đến khi có thể in được trọn vẹn 1 dòng
- Nhưng, bao nhiêu space cần phải thêm vào giữa các từ?
 - Cần ít nhất 1 dấu space giữa các từ riêng biệt trên 1 dòng
 - Có thể thêm 1 vài dấu spaces để phủ kín 1 dòng



Viết chương trình

- Các cấu trúc dữ liệu chính
 - Từ - Word
 - Dòng - Line
- Các bước tiếp theo
 - Viết pseudocode cho hàm main()
 - Tinh chỉnh
- Lưu ý :
 - Chú thích hàm và một số dòng trống được bỏ qua vì những hạn chế không gian
 - Phải tôn trọng các quy tắc trình bày mã nguồn khi viết CT thực tế
 - Trình tự thiết kế là lý tưởng
 - Trong thực tế, nhiều backtracking sẽ xảy ra

Mức đỉnh

- Phác thảo hàm main()...

```
int main(void) {  
    <Xóa dòng>  
    for (;;) {  
        <Đọc 1 từ>  
        if (<Hết từ>) {  
            <In dòng không cần căn phải>  
            return 0;  
        }  
        if (<Từ không vừa dòng hiện tại>) {  
            <In dòng có căn lề phải>  
            <Xóa dòng>  
        }  
        <Thêm từ vào dòng>  
    }  
    return 0;  
}
```

Tính chỉnh từng bước

- Đọc 1 từ

- <Đọc 1 từ> nghĩa là gì?
- Việc này khá phức tạp nên cần tách thành 1 hàm riêng ...

```
#include <stdio.h>
enum {MAX_WORD_LEN = 20};
int main(void) {
    char word[MAX_WORD_LEN + 1];
    int wordLen;
    < Xóa dòng >
    for (;;) {
        wordLen = ReadWord(word);
        if (<Hết từ>) {
            < In dòng không cần căn phải >
            return 0;
        }
        if (< Từ không vừa dòng hiện tại >)
        {
            < In dòng có căn lề phải >
            < Xóa dòng >
        }
        < Thêm từ vào dòng >
    }
    return 0;
}
```

```
int ReadWord(char *word) {
    <Bỏ qua whitespace>
    <Lưu các ký tự cho đến MAX_WORD_LEN của từ>
    <Trả về độ dài từ>
}
```




The “End-of-File Character”

- Các files không kết thúc bằng “EOF character”, vì không tồn tại ký tự đó
- EOF là:
 - Một giá trị đặc biệt được hàm `getchar()` hoặc các hàm liên quan trả về để chỉ ra 1 lỗi vào ra
 - Được định nghĩa trong `stdio.h` (thường với giá trị -1)
 - Trong môi trường windows, có thể tương đương với mã ASCII của cụm phím tắt `Ctrl + Z`

Using EOF

- Correct code

```
int c;  
c = getchar();  
while (c != EOF) {  
    ...  
    c = getchar();  
}
```

getchar() trả lại giá trị kiểu int cho tất cả các ký tự và ký hiệu **EOF**

- Equivalent idiom

```
int c;  
while ((c = getchar()) != EOF) {  
    ...  
}
```

- Incorrect code

```
char c;  
while ((c = getchar()) != EOF) {  
    ...  
}
```

Tại sao ?

Tính chỉnh từng bước

- Đọc 1 từ

- ReadWord() function

```
int ReadWord(char *word) {
    int ch, pos = 0;

    /* Bỏ qua whitespace. */
    ch = getchar();
    while ((ch == ' ') || (ch == '\n') || (ch == '\t'))
        ch = getchar();

    /* Lưu các ký tự vào từ cho đến MAX_WORD_LEN . */
    while ((ch != ' ') && (ch != '\n') && (ch != '\t') && (ch != EOF)) {
        if (pos < MAX_WORD_LEN) {
            word[pos] = (char)ch;
            pos++;
        }
        ch = getchar();
    }
    word[pos] = '\0';

    /* Trả về độ dài từ. */
    return pos;
}
```

Tính chỉnh từng bước

- Đọc 1 từ

- Hmm. ReadWord() chứa 1 vài đoạn code lặp lại

```
int ReadWord(char *word) {
    int ch, pos = 0;

    /* Bỏ qua whitespace. */
    ch = getchar();
    while (IsWhitespace(ch))
        ch = getchar();

    /* Lưu các ký tự vào từ cho đến MAX_WORD_LEN . */
    while (!IsWhitespace(ch) && (ch != EOF)) {
        if (pos < MAX_WORD_LEN) {
            word[pos] = (char)ch;
            pos++;
        }
        ch = getchar();
    }
    word[pos] = '\0';

    /* trả về độ dài từ. */
    return pos;
}
```

IsWhitespace(ch)

Có thật sự phải
tự viết hàm kiểm
tra này không ?
ctype.h cung cấp
hàm isspace()
với chức năng
tương đương

```
int IsWhitespace(int ch) {
    return (ch == ' ') || (ch == '\n') || (ch == '\t');
}
```

Tình chỉnh từng bước: Nhớ từ

```
#include <stdio.h>
#include <string.h>
enum {MAX_WORD_LEN = 20};
enum {MAX_LINE_LEN = 50};
int main(void) {
    char word[MAX_WORD_LEN + 1];
    int wordLen;
    char line[MAX_LINE_LEN + 1];
    int lineLen = 0;
    <Xóa dòng>
    for (;;) {
        wordLen = ReadWord(word);
        if (<Hết từ>) {
            <In dòng không căn lề>
            return 0;
        }
        if (<Từ không vừa dòng>) {
            < In dòng có căn lề >
            <Xóa dòng>
        }
        AddWord(word, 1);
    }
    return 0;
}
```

- Quay lại main().
<Thêm từ vào dòng> có nghĩa là gì ?
- Tạo 1 hàm riêng cho việc đó :

AddWord(word, line, &lineLen)

```
void AddWord(const char *word, char *line, int *lineLen) {
    <Nếu dòng đã chứa 1 số từ, thêm 1 dấu trắng>
    strcat(line, word);
    (*lineLen) += strlen(word);
}
```

Tình chỉnh từng bước: Nhớ từ

- AddWord()

```
void AddWord(const char *word, char *line, int *lineLen) {  
  
    /* Nếu dòng đã chứa 1 số từ, thêm 1 dấu trắng. */  
    if (*lineLen > 0) {  
        line[*lineLen] = ' '  
        line[*lineLen + 1] = '\\0';  
        (*lineLen)++;  
    }  
  
    strcat(line, word);  
    (*lineLen) += strlen(word);  
}
```

Tính chỉnh từng bước: In dòng cuối cùng

- <Hết từ> và <In dòng không căn lề> nghĩa là gì ?
- Tạo các hàm để thực hiện

```
...
int main(void) {
    char word[MAX_WORD_LEN + 1];
    int wordLen;
    char line[MAX_LINE_LEN + 1];
    int lineLen = 0;
    <Xóa dòng>
    for (;;) {
        wordLen = ReadWord(word);

        /* Nếu hết từ, in dòng không căn lề. */
        if ((wordLen == 0) && (lineLen > 0)) {
            puts(line);
            return 0;
        }
        if (<Từ không vừa dòng>) {
            <In dòng có căn lề>
            <Xóa dòng>
        }
        AddWord(word, line, &lineLen);
    }
    return 0;
}
```

Tính chỉnh từng bước:

- Quyết định khi nào thì in

```
...
int main(void) {
    char word[MAX_WORD_LEN + 1];
    int wordLen;
    char line[MAX_LINE_LEN + 1];
    int lineLen = 0;
    <Xóa dòng>
    for (;;) {
        wordLen = ReadWord(word);
        /* If no more words, print line
           with no justification. */
        if ((wordLen == 0) && (lineLen > 0)) {
            puts(line);
            return 0;
        }
        /* Nếu từ không vừa dòng, thì ... */
        if ((wordLen + 1 + lineLen) > MAX_LINE_LEN) {
            <In dòng có căn lề>
            < Xóa dòng >
        }
        AddWord(word, line, &lineLen);
    }
    return 0;
}
```

- <Từ không vừa dòng>
Nghĩa là gì?



Tính chỉnh từng bước:

- In dòng có căn lề

- <In dòng có căn lề> nghĩa là gì ?
- Rõ ràng hàm này cần biết trong dòng hiện tại có bao nhiêu từ. Vì vậy ta thêm **numWords** vào hàm main ...

```
...
int main(void) {
    ...
    int numWords = 0;
    <Xóa dòng>
    for (;;) {
        ...
        /* Nếu từ không vừa dòng, thì... */
        if ((wordLen + 1 + lineLen) > MAX_LINE_LEN) {
            WriteLine(line, lineLen, numWords);
            <Xóa dòng>
        }

        AddWord(word, line, &lineLen);
        numWords++;
    }
    return 0;
}
```

Tính chỉnh từng bước:

- In dòng có căn lề
- Viết pseudocode cho WriteLine()...

```
void WriteLine(const char *line, int lineLen, int numWords) {  
  
    <Tính số khoảng trống dư thừa cho dòng>  
  
    for (i = 0; i < lineLen; i++) {  
        if (<line[i] is not a space>)  
            <Print the character>  
        else {  
            <Tính số khoảng trống cần bù thêm>  
  
            <In 1 space, cộng thêm các spaces cần bù>  
  
            <Giảm thêm không gian và đếm số từ>  
        }  
    }  
}
```

Printing with Justification (cont.)

```
void WriteLine(const char *line, int lineLen, int numWords) {
    int extraSpaces, spacesToInsert, i, j;
    /* Tính số khoảng trống dư thừa cho dòng. */
    extraSpaces = MAX_LINE_LEN - lineLen;
    for (i = 0; i < lineLen; i++) {
        if (line[i] != ' ')
            putchar(line[i]);
        else {
            /* Tính số khoảng trống cần thêm. */
            spacesToInsert = extraSpaces / (numWords - 1);

            /* In 1 space, cộng thêm các spaces phụ. */
            for (j = 1; j <= spacesToInsert + 1; j++)
                putchar(' ');

            /* Giảm bớt spaces và đếm từ. */
            extraSpaces -= spacesToInsert;
            numWords--;
        }
    }
    putchar('\n');
}
```

Số lượng các
khoảng trống

VD:
Nếu extraSpaces = 10
và numWords = 5,
thì space bù sẽ là
2, 2, 3, and 3 tương ứng

Clearing the Line

- Cuối cùng. <Xóa dòng> nghĩa là gì ?
- Tuy đơn giản, nhưng ta cũng viết thành 1 hàm

```
...
int main(void) {
    ...
    int numWords = 0;
    ClearLine(line, &lineLen, &numWords);
    for (;;) {
        ...
        /* If word doesn't fit on this line, then... */
        if ((wordLen + 1 + lineLen) > MAX_LINE_LEN) {
            WriteLine(line, lineLen, numWords);
            ClearLine(line, &lineLen, &numWords);
        }

        addWord(word, line, &lineLen, &numWords);
        numWords++;
    }
    return 0;
}
```

```
void ClearLine(char *line, int *lineLen, int *numWords) {
    line[0] = '\\0';
    *lineLen = 0;
    *numWords = 0;
}
```

Cấu trúc chương trình dựa trên phương pháp modul hóa

- Với người sử dụng CT
 - Input: Văn bản với định dạng lộn xộn
 - Output: Cùng nội dung, nhưng trình bày căn lề trái, phải, rõ ràng, sáng sủa
- Giữa các phần của CT
 - Các hàm xử lý từ : Word-handling functions
 - Các hàm xử lý dòng : Line-handling functions
 - main() function
- Lợi ích của modularity
 - Đọc code: dễ dàng, qua các mẫu nhỏ, riêng biệt
 - Testing : Test từng hàm riêng biệt
 - Tăng tốc độ: Chỉ tập trung vào các phần tốc độ còn chậm
 - Mở rộng: Chỉ thay đổi các phần liên quan



3. Thiết kế dữ liệu

- Bài toán: cho các bộ dữ liệu mẫu như sau:
 - (tên sinh viên, điểm)
 - ("john smith", 84)
 - ("jane doe", 93)
 - ("bill clinton", 81)
 - ...
 - (tên cầu thủ, vị trí chơi trên sân)
 - ("Ruth", 3)
 - ("Gehrig", 4)
 - ("Mantle", 7)
 - ...
 - (tên biển, giá trị)
 - ("maxLength", 2000)
 - ("i", 7)
 - ("j", -10)
 - ...
 - ...
- Cần thiết kế cấu trúc dữ liệu cho phép thực hiện các thao tác sau:
 - Create: Tạo mới các bộ dữ liệu
 - Add: Thêm mới các dữ liệu thành phần
 - Search: Tìm kiếm các dữ liệu thành phần
 - Free: Hủy cấu trúc dữ liệu



3. Thiết kế dữ liệu

- Bài toán: cho các bộ dữ liệu mẫu như sau:
 - (tên sinh viên, điểm)
 - ("john smith", 84)
 - ("jane doe", 93)
 - ("bill clinton", 81)
 - ...
 - (tên cầu thủ, vị trí chơi trên sân)
 - ("Ruth", 3)
 - ("Gehrig", 4)
 - ("Mantle", 7)
 - ...
 - (tên biển, giá trị)
 - ("maxLength", 2000)
 - ("i", 7)
 - ("j", -10)
 - ...
 - ...
- Cấu trúc dữ liệu: cho phép lưu trữ các bảng có chứa các cặp: khóa (key) và giá trị tương ứng với khóa (value)
 - Mỗi khóa là 1 xâu, mỗi giá trị là 1 số nguyên
 - Không biết trước số cặp khóa/giá trị
 - Có/Không cho phép có khóa trùng lặp

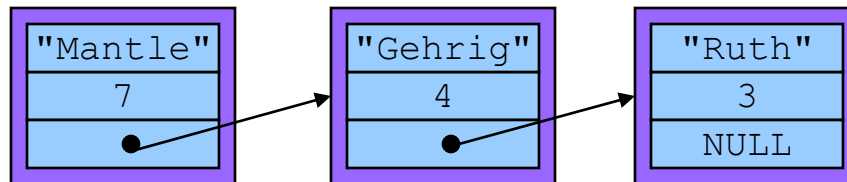
→ Linked list

→ Hash table

→ Expanding array

Data Structure #1: Linked List

- Data structure: Các nút, mỗi nút chứa cặp key/value và con trỏ trỏ đến nút tiếp theo



- Algorithms:
 - Create: Cấp phát nút giả trỏ đến nút thật đầu tiên
 - Add: Tạo nút mới, thêm vào đầu danh sách
 - Search: Tìm kiếm tuyến tính
 - Free: Giải phóng các nút trong khi duyệt, giải phóng các nút giả



Linked List: Data Structure

```
struct Node {  
    const char *key;  
    int value;  
    struct Node *next;  
};  
  
struct Table {  
    struct Node *first;  
};
```

Linked List: Create (1)

```
struct Table *Table_create(void) {  
    struct Table *t;  
    t = (struct Table*)  
        malloc(sizeof(struct Table));  
    t->first = NULL;  
    return t;  
}
```

```
struct Table *t;  
...  
t = Table_create();  
...
```



Linked List: Create (2)

```
struct Table *Table_create(void) {  
    struct Table *t;  
    t = (struct Table*)  
        malloc(sizeof(struct Table));  
    t->first = NULL;  
    return t;  
}
```

```
struct Table *t;  
...  
t = Table_create();  
...
```



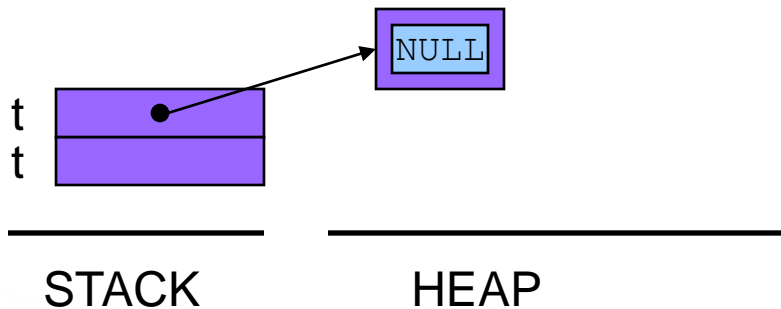
STACK

HEAP

Linked List: Create (3)

```
struct Table *Table_create(void) {  
    struct Table *t;  
    t = (struct Table*)  
        malloc(sizeof(struct Table));  
    t->first = NULL;  
    return t;  
}
```

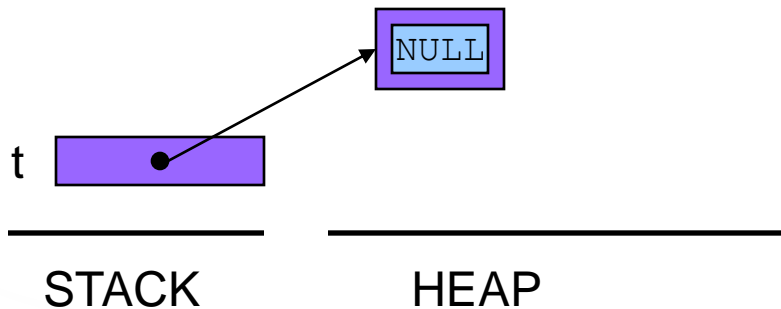
```
struct Table *t;  
...  
t = Table_create();  
...
```



Linked List: Create (4)

```
struct Table *Table_create(void) {  
    struct Table *t;  
    t = (struct Table*)  
        malloc(sizeof(struct Table));  
    t->first = NULL;  
    return t;  
}
```

```
struct Table *t;  
...  
t = Table_create();  
...
```

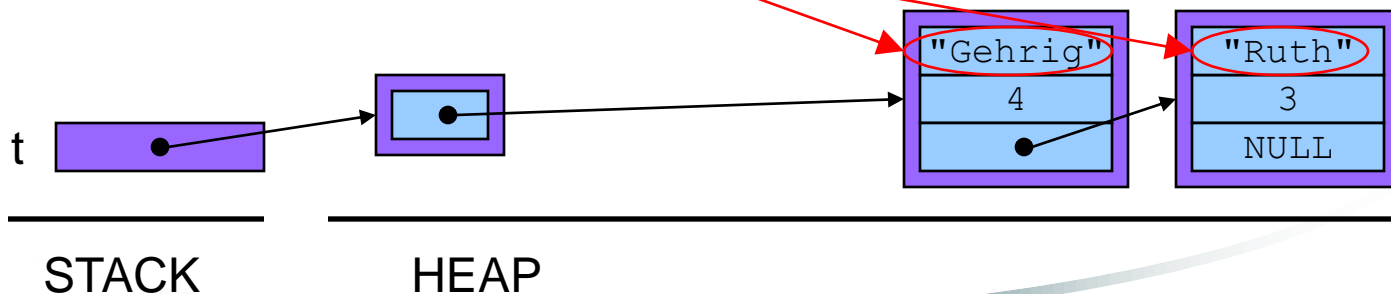


Linked List: Add (1)

```
void Table_add(struct Table *t,  
    const char *key, int value) {  
    struct Node *p = (struct Node*)malloc(sizeof(struct Node));  
    p->key = key;  
    p->value = value;  
    p->next = t->first;  
    t->first = p;  
}
```

Các con trỏ trỏ đến
các chuỗi nằm trong
vùng RODATA

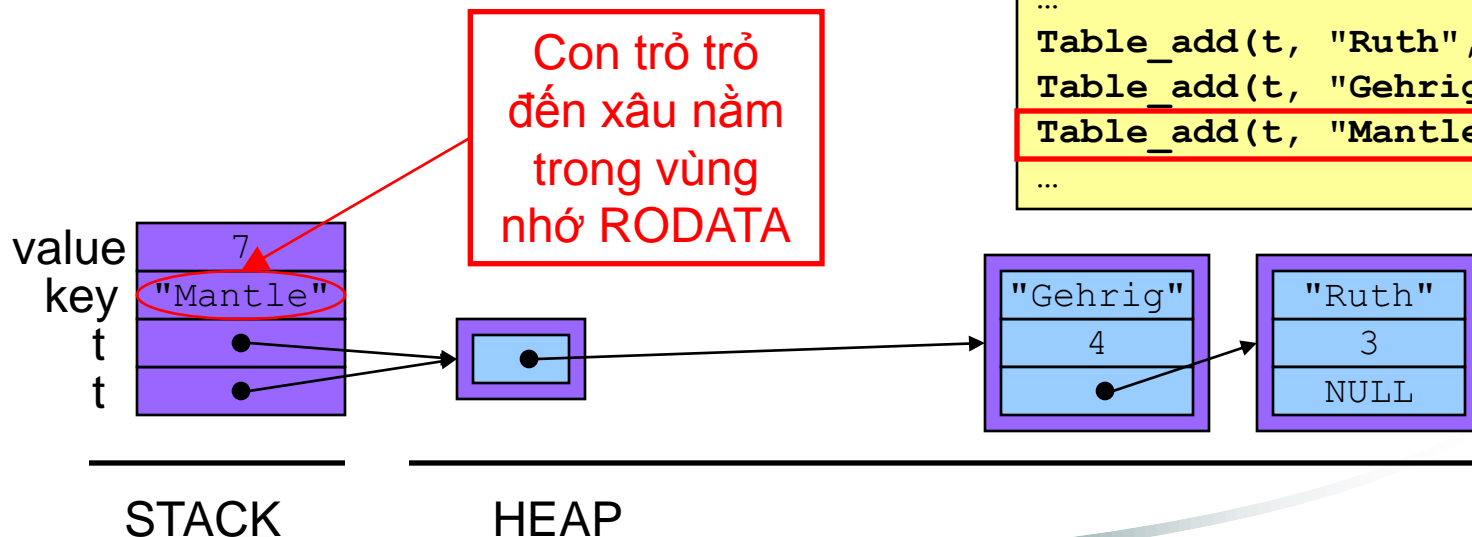
```
struct Table *t;  
...  
Table_add(t, "Ruth", 3);  
Table_add(t, "Gehrig", 4);  
Table_add(t, "Mantle", 7);  
...
```



Linked List: Add (2)

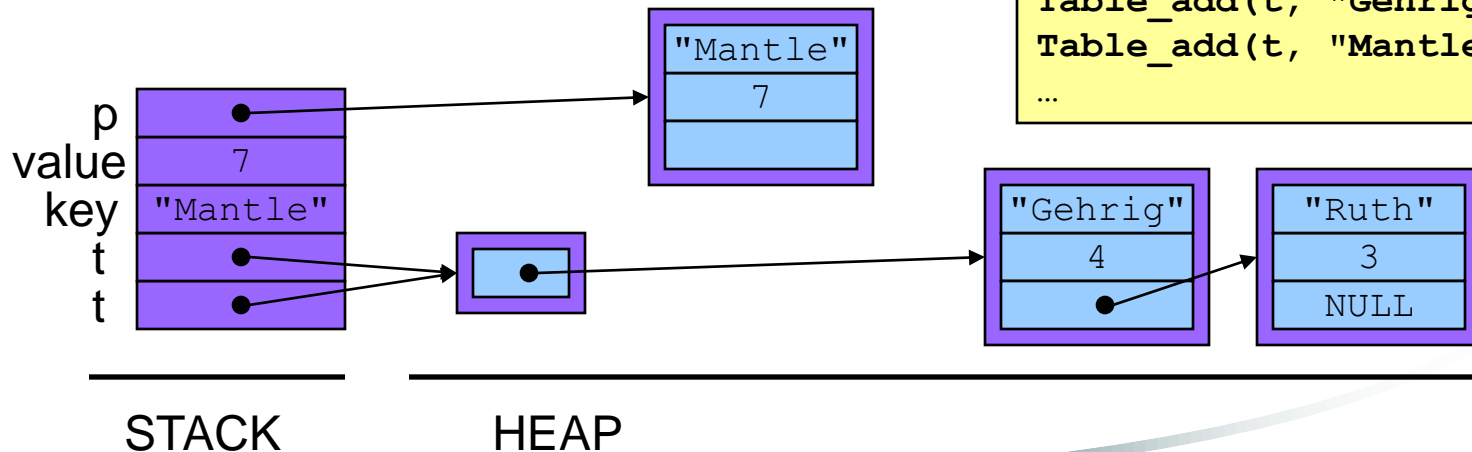
```
void Table_add(struct Table *t,  
    const char *key, int value) {  
    struct Node *p = (struct Node*)malloc(sizeof(struct Node));  
    p->key = key;  
    p->value = value;  
    p->next = t->first;  
    t->first = p;  
}
```

```
struct Table *t;  
...  
Table_add(t, "Ruth", 3);  
Table_add(t, "Gehrig", 4);  
Table_add(t, "Mantle", 7);  
...
```



Linked List: Add (3)

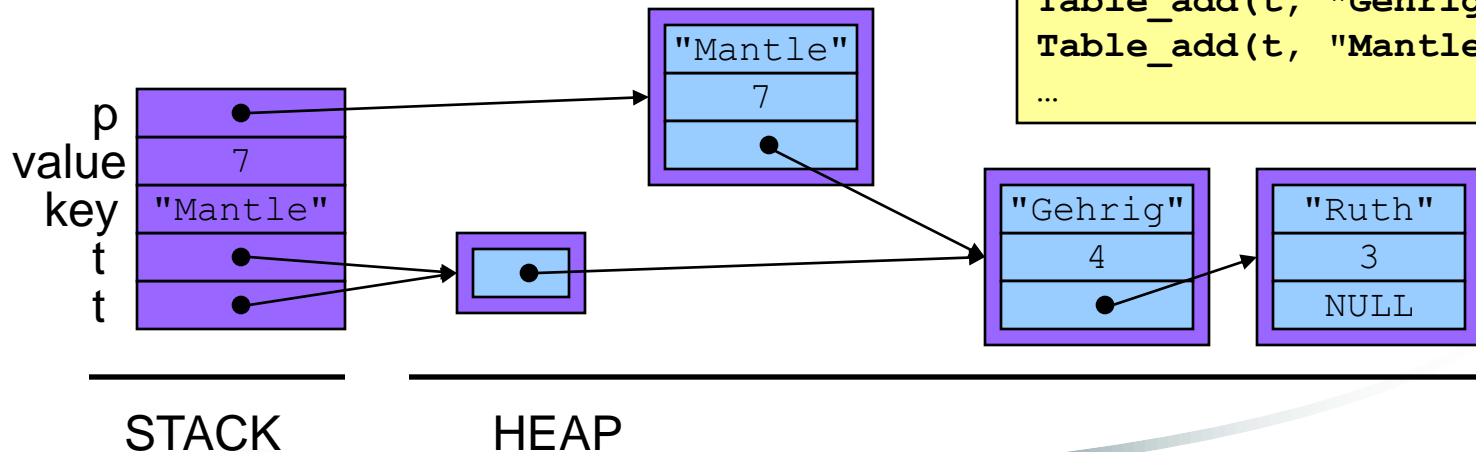
```
void Table_add(struct Table *t,  
    const char *key, int value) {  
    struct Node *p = (struct Node*)malloc(sizeof(struct Node));  
    p->key = key;  
    p->value = value;  
    p->next = t->first;  
    t->first = p;  
}
```



```
struct Table *t;  
...  
Table_add(t, "Ruth", 3);  
Table_add(t, "Gehrig", 4);  
Table_add(t, "Mantle", 7);  
...
```

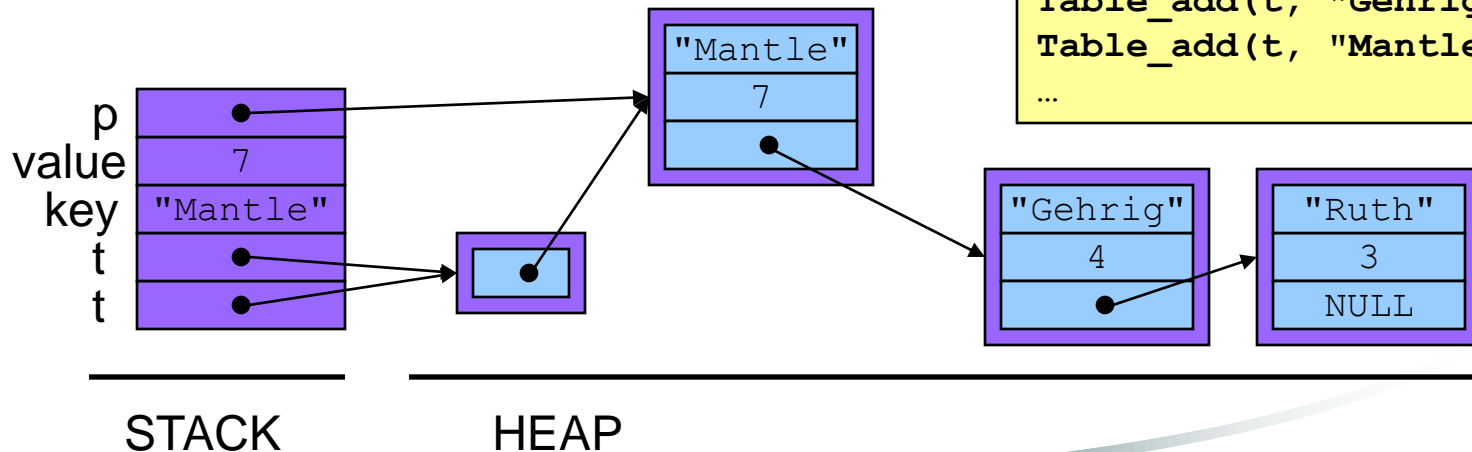

Linked List: Add (4)

```
void Table_add(struct Table *t,  
    const char *key, int value) {  
    struct Node *p = (struct Node*)malloc(sizeof(struct Node));  
    p->key = key;  
    p->value = value;  
    p->next = t->first;  
    t->first = p;  
}
```



Linked List: Add (5)

```
void Table_add(struct Table *t,  
    const char *key, int value) {  
    struct Node *p = (struct Node*)malloc(sizeof(struct Node));  
    p->key = key;  
    p->value = value;  
    p->next = t->first;  
    t->first = p;  
}
```

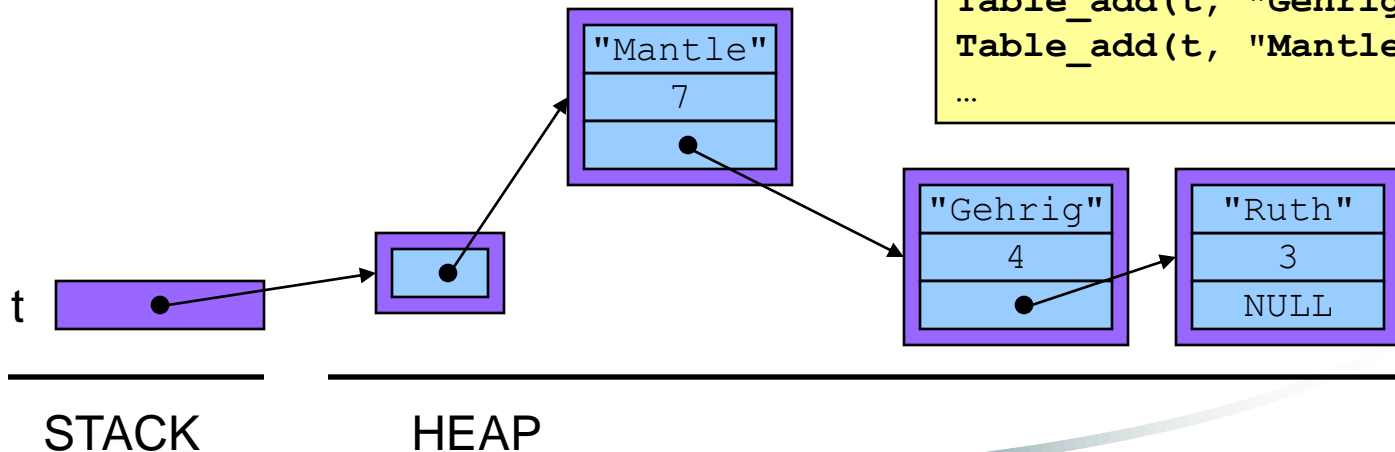


```
struct Table *t;  
...  
Table_add(t, "Ruth", 3);  
Table_add(t, "Gehrig", 4);  
Table_add(t, "Mantle", 7);  
...
```

Linked List: Add (6)

```
void Table_add(struct Table *t,  
    const char *key, int value) {  
    struct Node *p = (struct Node*)malloc(sizeof(struct Node));  
    p->key = key;  
    p->value = value;  
    p->next = t->first;  
    t->first = p;  
}
```

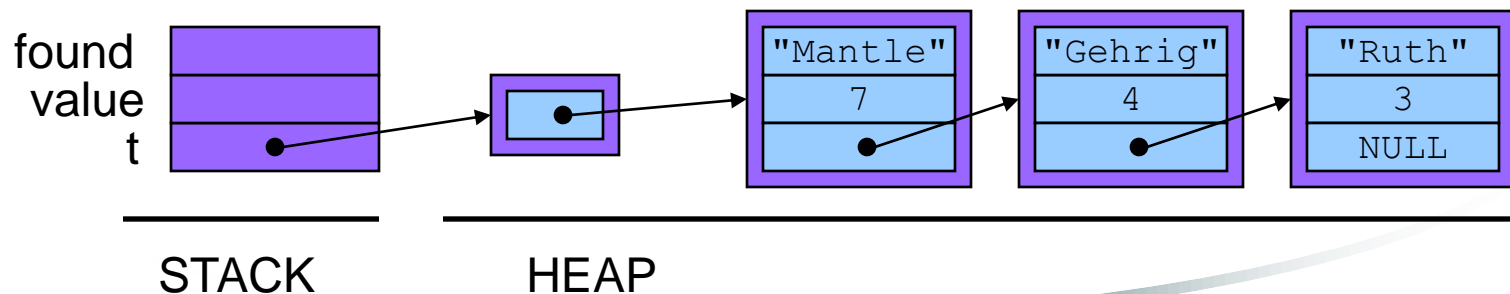
```
struct Table *t;  
...  
Table_add(t, "Ruth", 3);  
Table_add(t, "Gehrig", 4);  
Table_add(t, "Mantle", 7);  
...
```



Linked List: Search (1)

```
int Table_search(struct Table *t,  
    const char *key, int *value) {  
    struct Node *p;  
    for (p = t->first; p != NULL; p = p->next)  
        if (strcmp(p->key, key) == 0) {  
            *value = p->value;  
            return 1;  
        }  
    return 0;  
}
```

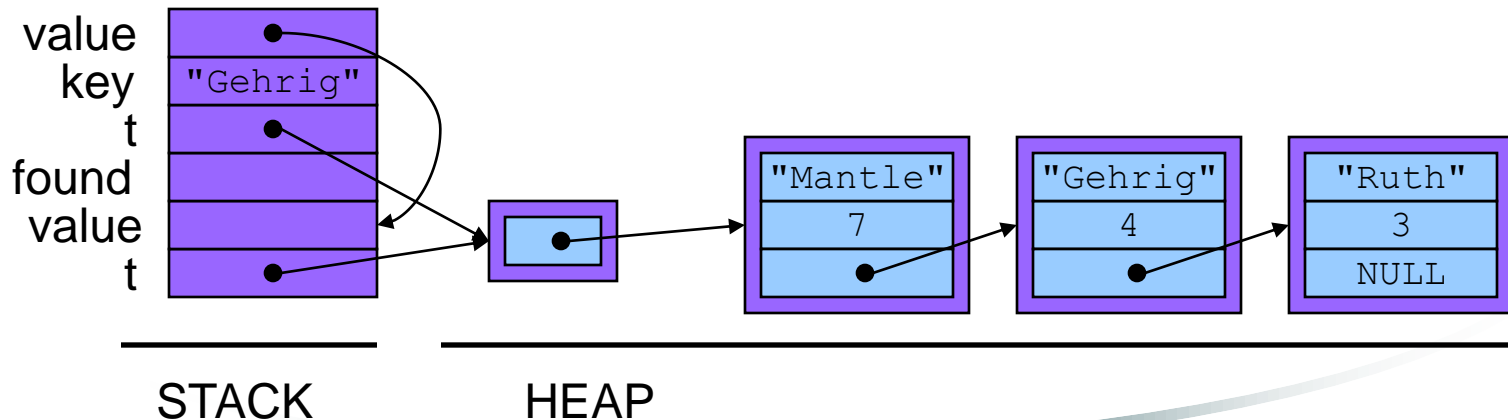
```
struct Table *t;  
int value;  
int found;  
...  
found =  
    Table_search(t, "Gehrig", &value);  
...
```



Linked List: Search (2)

```
int Table_search(struct Table *t,  
    const char *key, int *value) {  
    struct Node *p;  
    for (p = t->first; p != NULL; p = p->next)  
        if (strcmp(p->key, key) == 0) {  
            *value = p->value;  
            return 1;  
        }  
    return 0;  
}
```

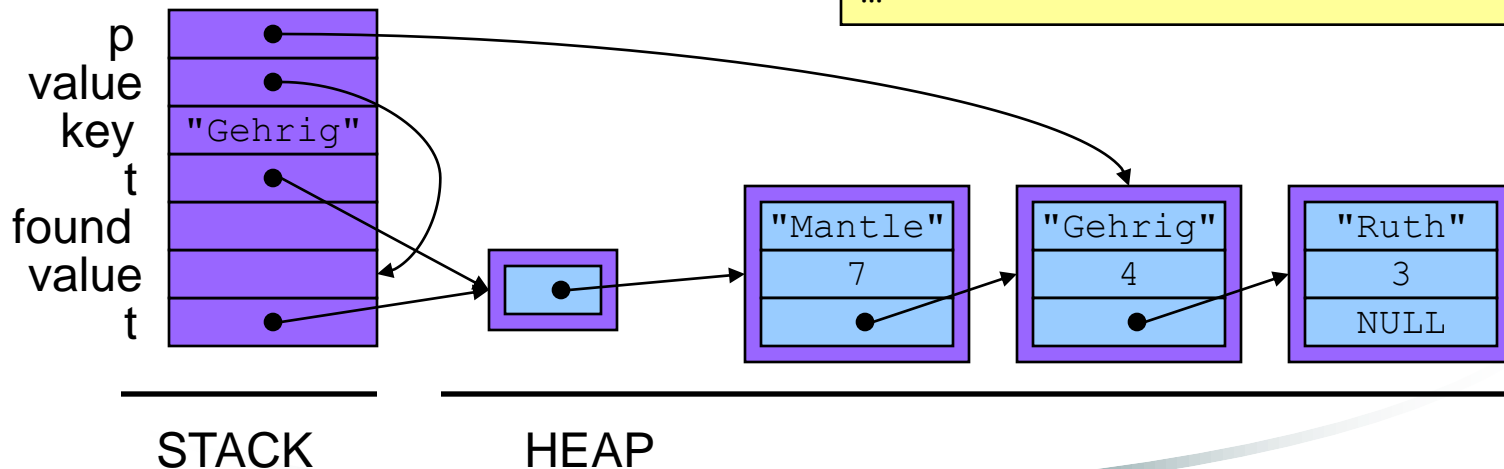
```
struct Table *t;  
int value;  
int found;  
...  
found =  
    Table_search(t, "Gehrig", &value);  
...
```



Linked List: Search (3)

```
int Table_search(struct Table *t,  
    const char *key, int *value) {  
    struct Node *p;  
    for (p = t->first; p != NULL; p = p->next)  
        if (strcmp(p->key, key) == 0) {  
            *value = p->value;  
            return 1;  
        }  
    return 0;  
}
```

```
struct Table *t;  
int value;  
int found;  
...  
found =  
    Table_search(t, "Gehrig", &value);  
...
```

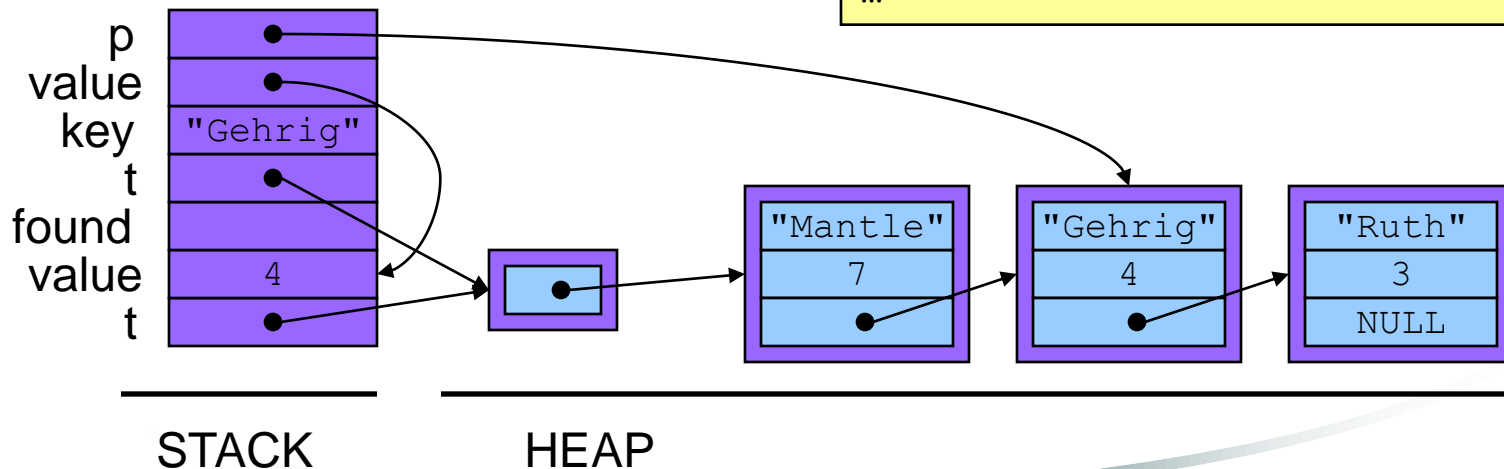


Linked List: Search (4)

```
int Table_search(struct Table *t,
    const char *key, int *value) {
    struct Node *p;
    for (p = t->first; p != NULL; p = p->next)
        if (strcmp(p->key, key) == 0) {
            *value = p->value;
            return 1;
        }
    return 0;
}
```

```
struct Table *t;
int value;
int found;

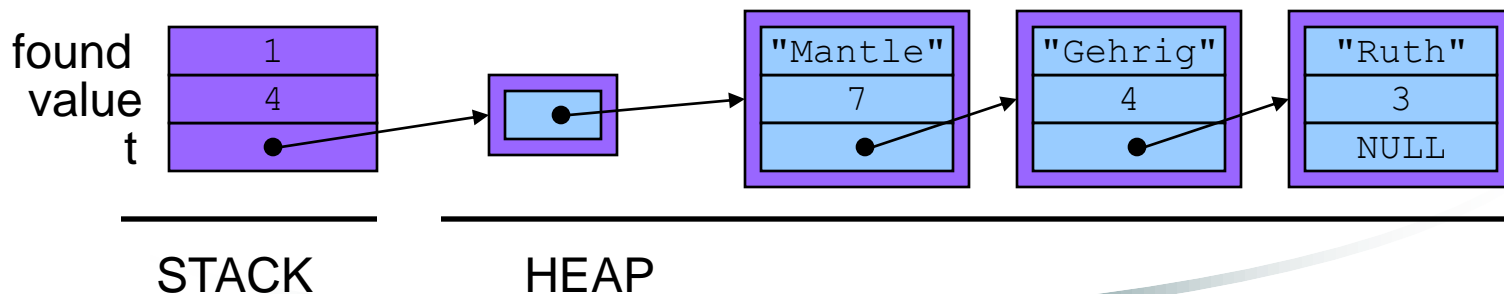
...
found =
    Table_search(t, "Gehrig", &value);
...
```



Linked List: Search (5)

```
int Table_search(struct Table *t,
    const char *key, int *value) {
    struct Node *p;
    for (p = t->first; p != NULL; p = p->next)
        if (strcmp(p->key, key) == 0) {
            *value = p->value;
            return 1;
        }
    return 0;
}
```

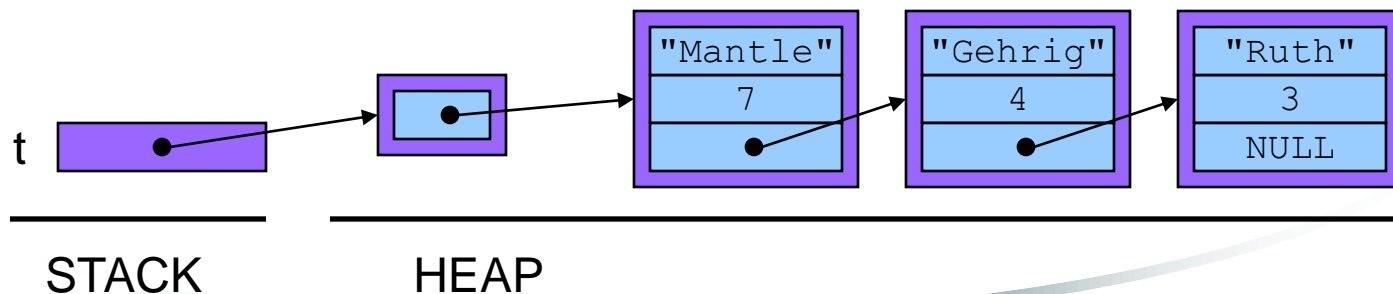
```
struct Table *t;
int value;
int found;
...
found =
    Table_search(t, "Gehrig", &value);
...
```



Linked List: Free (1)

```
void Table_free(struct Table *t) {  
    struct Node *p;  
    struct Node *nextp;  
    for (p = t->first; p != NULL; p = nextp) {  
        nextp = p->next;  
        free(p);  
    }  
    free(t); /* Free the dummy node */  
}
```

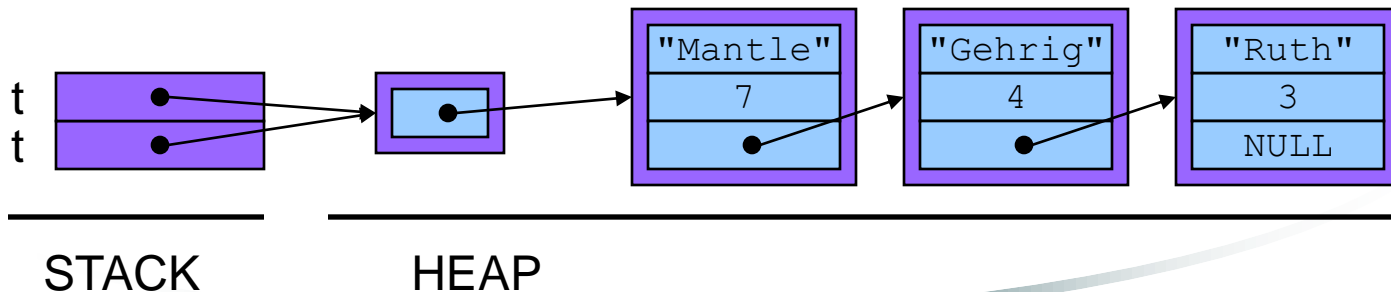
```
struct Table *t;  
...  
Table_free(t);  
...
```



Linked List: Free (2)

```
void Table_free(struct Table *t) {  
    struct Node *p;  
    struct Node *nextp;  
    for (p = t->first; p != NULL; p = nextp) {  
        nextp = p->next;  
        free(p);  
    }  
    free(t); /* Free the dummy node */  
}
```

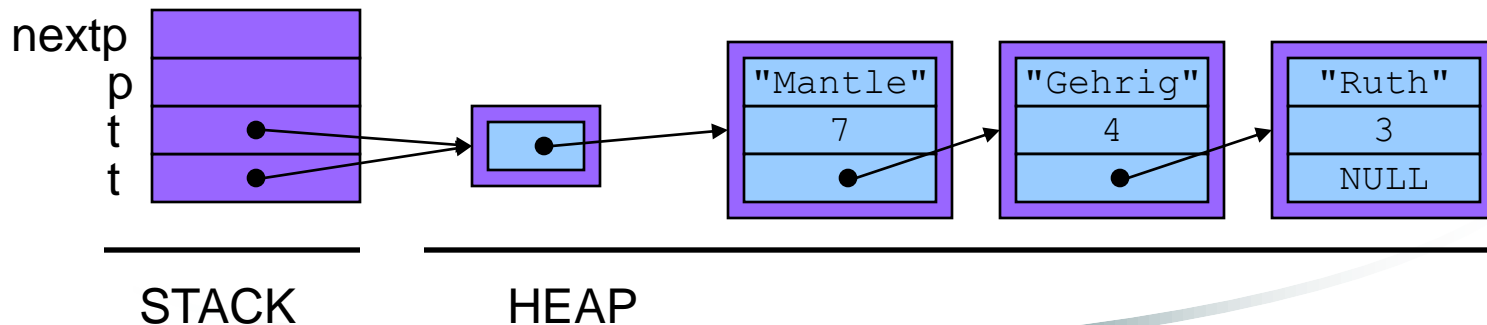
```
struct Table *t;  
...  
Table_free(t);  
...
```



Linked List: Free (3)

```
void Table_free(struct Table *t) {  
    struct Node *p;  
    struct Node *nextp;  
    for (p = t->first; p != NULL; p = nextp) {  
        nextp = p->next;  
        free(p);  
    }  
    free(t); /* Free the dummy node */  
}
```

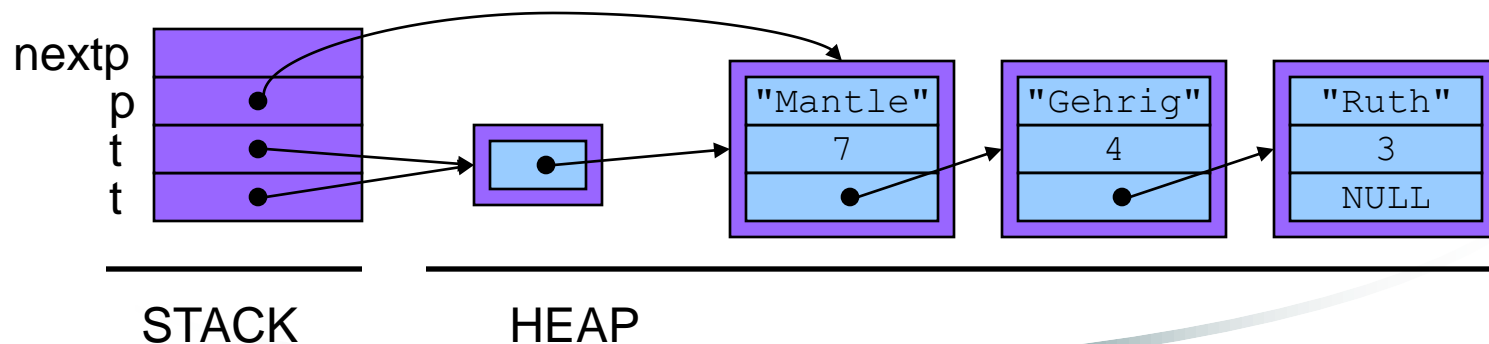
```
struct Table *t;  
...  
Table_free(t);  
...
```



Linked List: Free (4)

```
void Table_free(struct Table *t) {  
    struct Node *p;  
    struct Node *nextp;  
    for (p = t->first; p != NULL; p = nextp) {  
        nextp = p->next;  
        free(p);  
    }  
    free(t); /* Free the dummy node */  
}
```

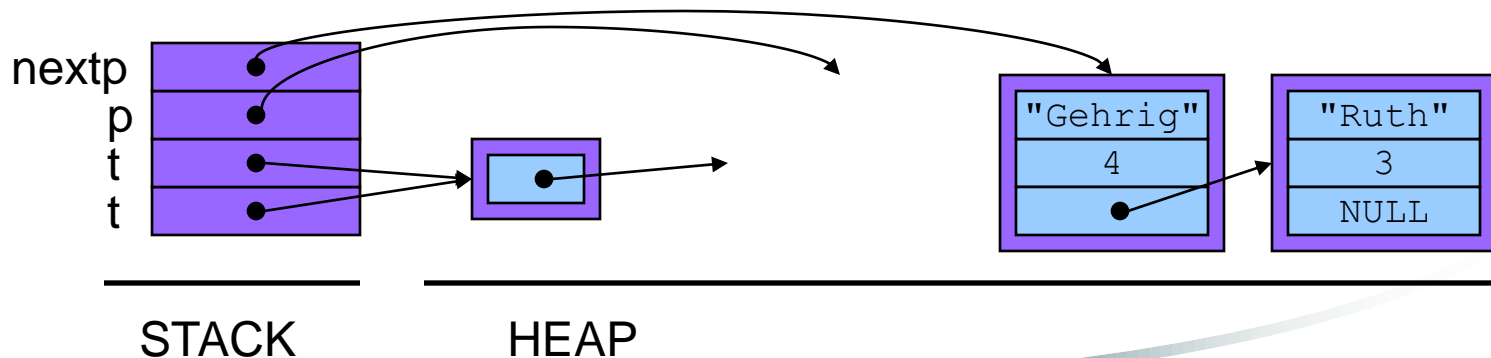
```
struct Table *t;  
...  
Table_free(t);  
...
```



Linked List: Free (5)

```
void Table_free(struct Table *t) {  
    struct Node *p;  
    struct Node *nextp;  
    for (p = t->first; p != NULL; p = nextp) {  
        nextp = p->next;  
        free(p);  
    }  
    free(t); /* Free the dummy node */  
}
```

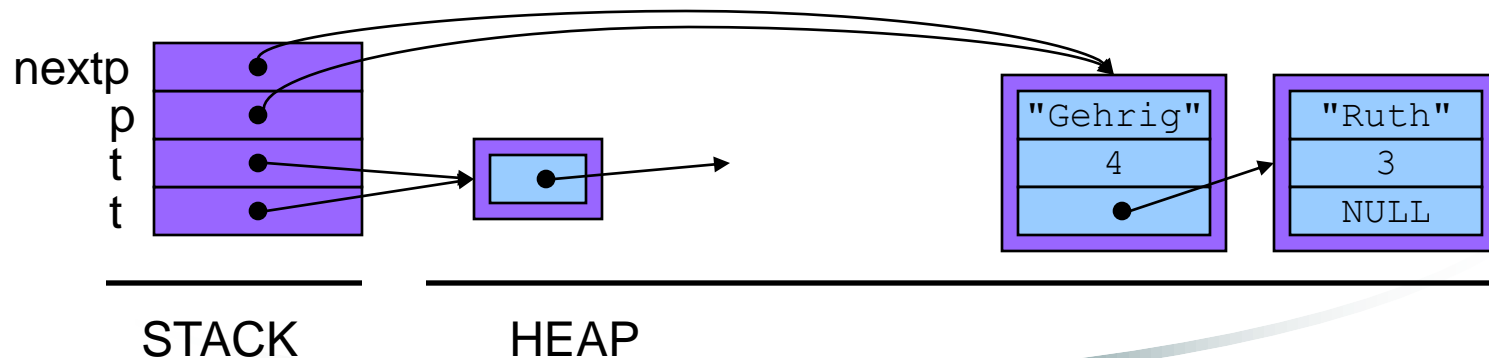
```
struct Table *t;  
...  
Table_free(t);  
...
```



Linked List: Free (6)

```
void Table_free(struct Table *t) {  
    struct Node *p;  
    struct Node *nextp;  
    for (p = t->first; p != NULL; p = nextp) {  
        nextp = p->next;  
        free(p);  
    }  
    free(t); /* Free the dummy node */  
}
```

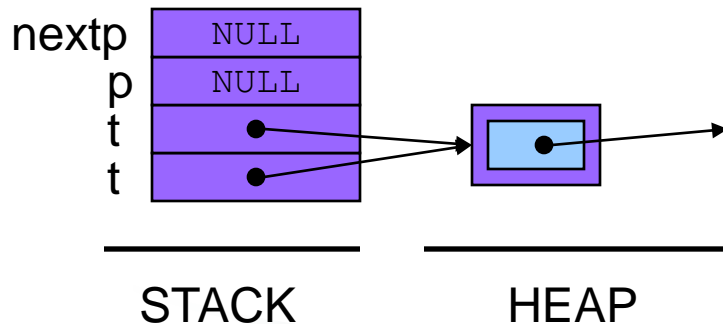
```
struct Table *t;  
...  
Table_free(t);  
...
```



Linked List: Free (7)

```
void Table_free(struct Table *t) {  
    struct Node *p;  
    struct Node *nextp;  
    for (p = t->first; p != NULL; p = nextp) {  
        nextp = p->next;  
        free(p);  
    }  
    free(t); /* Free the dummy node */  
}
```

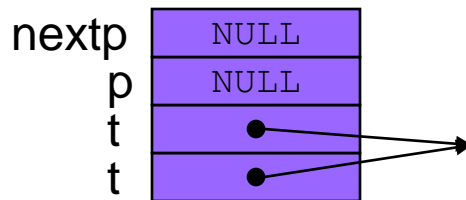
```
struct Table *t;  
...  
Table_free(t);  
...
```



Linked List: Free (8)

```
void Table_free(struct Table *t) {  
    struct Node *p;  
    struct Node *nextp;  
    for (p = t->first; p != NULL; p = nextp) {  
        nextp = p->next;  
        free(p);  
    }  
    free(t); /* Free the dummy node */  
}
```

```
struct Table *t;  
...  
Table_free(t);  
...
```



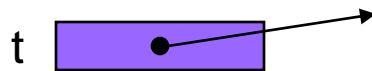
STACK

HEAP

Linked List: Free (9)

```
void Table_free(struct Table *t) {  
    struct Node *p;  
    struct Node *nextp;  
    for (p = t->first; p != NULL; p = nextp) {  
        nextp = p->next;  
        free(p);  
    }  
    free(t); /* Free the dummy node */  
}
```

```
struct Table *t;  
...  
Table_free(t);  
...
```



STACK

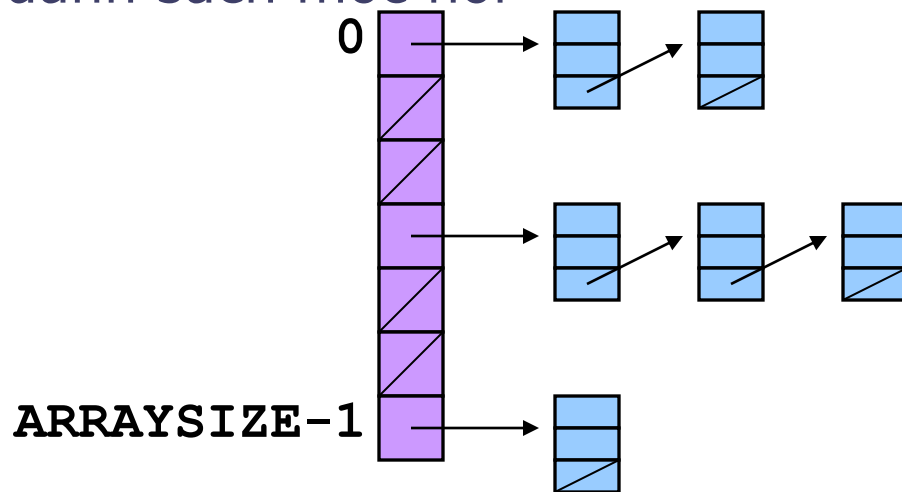
HEAP

Hiệu năng của chương trình khi cài đặt CTDL bằng Linked List

- Phân tích độ phức tạp về thời gian:
 - Create: $O(1)$, nhanh
 - Add: $O(1)$, nhanh
 - Search: $O(n)$, chậm
 - Free: $O(n)$, chậm
- Giải pháp khác: luôn giữ các nút được sắp xếp theo thứ tự tăng/giảm dần của khóa
 - Create: $O(1)$, nhanh
 - Add: $O(n)$, chậm; cần duyệt danh sách trước khi tìm được chỗ để thêm
 - Search: $O(n)$, vẫn chậm; cần duyệt một phần danh sách
 - Free: $O(n)$, chậm

Data Structure #2: Bảng băm (Hash Table)

- Mảng có kích thước cố định, mỗi phần tử của mảng trỏ đến một danh sách móc nối



```
struct Node *array[ARRAYSIZE];
```

- Hàm ánh xạ một khóa vào 1 chỉ số mảng
 - Vì khóa là xâu, tìm hàm băm hợp lý
 - Tìm đến phần tử i của mảng để thực hiện các thao tác, tức là thao tác trên danh sách móc nối `hashtab[i]`

Băm khóa kiểu string thành giá trị kiểu int

- Hàm băm đơn giản sẽ không đưa ra được nhiều khóa phân biệt
 - Số các ký tự trong xâu % ARRAYSIZE
 - Tổng các giá trị ASCII của các ký tự trong xâu % ARRAYSIZE
 - ...
- Hàm băm hợp lý
 - Tổng các giá trị có tính đến trọng số của các ký tự trong xâu: $\mathbf{x_i}$ giá trị ASCII của ký tự, \mathbf{a} trọng số của ký tự, \mathbf{i} vị trí trong xâu
 - $(\sum a^i x_i) \bmod \text{ARRAYSIZE}$
 - Trường hợp tốt nhất: \mathbf{a} và ARRAYSIZE đều là số nguyên tố
 - E.g., $\mathbf{a = 65599}$, $\text{ARRAYSIZE} = 1024$

Cài đặt hàm băm

- Phải trả giá đắt cho việc tính a^i cho mỗi chỉ số i
 - Thay vì tính a^i cho mỗi chỉ số i
 - Tính $((x[0] * 65599 + x[1]) * 65599 + x[2]) * 65599 + x[3]) * 65599 + \dots$

```
unsigned int hash(const char *x) {  
    int i;  
    unsigned int h = 0U;  
    for (i=0; x[i]!='\0'; i++)  
        h = h * 65599 + (unsigned char)x[i];  
    return h % 1024;  
}
```

Ví dụ về bảng băm

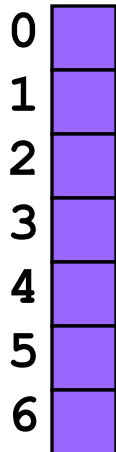
Cho `ARRAYSIZE = 7`

Tìm kiếm và thêm vào nếu không tìm thấy các xâu sau: the, cat, in, the, hat

Bảng băm ban đầu là rỗng

Từ 1: `hash("the") = 965156977`. $965156977 \% 7 = 1$.

Tìm xâu "the" trong DS móc nối `table[1]` : không tìm thấy



0	
1	
2	
3	
4	
5	
6	

Ví dụ về bảng băm

Cho `ARRAYSIZE = 7`

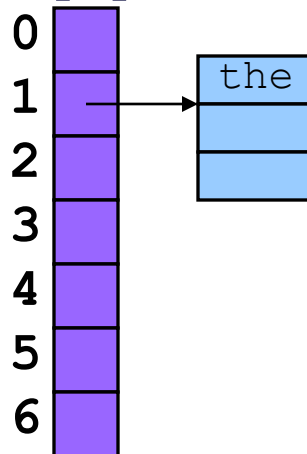
Tìm kiếm và thêm vào nếu không tìm thấy các xâu sau: the, cat, in, the, hat

Bảng băm ban đầu là rỗng

Từ 1: `hash("the") = 965156977`. $965156977 \% 7 = 1$.

Tìm xâu "the" trong DS móc nối `table[1]` : không tìm thấy

Now: `table[1] = makelink(key, value, table[1])`

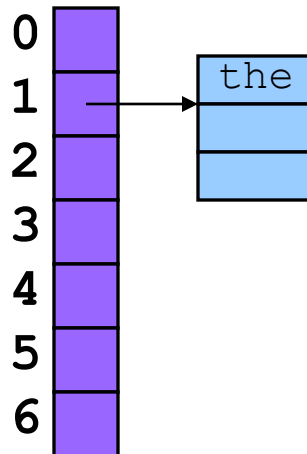


Ví dụ về bảng băm

Từ 2: "cat". $\text{hash}(\text{"cat"}) = 3895848756$. $3895848756 \% 7 = 2$.

Tìm trong DS móc nối table[2] xâu "cat": không tìm thấy

Now: $\text{table}[2] = \text{makelink}(\text{key}, \text{value}, \text{table}[2])$

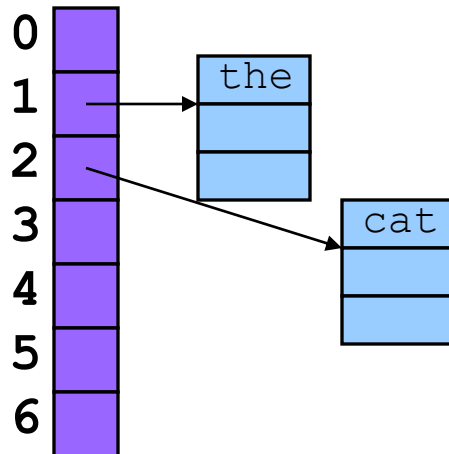


Ví dụ về bảng băm

Từ 3: "in". $\text{hash}(\text{"in"}) = 6888005$. $6888005 \% 7 = 5$.

Tìm xâu "in" trong DS móc nối `table[5]`: không thấy

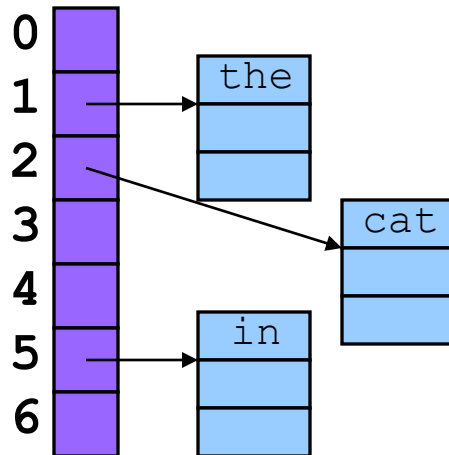
Now: `table[5] = makelink(key, value, table[5])`



Ví dụ về bảng băm

Từ 4: "the". $\text{hash}(\text{"the"}) = 965156977$. $965156977 \% 7 = 1$.

Tìm xâu "the" trong DS móc nối $\text{table}[1]$; có thấy



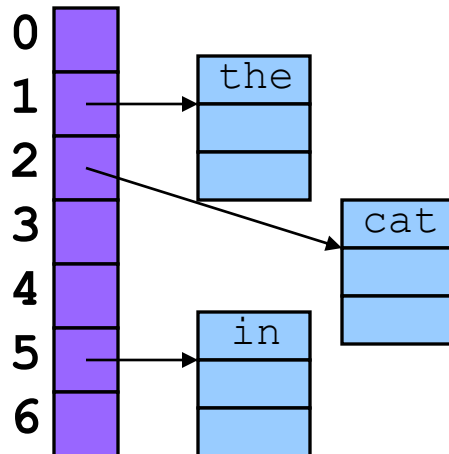
Ví dụ về bảng băm

Từ 4: "hat". $\text{hash}(\text{"hat"}) = 865559739.$ $865559739 \% 7 = 2.$

Tìm xâu "hat" trong DS móc nối table[2]; không thấy

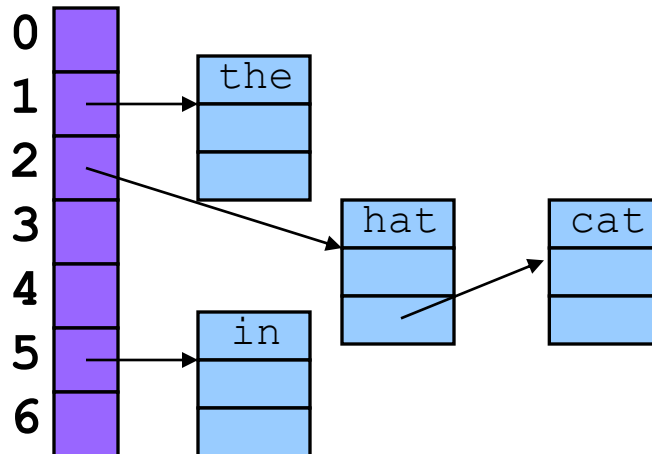
Now: Thêm "hat" vào table[2].

Thêm vào đâu ? Đầu hay cuối danh sách ?



Ví dụ về bảng băm

Thêm vào đầu danh sách để hơn



Bảng băm: cấu trúc dữ liệu

```
enum {BUCKET_COUNT = 1024};

struct Node {
    const char *key;
    int value;
    struct Node *next;
};

struct Table {
    struct Node *array[BUCKET_COUNT];
};
```

Bảng băm: Create (1)

```
struct Table *Table_create(void) {  
    struct Table *t;  
    t = (struct Table*)calloc(1, sizeof(struct Table));  
    return t;  
}
```

```
struct Table *t;  
...  
t = Table_create();  
...
```

t 

STACK

HEAP

Hash Table: Create (2)

```
struct Table *Table_create(void) {  
    struct Table *t;  
    t = (struct Table*)calloc(1, sizeof(struct Table));  
    return t;  
}
```

```
struct Table *t;  
...  
t = Table_create();  
...
```



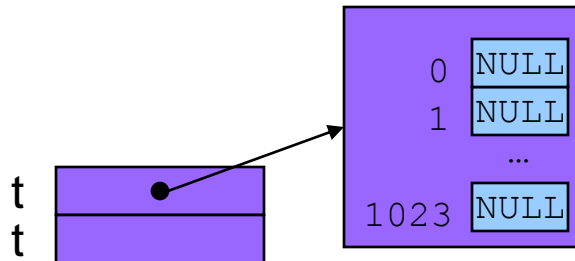
STACK

HEAP

Hash Table: Create (3)

```
struct Table *Table_create(void) {  
    struct Table *t;  
    t = (struct Table*)calloc(1, sizeof(struct Table));  
    return t;  
}
```

```
struct Table *t;  
...  
t = Table_create();  
...
```



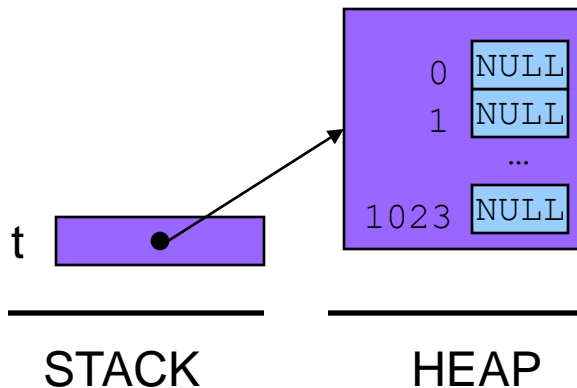
STACK

HEAP

Bảng băm: Create (4)

```
struct Table *Table_create(void) {  
    struct Table *t;  
    t = (struct Table*)calloc(1, sizeof(struct Table));  
    return t;  
}
```

```
struct Table *t;  
...  
t = Table_create();  
...
```

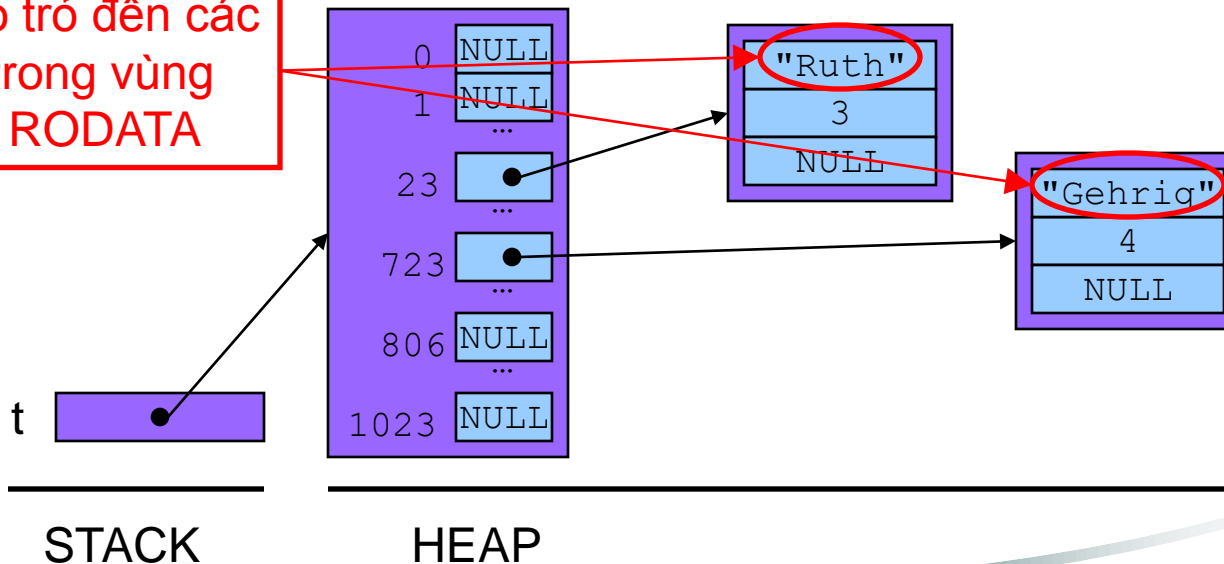


Bảng băm: Add (1)

```
void Table_add(struct Table *t,  
    const char *key, int value) {  
    struct Node *p = (struct Node*)malloc(sizeof(struct Node));  
    int h = hash(key);  
    p->key = key;  
    p->value = value;  
    p->next = t->array[h];  
    t->array[h] = p;  
}
```

```
struct Table *t;  
...  
Table_add(t, "Ruth", 3);  
Table_add(t, "Gehrig", 4);  
Table_add(t, "Mantle", 7);  
...
```

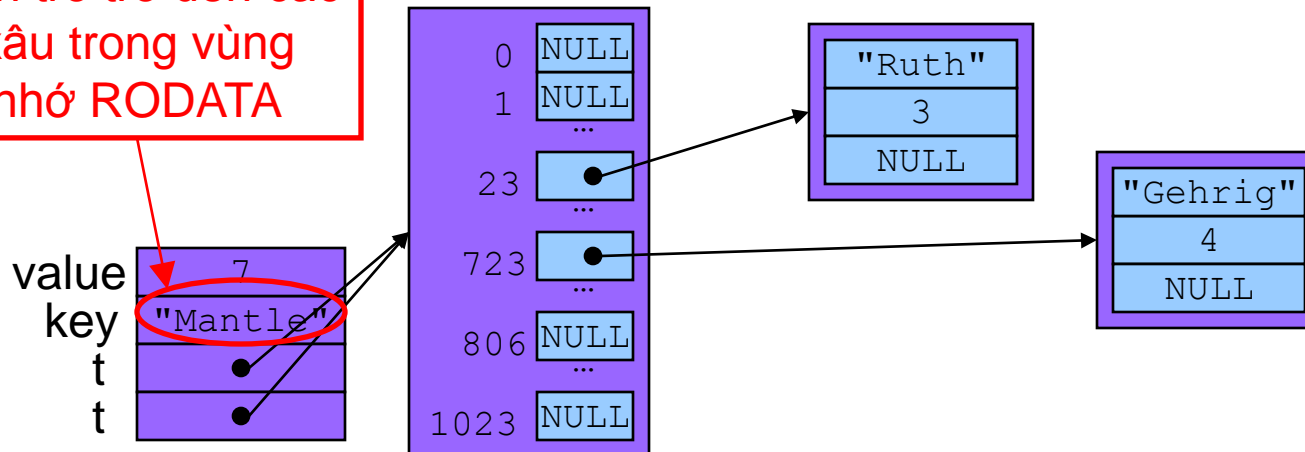
Con trỏ trỏ đến các
xâu trong vùng
nhớ RODATA



Bảng băm: Add (2)

```
void Table_add(struct Table *t,  
    const char *key, int value) {  
    struct Node *p = (struct Node*)malloc(sizeof(struct Node));  
    int h = hash(key);  
    p->key = key;  
    p->value = value;  
    p->next = t->array[h];  
    t->array[h] = p;  
}  
  
struct Table *t;  
...  
Table_add(t, "Ruth", 3);  
Table_add(t, "Gehrig", 4);  
Table_add(t, "Mantle", 7);  
...
```

Con trỏ trỏ đến các
xâu trong vùng
nhớ RODATA



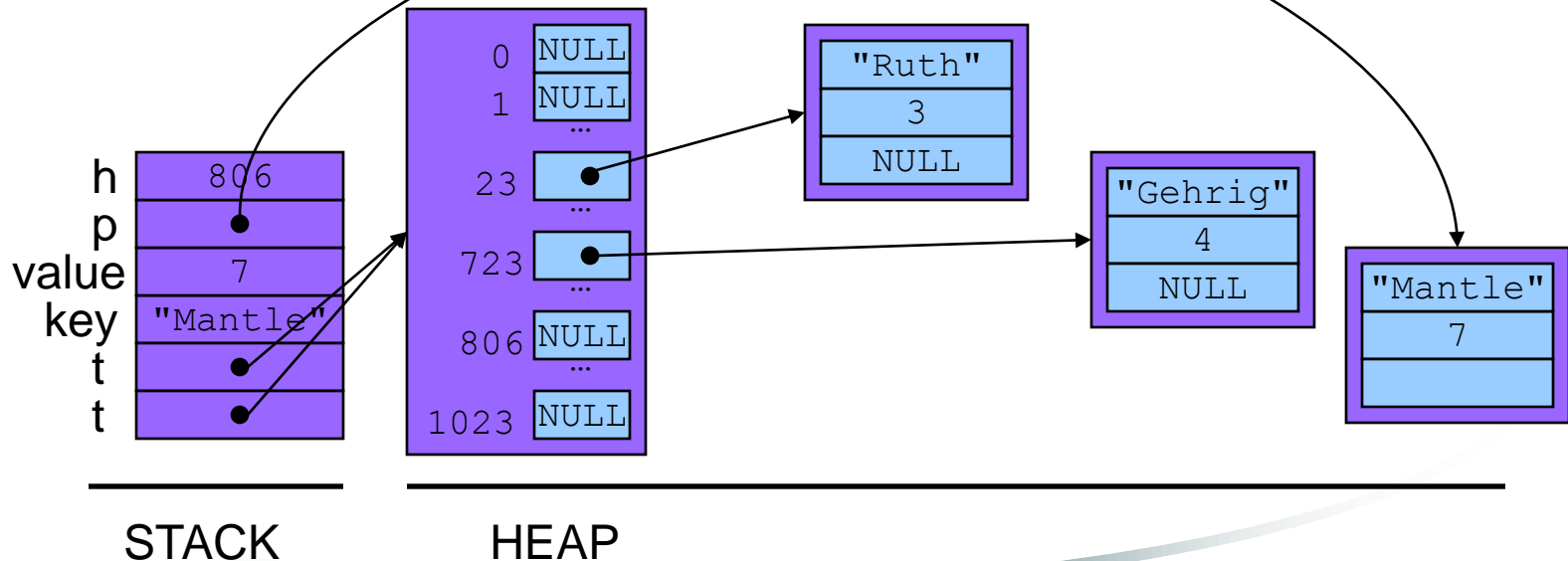
STACK

HEAP

Bảng băm: Add (3)

```
void Table_add(struct Table *t,  
    const char *key, int value) {  
    struct Node *p = (struct Node*)malloc(sizeof(struct Node));  
    int h = hash(key);  
    p->key = key;  
    p->value = value;  
    p->next = t->array[h];  
    t->array[h] = p;  
}
```

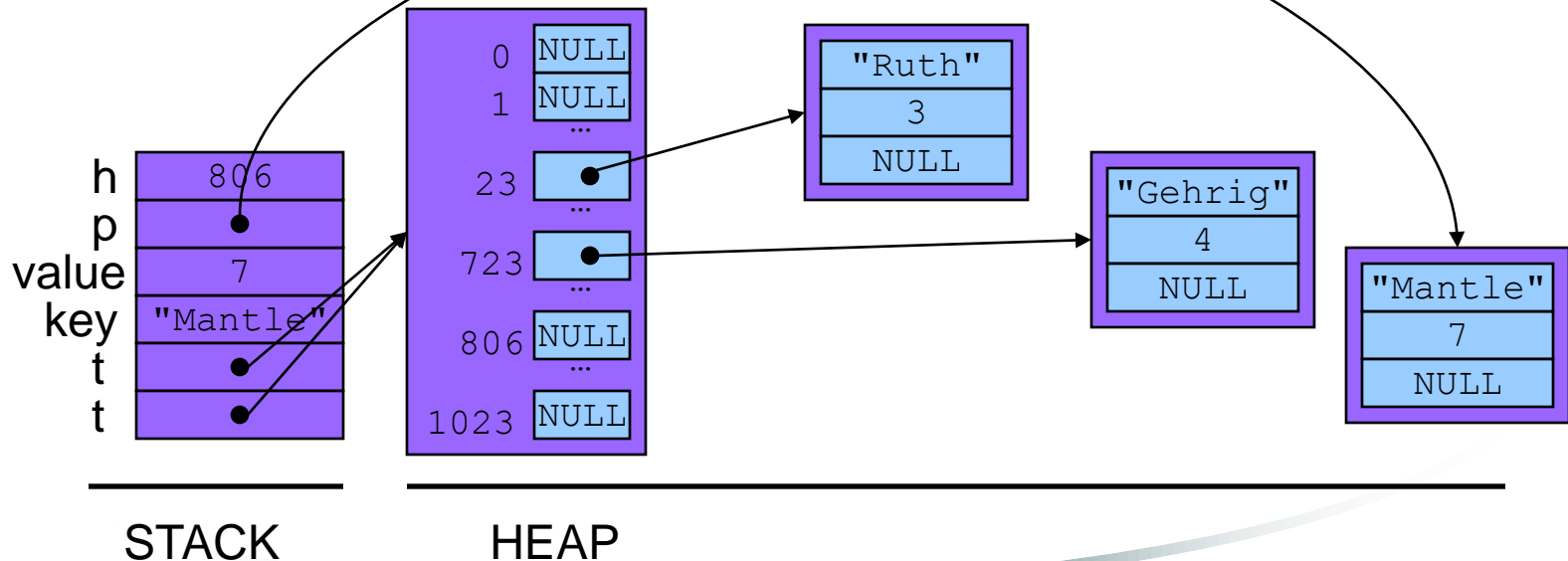
```
struct Table *t;  
...  
Table_add(t, "Ruth", 3);  
Table_add(t, "Gehrig", 4);  
Table_add(t, "Mantle", 7);  
...
```



Bảng băm: Add (4)

```
void Table_add(struct Table *t,  
    const char *key, int value) {  
    struct Node *p = (struct Node*)malloc(sizeof(struct Node));  
    int h = hash(key);  
    p->key = key;  
    p->value = value;  
    p->next = t->array[h];  
    t->array[h] = p;  
}
```

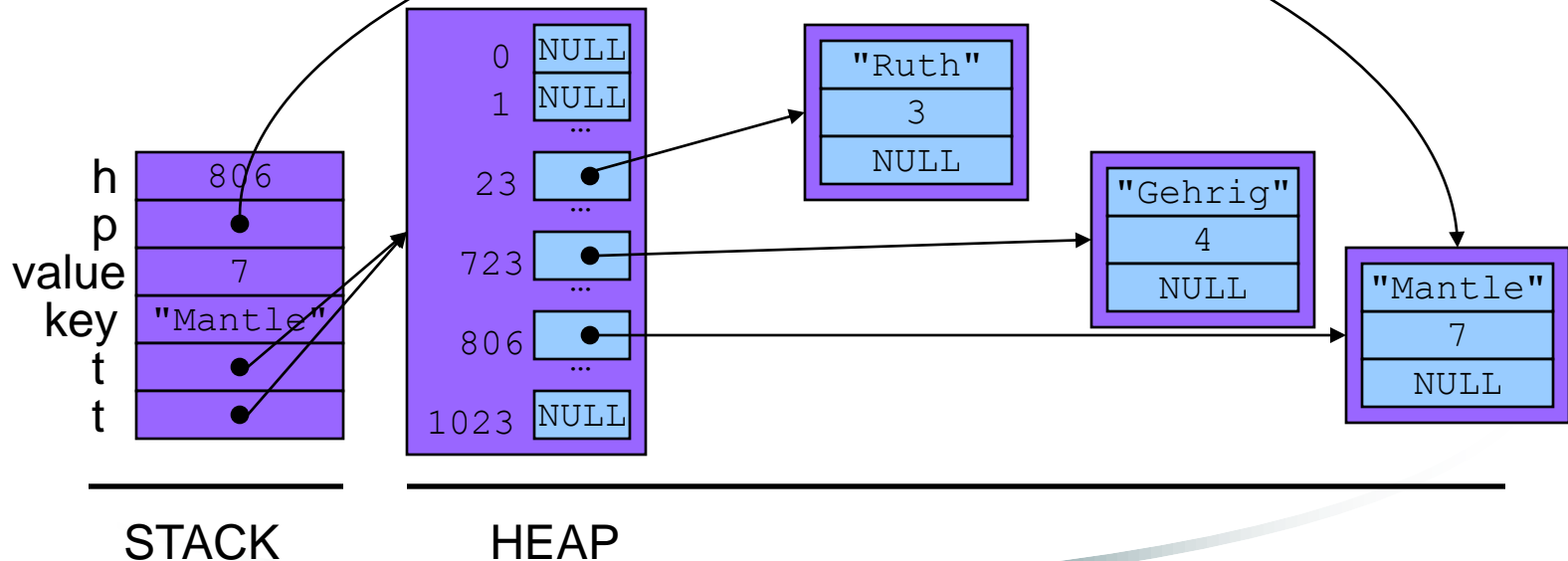
```
struct Table *t;  
...  
Table_add(t, "Ruth", 3);  
Table_add(t, "Gehrig", 4);  
Table_add(t, "Mantle", 7);  
...
```



Bảng băm: Add (5)

```
void Table_add(struct Table *t,  
    const char *key, int value) {  
    struct Node *p = (struct Node*)malloc(sizeof(struct Node));  
    int h = hash(key);  
    p->key = key;  
    p->value = value;  
    p->next = t->array[h];  
    t->array[h] = p;  
}
```

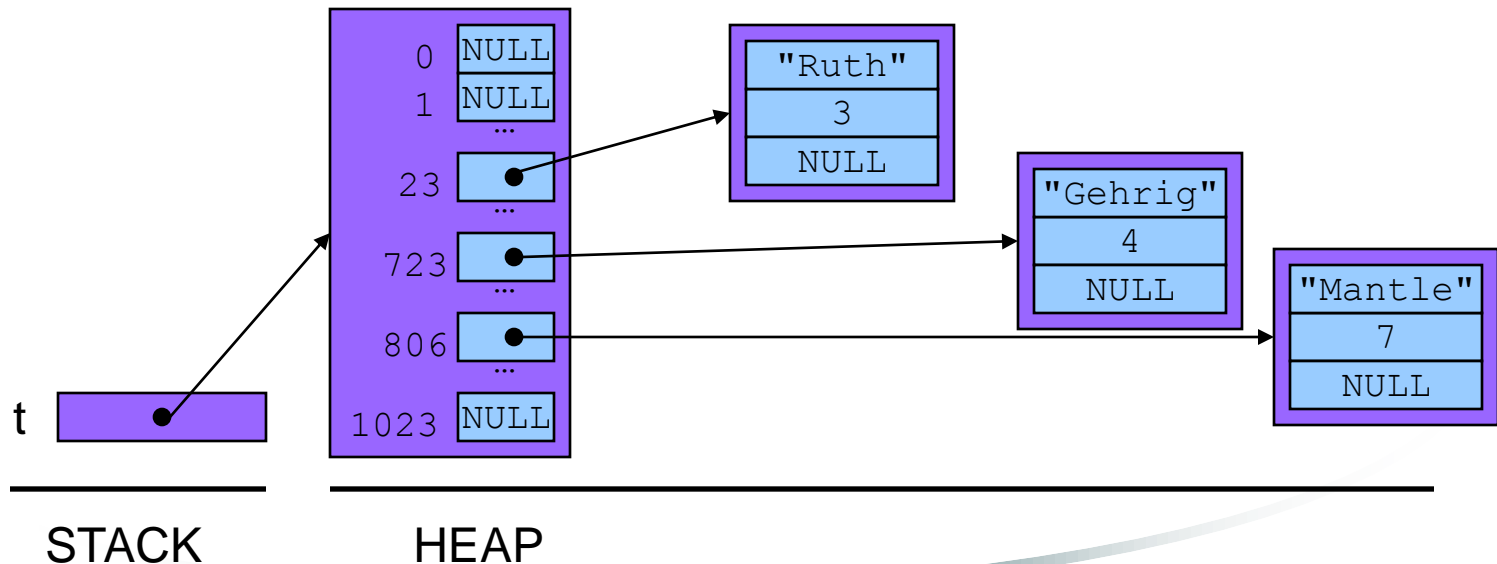
```
struct Table *t;  
...  
Table_add(t, "Ruth", 3);  
Table_add(t, "Gehrig", 4);  
Table_add(t, "Mantle", 7);  
...
```



Bảng băm: Add (6)

```
void Table_add(struct Table *t,  
    const char *key, int value) {  
    struct Node *p = (struct Node*)malloc(sizeof(struct Node));  
    int h = hash(key);  
    p->key = key;  
    p->value = value;  
    p->next = t->array[h];  
    t->array[h] = p;  
}
```

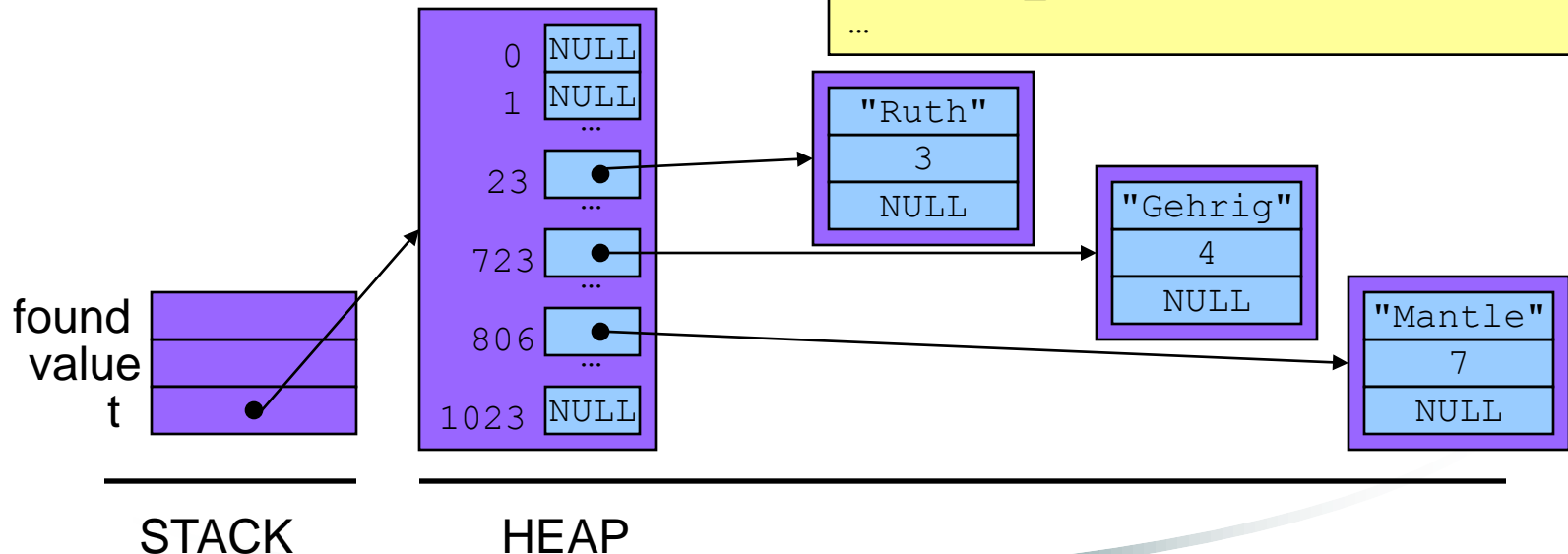
```
struct Table *t;  
...  
Table_add(t, "Ruth", 3);  
Table_add(t, "Gehrig", 4);  
Table_add(t, "Mantle", 7);  
...
```



Bảng băm: Search (1)

```
int Table_search(struct Table *t,
    const char *key, int *value) {
    struct Node *p;
    int h = hash(key);
    for (p = t->array[h]; p != NULL; p = p->next)
        if (strcmp(p->key, key) == 0) {
            *value = p->value;
            return 1;
        }
    return 0;
}
```

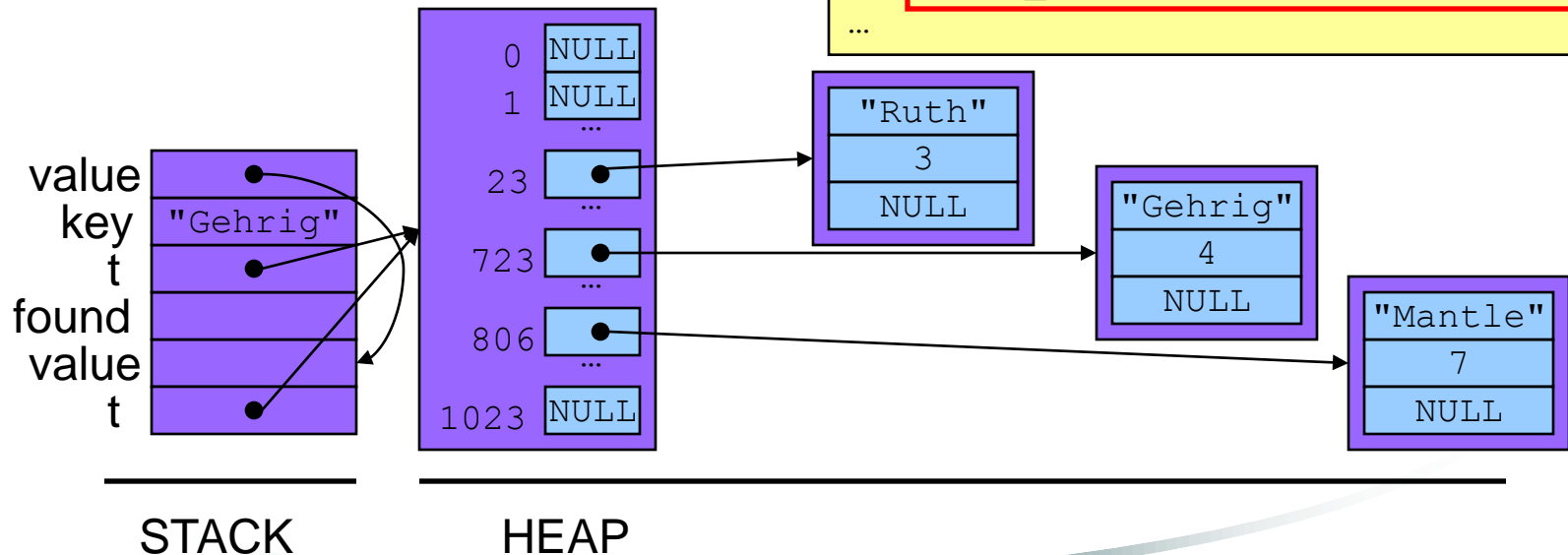
```
struct Table *t;
int value;
int found;
...
found =
    Table_search(t, "Gehrig", &value);
...
```



Bảng băm: Search (2)

```
int Table_search(struct Table *t,  
    const char *key, int *value) {  
    struct Node *p;  
    int h = hash(key);  
    for (p = t->array[h]; p != NULL; p = p->next)  
        if (strcmp(p->key, key) == 0) {  
            *value = p->value;  
            return 1;  
        }  
    return 0;  
}
```

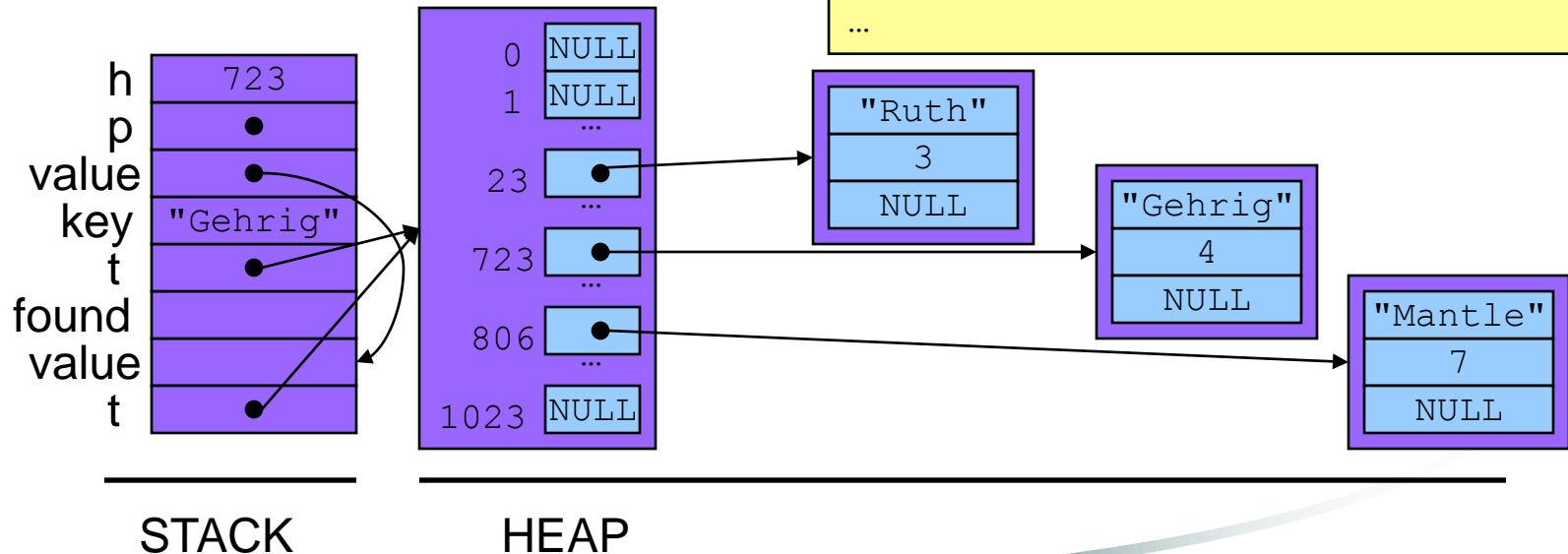
```
struct Table *t;  
int value;  
int found;  
...  
found =  
    Table_search(t, "Gehrig", &value);  
...
```



Bảng băm: Search (3)

```
int Table_search(struct Table *t,  
    const char *key, int *value) {  
    struct Node *p;  
    int h = hash(key);  
    for (p = t->array[h]; p != NULL; p = p->next)  
        if (strcmp(p->key, key) == 0) {  
            *value = p->value;  
            return 1;  
        }  
    return 0;  
}
```

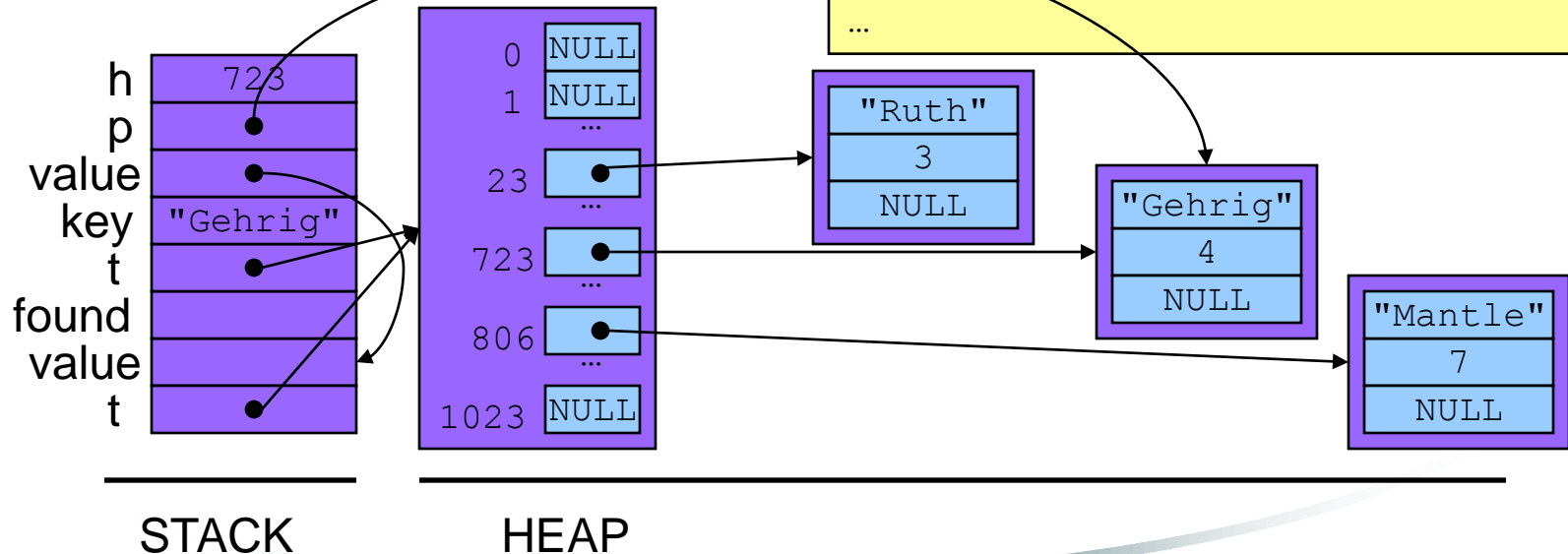
```
struct Table *t;  
int value;  
int found;  
...  
found =  
    Table_search(t, "Gehrig", &value);  
...
```



Bảng băm: Search (4)

```
int Table_search(struct Table *t,  
    const char *key, int *value) {  
    struct Node *p;  
    int h = hash(key);  
    for (p = t->array[h]; p != NULL; p = p->next)  
        if (strcmp(p->key, key) == 0) {  
            *value = p->value;  
            return 1;  
        }  
    return 0;  
}
```

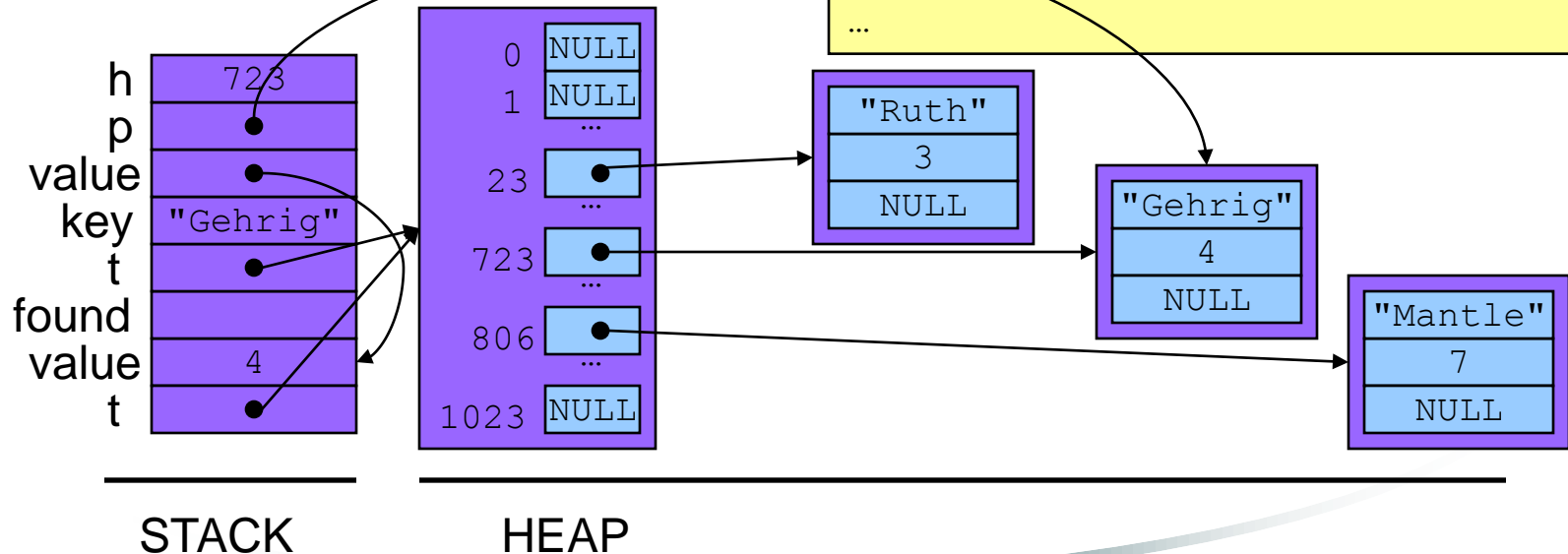
```
struct Table *t;  
int value;  
int found;  
...  
found =  
    Table_search(t, "Gehrig", &value);  
...
```



Bảng băm: Search (5)

```
int Table_search(struct Table *t,
    const char *key, int *value) {
    struct Node *p;
    int h = hash(key);
    for (p = t->array[h]; p != NULL; p = p->next)
        if (strcmp(p->key, key) == 0) {
            *value = p->value;
            return 1;
        }
    return 0;
}
```

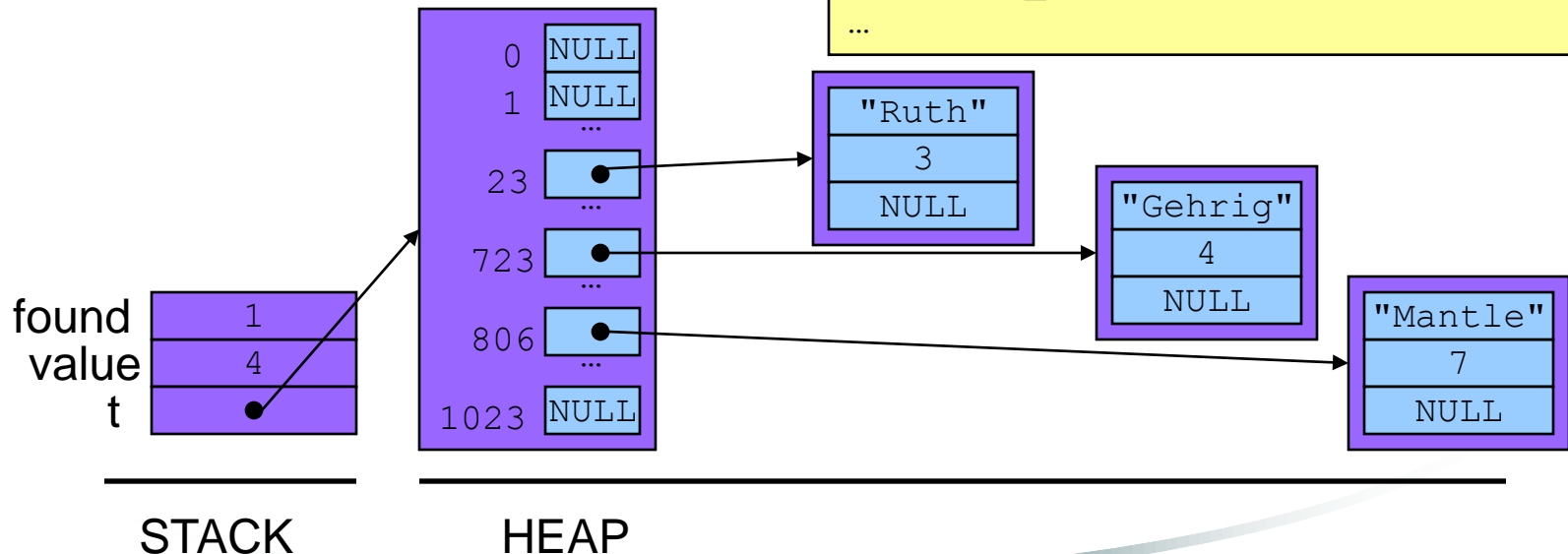
```
struct Table *t;
int value;
int found;
...
found =
    Table_search(t, "Gehrig", &value);
...
```



Bảng băm: Search (6)

```
int Table_search(struct Table *t,
    const char *key, int *value) {
    struct Node *p;
    int h = hash(key);
    for (p = t->array[h]; p != NULL; p = p->next)
        if (strcmp(p->key, key) == 0) {
            *value = p->value;
            return 1;
        }
    return 0;
}
```

```
struct Table *t;
int value;
int found;
...
found =
    Table_search(t, "Gehrig", &value);
...
```

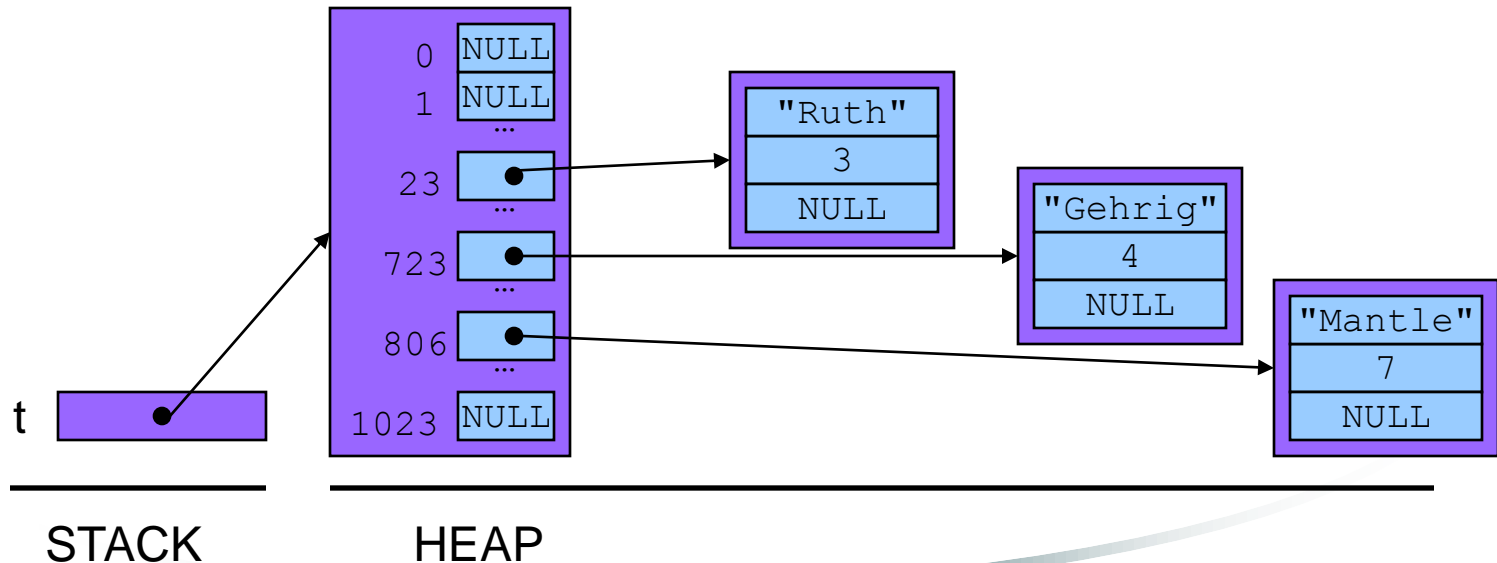


Bảng băm: Free (1)

```
void Table_free(struct Table *t) {  
    struct Node *p;  
    struct Node *nextp;  
    int b;  
    for (b = 0; b < BUCKET_COUNT; b++)  
        for (p = t->array[b]; p != NULL; p = nextp) {  
            nextp = p->next;  
            free(p);  
        }  
    free(t);  
}
```

```
struct Table *t;
```

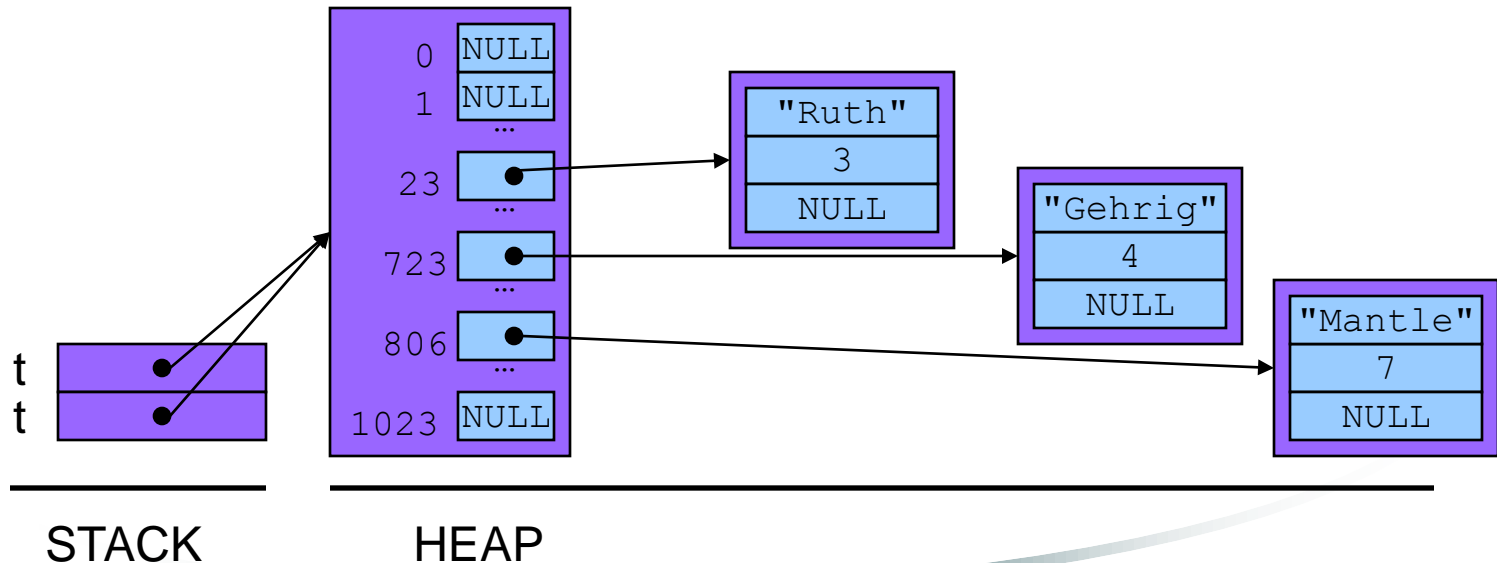
```
Table_free(t);
```



Bảng băm: Free (2)

```
void Table_free(struct Table *t) {  
    struct Node *p;  
    struct Node *nextp;  
    int b;  
    for (b = 0; b < BUCKET_COUNT; b++)  
        for (p = t->array[b]; p != NULL; p = nextp) {  
            nextp = p->next;  
            free(p);  
        }  
    free(t);  
}
```

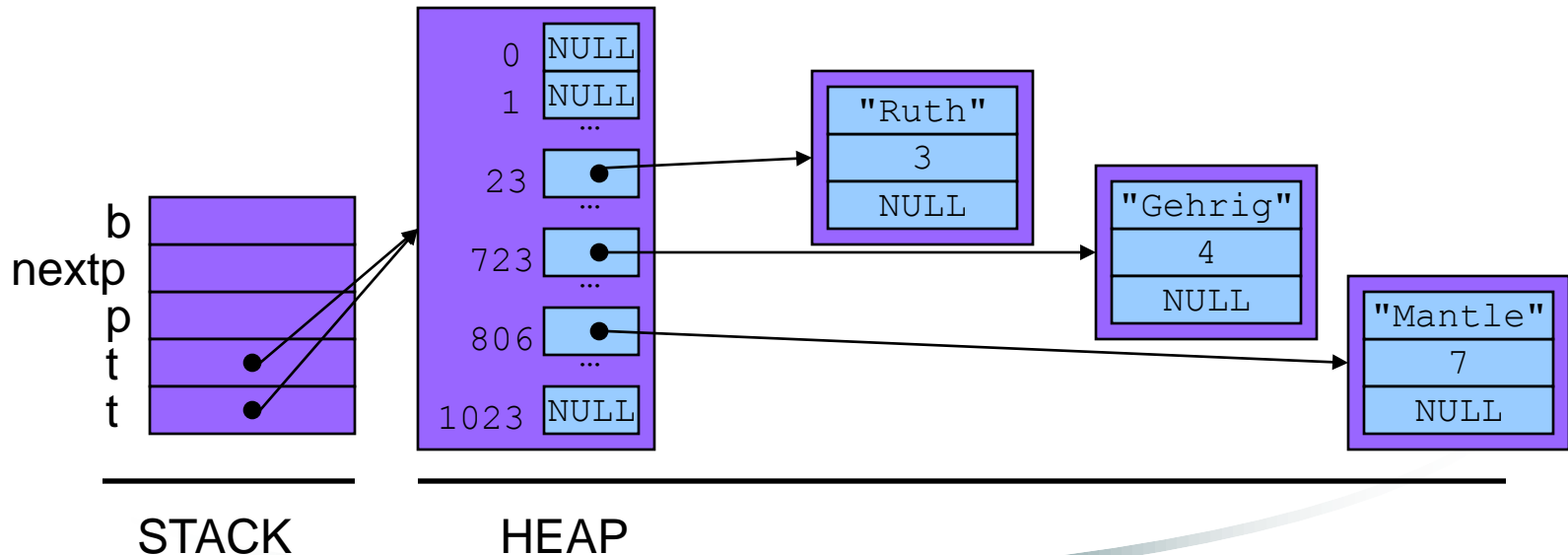
```
struct Table *t;  
...  
Table free(t);  
...
```



Bảng băm: Free (3)

```
void Table_free(struct Table *t) {  
    struct Node *p;  
    struct Node *nextp;  
    int b;  
    for (b = 0; b < BUCKET_COUNT; b++)  
        for (p = t->array[b]; p != NULL; p = nextp) {  
            nextp = p->next;  
            free(p);  
        }  
    free(t);  
}
```

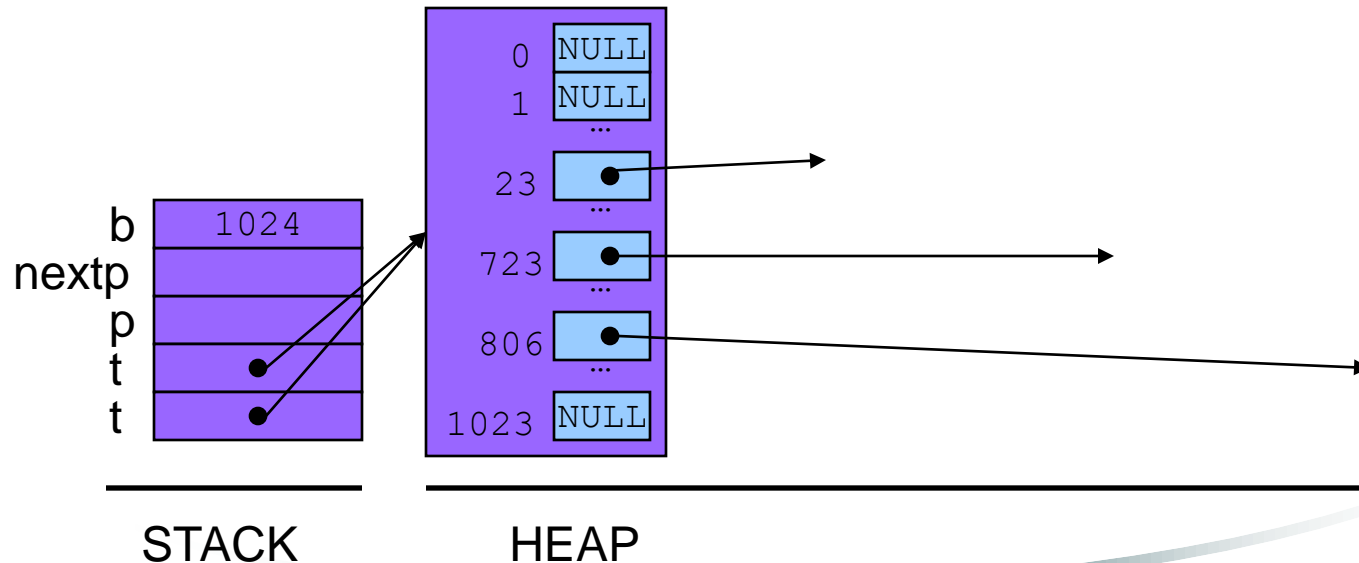
```
struct Table *t;  
...  
Table_free(t);  
...
```



Bảng băm: Free (4)

```
void Table_free(struct Table *t) {  
    struct Node *p;  
    struct Node *nextp;  
    int b;  
    for (b = 0; b < BUCKET_COUNT; b++)  
        for (p = t->array[b]; p != NULL; p = nextp) {  
            nextp = p->next;  
            free(p);  
        }  
    free(t);  
}
```

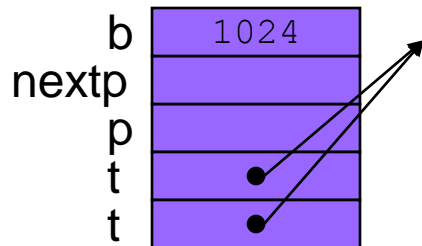
```
struct Table *t;  
...  
Table_free(t);  
...
```



Bảng băm: Free (5)

```
void Table_free(struct Table *t) {  
    struct Node *p;  
    struct Node *nextp;  
    int b;  
    for (b = 0; b < BUCKET_COUNT; b++)  
        for (p = t->array[b]; p != NULL; p = nextp) {  
            nextp = p->next;  
            free(p);  
        }  
    free(t);  
}
```

```
struct Table *t;  
...  
Table_free(t);  
...
```



STACK

HEAP

Bảng băm: Free (6)

```
void Table_free(struct Table *t) {  
    struct Node *p;  
    struct Node *nextp;  
    int b;  
    for (b = 0; b < BUCKET_COUNT; b++)  
        for (p = t->array[b]; p != NULL; p = nextp) {  
            nextp = p->next;  
            free(p);  
        }  
    free(t);  
}
```

```
struct Table *t;  
...  
Table_free(t);  
...
```





Bảng băm: hiệu năng

- Create: $O(1)$, nhanh
- Add: $O(1)$, nhanh
- Search: $O(1)$, nhanh trong trường hợp kích thước nhỏ
- Free: $O(n)$, chậm



Key Ownership

- Lưu ý: Hàm Table_add()

```
void Table_add(struct Table *t, const char *key, int value) {  
    ...  
    struct Node *p = (struct Node*)malloc(sizeof(struct Node));  
    p->key = key;  
    ...  
}
```

- Key là tham số đầu vào (con trỏ trỏ đến ô nhớ chứa xâu)
- Table_add() chỉ lưu trong bảng địa chỉ của xâu đó

Key Ownership (cont.)

- Problem: Xét đoạn mã sau

```
struct Table t;  
char k[100] = "Ruth";  
...  
Table_add(t, k, 3);  
strcpy(k, "Gehrig");  
...
```

Sau khi gọi Table_add(), bảng chứa địa chỉ ô nhớ k
LTV thay đổi xâu tại địa chỉ ô nhớ k
Rồi LTV thay đổi khóa trong bảng

- Vấn đề trong bảng băm
 - Khóa của nút hiện tại đổi từ "Ruth" thành "Gehrig"
 - Nút hiện có đặt sai chỗ
 - Bảng băm bị hỏng!!!
- Vấn đề này có thể xảy ra đối với các cấu trúc dữ liệu khác

Key Ownership (cont.)

- ```
void Table_add(struct Table *t, const char *key, int value) {
 ...
 struct Node *p = (struct Node*)malloc(sizeof(struct Node));
 p->key = (const char*)malloc(strlen(key) + 1);
 strcpy(p->key, key);
 ...
}
```

Cho phép lưu  
phần tử cuối

- Nếu LTV thay đổi xâu tại địa chỉ ô nhớ k, cấu trúc dữ liệu không bị ảnh hưởng
- Trong trường hợp này, cấu trúc dữ liệu quản lý bản sao của khóa, tức là:
  - CTDL chịu trách nhiệm giải phóng ô nhớ chứa bản sao
  - Hàm Table\_free() phải giải phóng bản sao



# Tóm tắt

- Các cấu trúc dữ liệu phổ biến và các giải thuật liên quan
  - **Linked list**
    - Unsorted => thêm mới nhanh, tìm kiếm chậm
    - Sorted => thêm mới chậm, tìm kiếm chậm
  - **Hash table**
    - Thêm mới nhanh, tìm kiếm nhanh nếu hàm băm hoạt động tốt
    - Khó đánh giá việc lưu trữ các cặp khóa/giá trị
    - Rất phổ biến
    - Các chủ đề liên quan:
      - Giải thuật băm
      - Quản lý bộ nhớ
- Khác
  - #1: tiểu xảo tạo ra các hàm băm nhanh hơn
  - #2: ví dụ về cấu trúc dữ liệu khác



# #1: Xem lại các hàm băm

- Tính “mod c” có vẻ mất nhiều thời gian
  - Bao gồm việc thực hiện phép chia với số chia là c và lưu trữ số dư
  - Sẽ dễ hơn nếu c là lũy thừa của 2 (e.g.,  $16 = 2^4$ )
- Đề xuất giải pháp thay thế:
  - $53 = 32 + 16 + 4 + 1$

|     |    |    |   |   |   |   |   |
|-----|----|----|---|---|---|---|---|
| ... | 32 | 16 | 8 | 4 | 2 | 1 |   |
| 0   | 0  | 1  | 1 | 0 | 1 | 0 | 1 |

- $53 \% 16 = 5$ , chính là giá trị chứa trong 4 bits cuối

|     |    |    |   |   |   |   |   |
|-----|----|----|---|---|---|---|---|
| ... | 32 | 16 | 8 | 4 | 2 | 1 |   |
| 0   | 0  | 0  | 0 | 0 | 1 | 0 | 1 |

- Nếu có thể để đang tác ra 4 bits này...

# Recall: Bitwise Operators in C

- Bitwise AND (&)

| & | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

- Bitwise OR (|)

|   | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

– Thực hiện phép chia lấy dư, nhưng ít tốn kém hơn

- E.g.,  $h = 53 \& 15$ ;

53    

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

& 15    

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|

---

5    

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

Cách khác

Đảo 0 thành 1, 1 thành 0

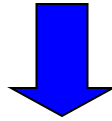
E.g., đặt 3 bits cuối thành 0

$x = x \& \sim 7$ ;

# Hàm băm nhanh hơn

```
unsigned int hash(const char *x) {
 int i;
 unsigned int h = 0U;
 for (i=0; x[i]!='\0'; i++)
 h = h * 65599 + (unsigned char)x[i];
 return h % 1024;
}
```

Previous  
version



```
unsigned int hash(const char *x) {
 int i;
 unsigned int h = 0U;
 for (i=0; x[i]!='\0'; i++)
 h = h * 65599 + (unsigned char)x[i];
 return h & 1023;
}
```

Nhanh hơn

- Chú ý: không viết "h & 1024"

# Tăng tốc việc so sánh các khóa

- Cho mọi hàm so sánh giá trị không tầm thường
- Trick: lưu trữ toàn bộ giá trị băm trong CTDL

```
int Table_search(struct Table *t,
 const char *key, int *value) {
 struct Node *p;
 int h = hash(key); /* No % in hash function */
 for (p = t->array[h%1024]; p != NULL; p = p->next)
 if ((p->hash == h) && strcmp(p->key, key) == 0) {
 *value = p->value;
 return 1;
 }
 return 0;
}
```



## # 2: Cấu trúc dữ liệu khác

- Expanding array...
- 



# Expanding Array

- Ý tưởng
- Cấu trúc dữ liệu: mảng có thể mở rộng kích thước khi cần
- **Giải thuật Create:** cấp phát mảng lưu trữ các cặp khóa/giá trị, ban đầu có rất ít phần tử
- **Giải thuật Add:** Nếu hết chỗ để lưu trữ thêm phần tử mới, thì tăng gấp đôi kích thước mảng và đặt cặp khóa/giá trị cần thêm vào vào trong phần tử đầu tiên chưa sử dụng
  - Để tăng tính hiệu quả, không nên mở rộng mảng theo kiểu tuyến tính
- **Giải thuật Search:** tìm kiếm tuyến tính
- **Giải thuật Free:** giải phóng mảng

# Expanding Array: Data Structure

```
enum {INITIAL_SIZE = 2};
enum {GROWTH_FACTOR = 2};

struct Pair {
 const char *key;
 int value;
};

struct Table {
 int pairCount; /* Number of pairs in table */
 int arraySize; /* Physical size of array */
 struct Pair *array; /* Address of array */
};
```

# Expanding Array: Create (1)

```
struct Table *Table_create(void) {
 struct Table *t;
 t = (struct Table*)
 malloc(sizeof(struct Table));
 t->pairCount = 0;
 t->arraySize = INITIAL_SIZE;
 t->array = (struct Pair*)
 calloc(INITIAL_SIZE,
 sizeof(struct Pair));
 return t;
}
```

```
{
 struct Table *t;
 ...
 t = Table_create();
 ...
}
```

t 

STACK

HEAP



# Expanding Array: Create (2)

```
struct Table *Table_create(void) {
 struct Table *t;
 t = (struct Table*)
 malloc(sizeof(struct Table));
 t->pairCount = 0;
 t->arraySize = INITIAL_SIZE;
 t->array = (struct Pair*)
 calloc(INITIAL_SIZE,
 sizeof(struct Pair));
 return t;
}
```

```
{
 struct Table *t;
 ...
 t = Table_create();
 ...
}
```



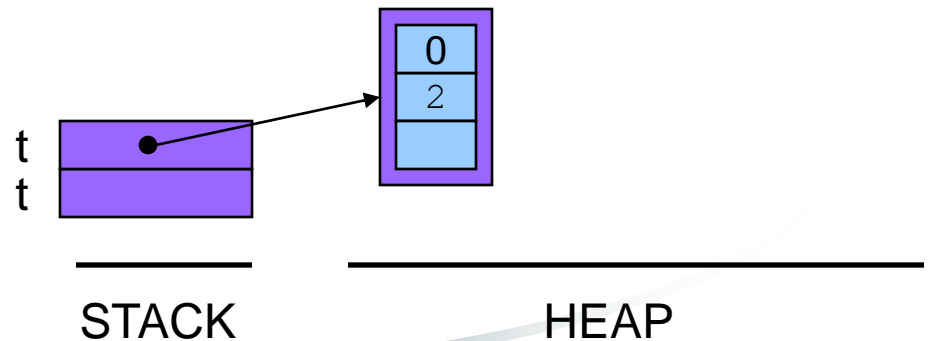
STACK

HEAP

# Expanding Array: Create (3)

```
struct Table *Table_create(void) {
 struct Table *t;
 t = (struct Table*)
 malloc(sizeof(struct Table));
 t->pairCount = 0;
 t->arraySize = INITIAL_SIZE;
 t->array = (struct Pair*)
 calloc(INITIAL_SIZE,
 sizeof(struct Pair));
 return t;
}
```

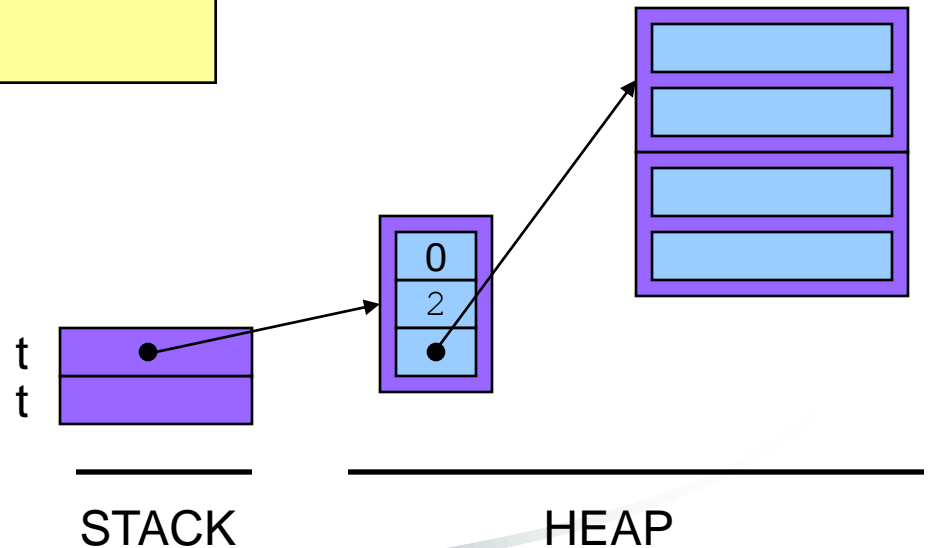
```
{
 struct Table *t;
 ...
 t = Table_create();
 ...
}
```



# Expanding Array: Create (4)

```
struct Table *Table_create(void) {
 struct Table *t;
 t = (struct Table*)
 malloc(sizeof(struct Table));
 t->pairCount = 0;
 t->arraySize = INITIAL_SIZE;
 t->array = (struct Pair*)
 calloc(INITIAL_SIZE,
 sizeof(struct Pair));
 return t;
}
```

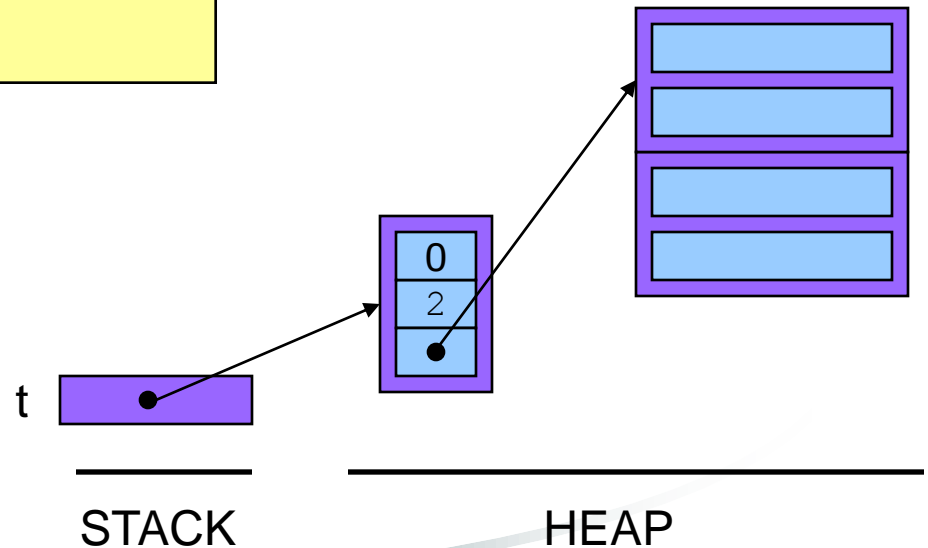
```
{
 struct Table *t;
 ...
 t = Table_create();
 ...
}
```



# Expanding Array: Create (5)

```
struct Table *Table_create(void) {
 struct Table *t;
 t = (struct Table*)
 malloc(sizeof(struct Table));
 t->pairCount = 0;
 t->arraySize = INITIAL_SIZE;
 t->array = (struct Pair*)
 calloc(INITIAL_SIZE,
 sizeof(struct Pair));
 return t;
}
```

```
{
 struct Table *t;
 ...
 t = Table_create();
 ...
}
```

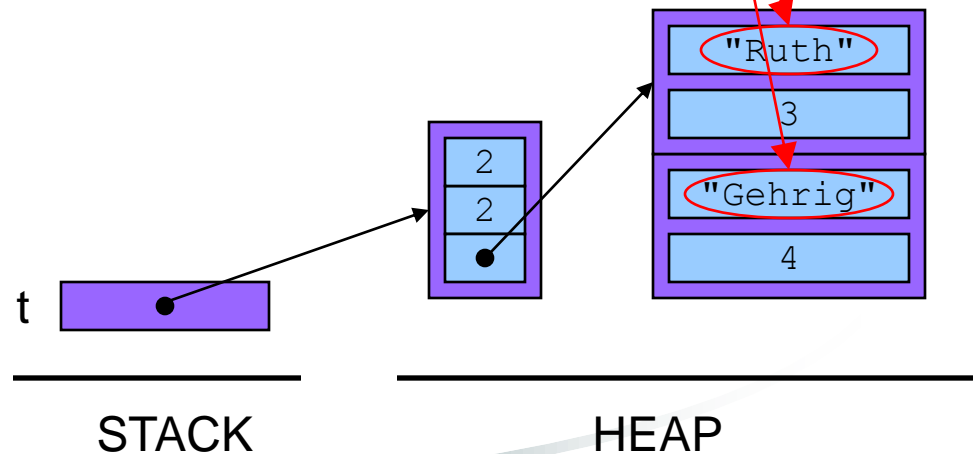


# Expanding Array: Add (1)

```
void Table_add(struct Table *t,
 const char *key, int value) {
 /* Expand if necessary. */
 if (t->pairCount == t->arraySize) {
 t->arraySize *= GROWTH_FACTOR;
 t->array = (struct Pair*)realloc(t->array,
 t->arraySize * sizeof(struct Pair));
 }
 t->array[t->pairCount].key = key;
 t->array[t->pairCount].value = value;
 t->pairCount++;
}
```

These are pointers to strings that exist in the RODATA section

```
struct Table *t;
...
Table_add(t, "Ruth", 3);
Table_add(t, "Gehrig", 4);
Table_add(t, "Mantle", 7);
...
```

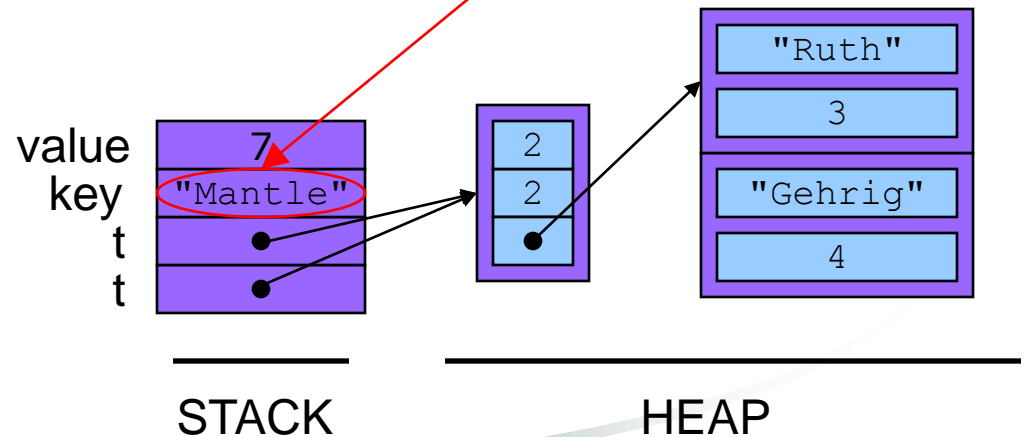


# Expanding Array: Add (2)

```
void Table_add(struct Table *t,
 const char *key, int value) {
 /* Expand if necessary. */
 if (t->pairCount == t->arraySize) {
 t->arraySize *= GROWTH_FACTOR;
 t->array = (struct Pair*)realloc(t->array,
 t->arraySize * sizeof(struct Pair));
 }
 t->array[t->pairCount].key = key;
 t->array[t->pairCount].value = value;
 t->pairCount++;
}
```

This is a pointer to a string that exists in the RODATA section

```
struct Table *t;
...
Table_add(t, "Ruth", 3);
Table_add(t, "Gehrig", 4);
Table_add(t, "Mantle", 7);
...
```

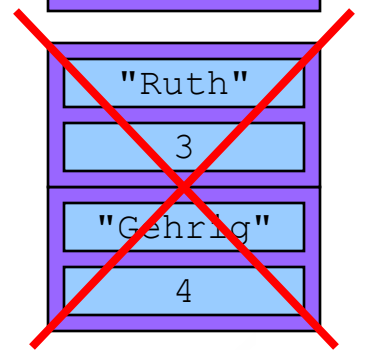
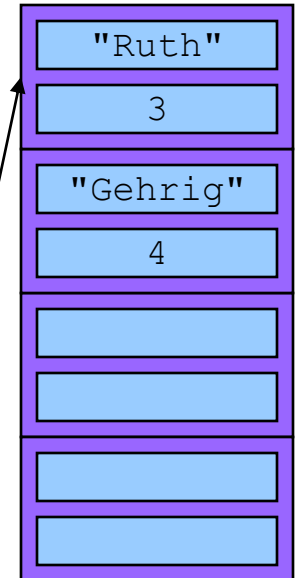
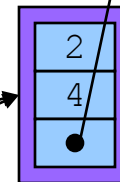
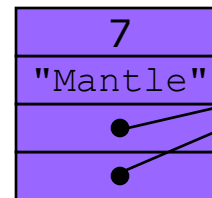


# Expanding Array: Add (3)

```
void Table_add(struct Table *t,
 const char *key, int value) {
 /* Expand if necessary. */
 if (t->pairCount == t->arraySize) {
 t->arraySize *= GROWTH_FACTOR;
 t->array = (struct Pair*)realloc(t->array,
 t->arraySize * sizeof(struct Pair));
 }
 t->array[t->pairCount].key = key;
 t->array[t->pairCount].value = value;
 t->pairCount++;
}
```

```
struct Table *t;
...
Table_add(t, "Ruth", 3);
Table_add(t, "Gehrig", 4);
Table_add(t, "Mantle", 7);
...
```

value  
key  
t  
t



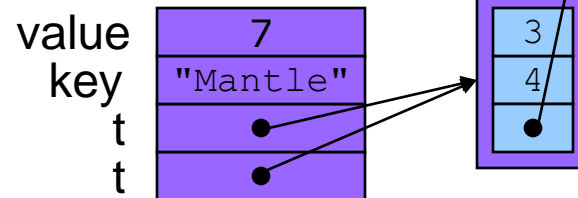
STACK

HEAP

# Expanding Array: Add (4)

```
void Table_add(struct Table *t,
 const char *key, int value) {
 /* Expand if necessary. */
 if (t->pairCount == t->arraySize) {
 t->arraySize *= GROWTH_FACTOR;
 t->array = (struct Pair*)realloc(t->array,
 t->arraySize * sizeof(struct Pair));
 }
 t->array[t->pairCount].key = key;
 t->array[t->pairCount].value = value;
 t->pairCount++;
}
```

```
struct Table *t;
...
Table_add(t, "Ruth", 3);
Table_add(t, "Gehrig", 4);
Table_add(t, "Mantle", 7);
...
```



STACK

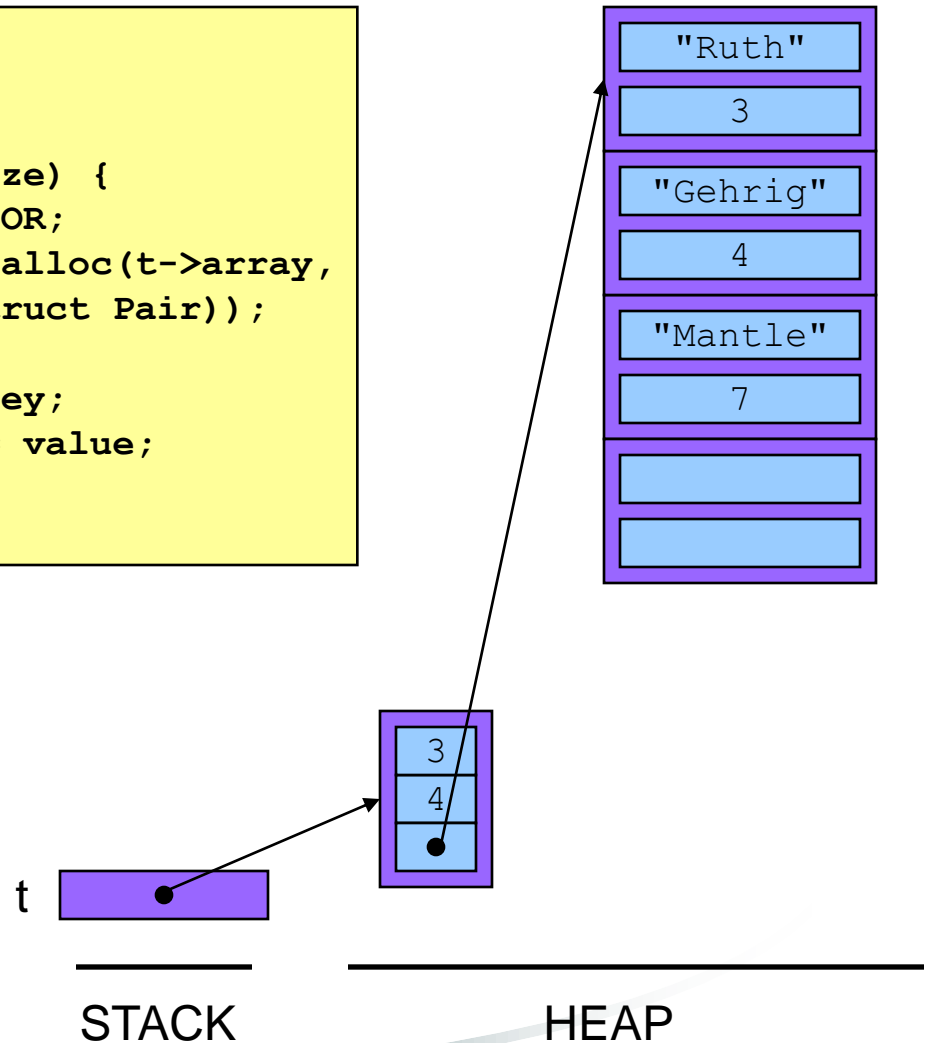
HEAP



# Expanding Array: Add (5)

```
void Table_add(struct Table *t,
 const char *key, int value) {
 /* Expand if necessary. */
 if (t->pairCount == t->arraySize) {
 t->arraySize *= GROWTH_FACTOR;
 t->array = (struct Pair*)realloc(t->array,
 t->arraySize * sizeof(struct Pair));
 }
 t->array[t->pairCount].key = key;
 t->array[t->pairCount].value = value;
 t->pairCount++;
}
```

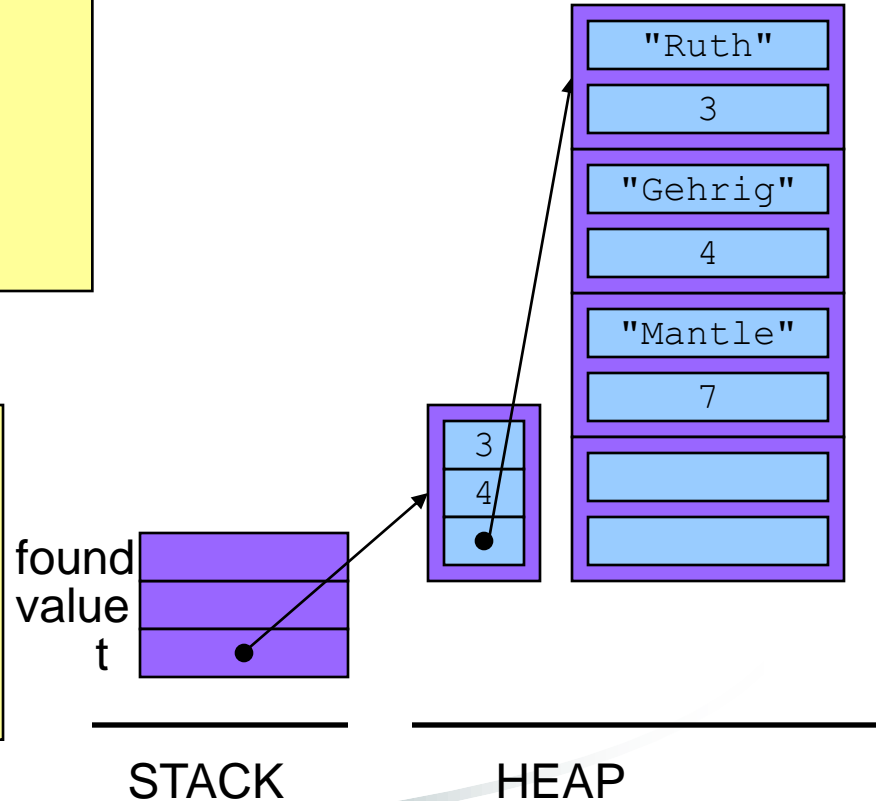
```
struct Table *t;
...
Table_add(t, "Ruth", 3);
Table_add(t, "Gehrig", 4);
Table_add(t, "Mantle", 7);
...
```



# Expanding Array: Search (1)

```
int Table_search(struct Table *t,
 const char *key, int *value) {
 int i;
 for (i = 0; i < t->pairCount; i++) {
 struct Pair p = t->array[i];
 if (strcmp(p.key, key) == 0) {
 *value = p.value;
 return 1;
 }
 }
 return 0;
}
```

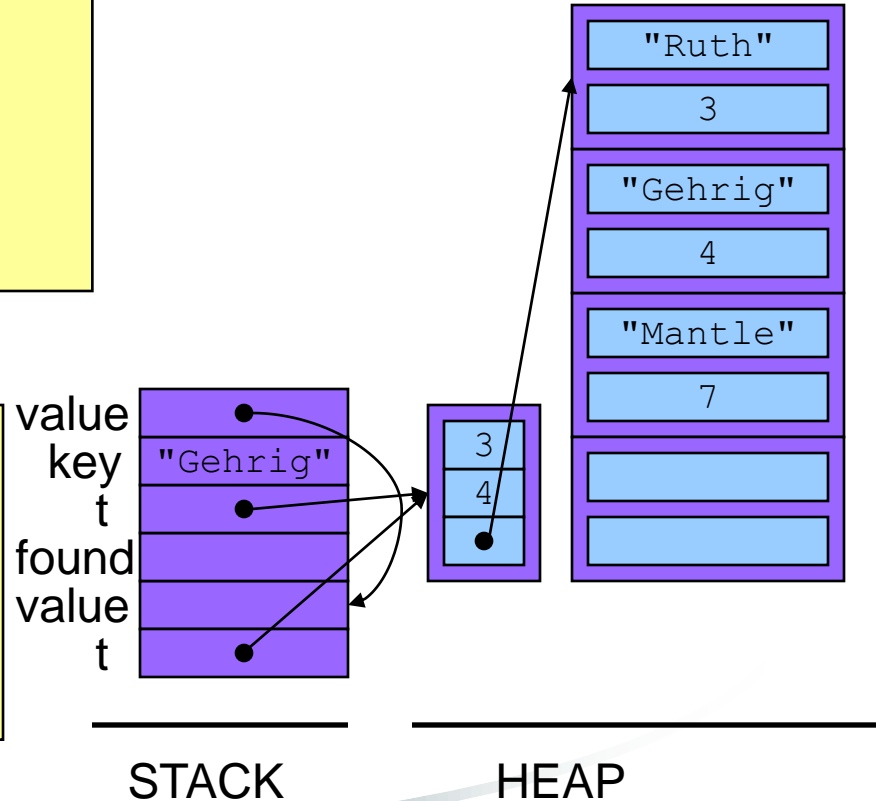
```
struct Table *t;
int value;
int found;
...
found =
 Table_search(t, "Gehrig", &value);
...
```



# Expanding Array: Search (2)

```
int Table_search(struct Table *t,
 const char *key, int *value) {
 int i;
 for (i = 0; i < t->pairCount; i++) {
 struct Pair p = t->array[i];
 if (strcmp(p.key, key) == 0) {
 *value = p.value;
 return 1;
 }
 }
 return 0;
}
```

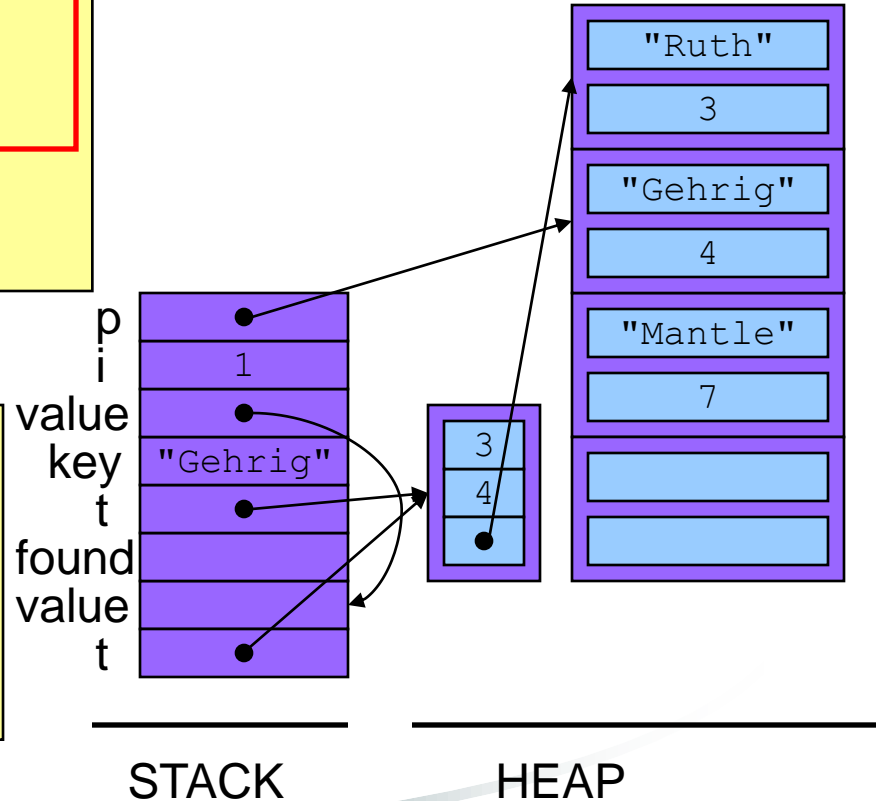
```
struct Table *t;
int value;
int found;
...
found =
 Table_search(t, "Gehrig", &value);
...
```



# Expanding Array: Search (3)

```
int Table_search(struct Table *t,
 const char *key, int *value) {
 int i;
 for (i = 0; i < t->pairCount; i++) {
 struct Pair p = t->array[i];
 if (strcmp(p.key, key) == 0) {
 *value = p.value;
 return 1;
 }
 }
 return 0;
}
```

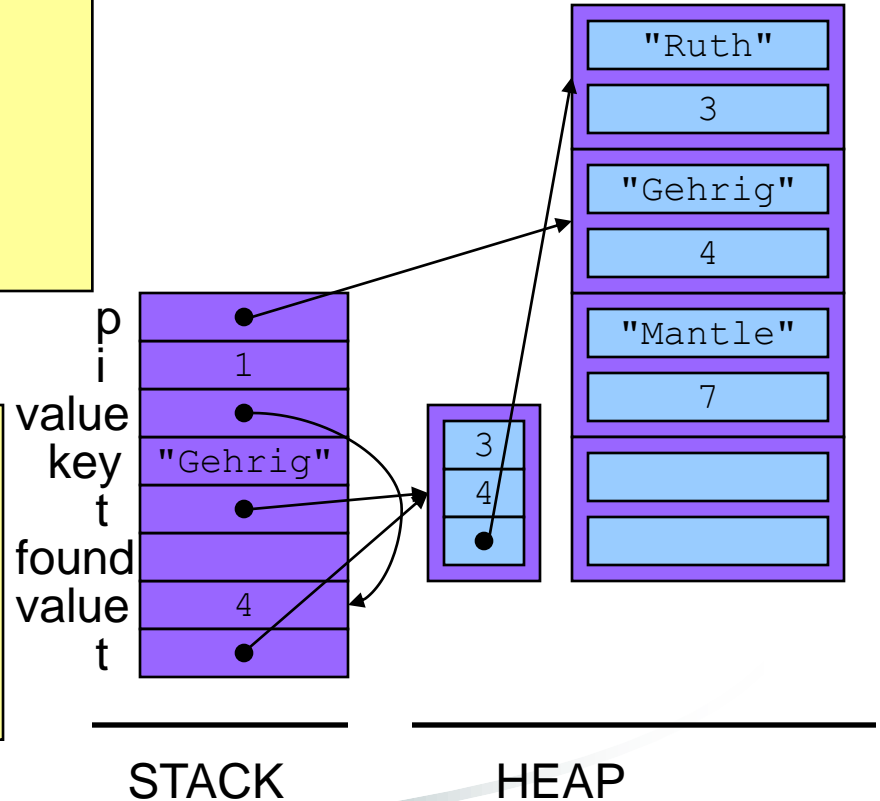
```
struct Table *t;
int value;
int found;
...
found =
 Table_search(t, "Gehrig", &value);
...
```



# Expanding Array: Search (4)

```
int Table_search(struct Table *t,
 const char *key, int *value) {
 int i;
 for (i = 0; i < t->pairCount; i++) {
 struct Pair p = t->array[i];
 if (strcmp(p.key, key) == 0) {
 *value = p.value;
 return 1;
 }
 }
 return 0;
}
```

```
struct Table *t;
int value;
int found;
...
found =
 Table_search(t, "Gehrig", &value);
...
```

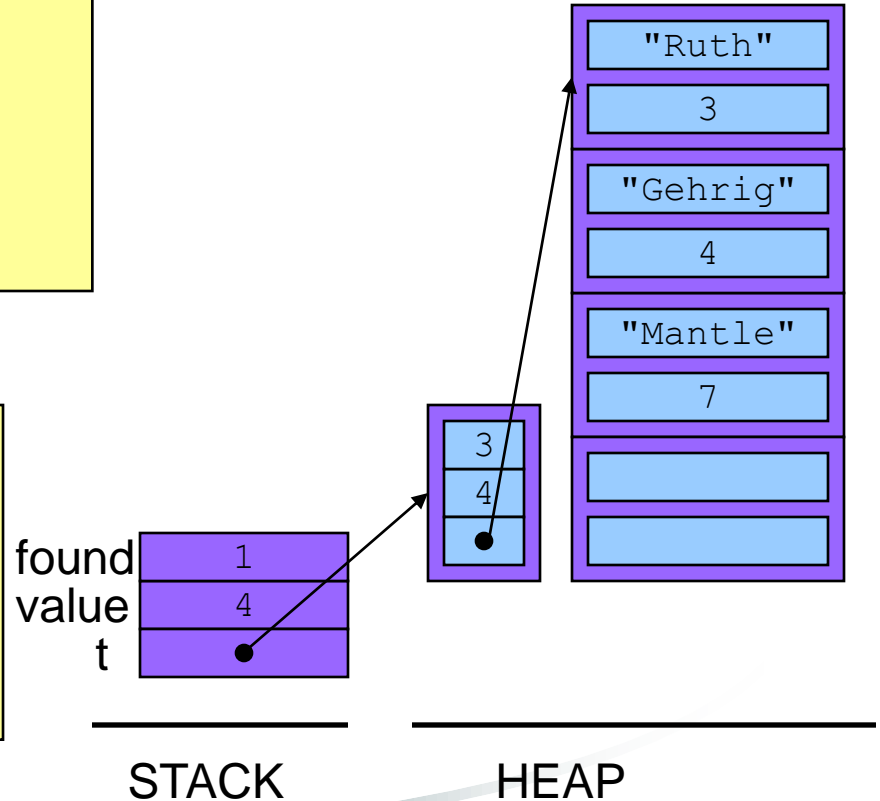


# Expanding Array: Search (5)

```
int Table_search(struct Table *t,
 const char *key, int *value) {
 int i;
 for (i = 0; i < t->pairCount; i++) {
 struct Pair p = t->array[i];
 if (strcmp(p.key, key) == 0) {
 *value = p.value;
 return 1;
 }
 }
 return 0;
}
```

```
struct Table *t;
int value;
int found;

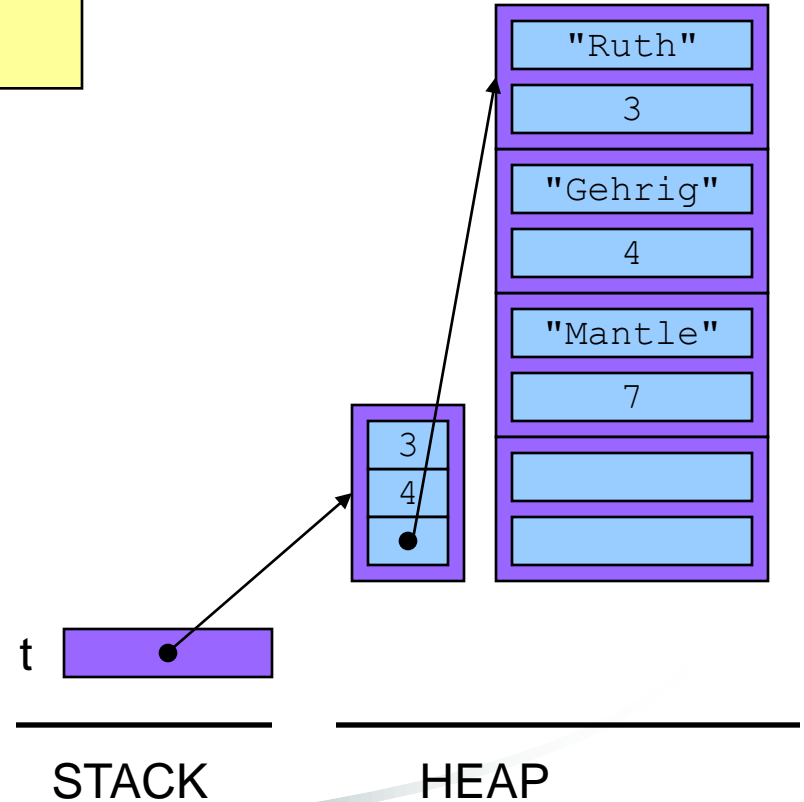
found =
 Table_search(t, "Gehrig", &value);
...
```



# Expanding Array: Free (1)

```
void Table_free(struct Table *t) {
 free(t->array);
 free(t);
}
```

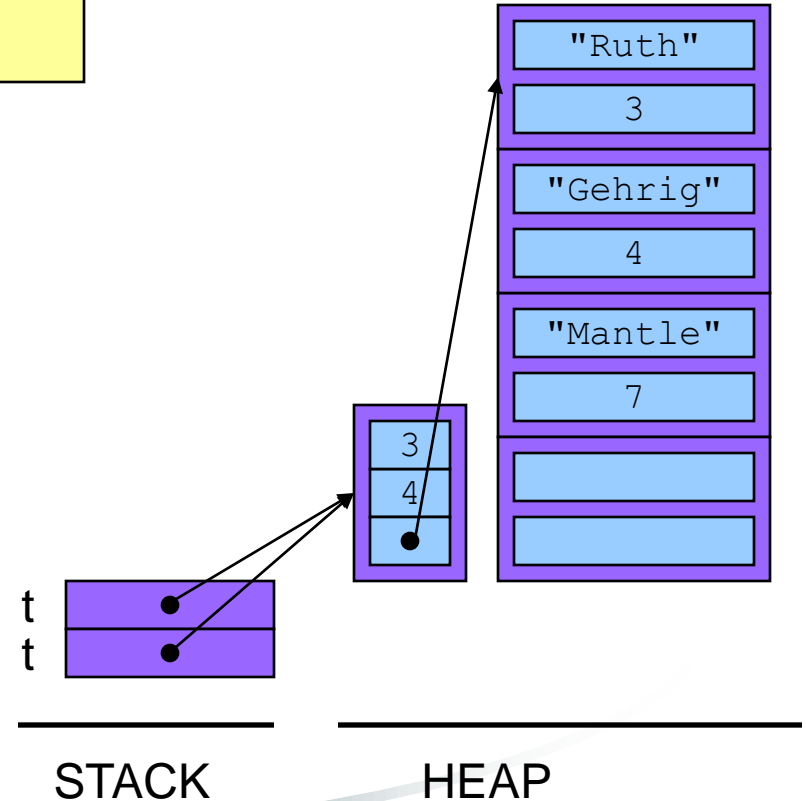
```
struct Table *t;
...
Table_free(t);
...
```



# Expanding Array: Free (2)

```
void Table_free(struct Table *t) {
 free(t->array);
 free(t);
}
```

```
struct Table *t;
...
Table_free(t);
...
```

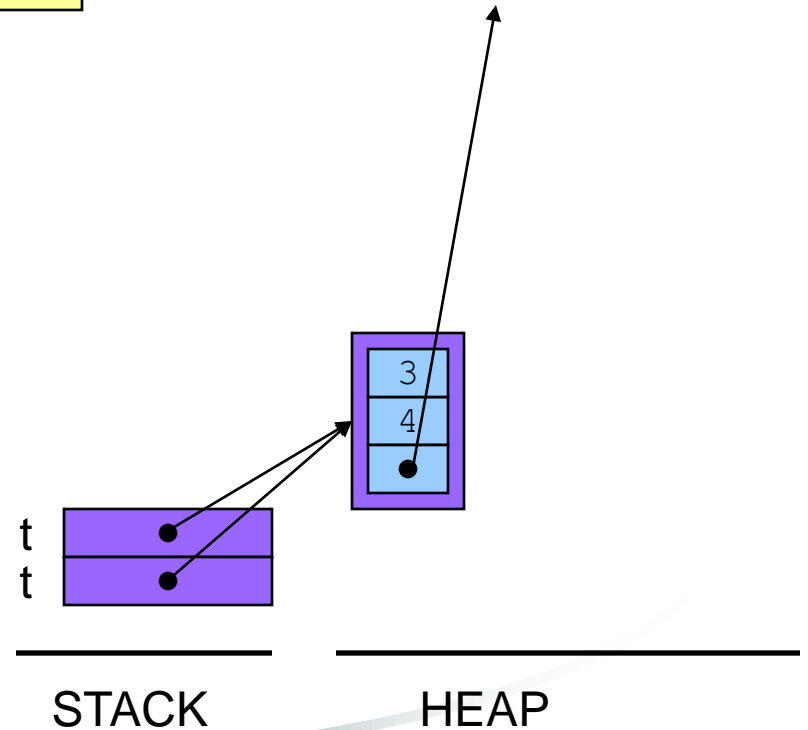




# Expanding Array: Free (3)

```
void Table_free(struct Table *t) {
 free(t->array);
 free(t);
}
```

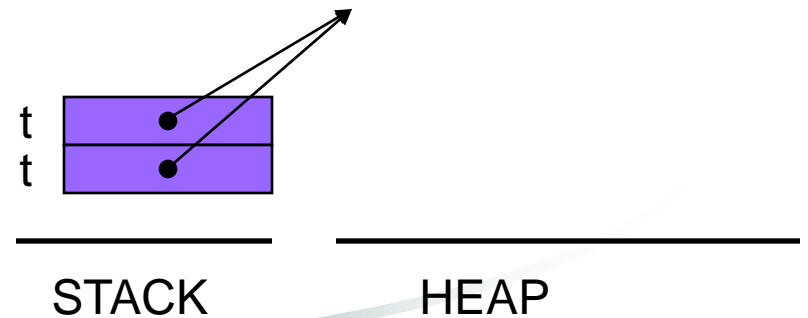
```
struct Table *t;
...
Table_free(t);
...
```



# Expanding Array: Free (4)

```
void Table_free(struct Table *t) {
 free(t->array);
 free(t);
}
```

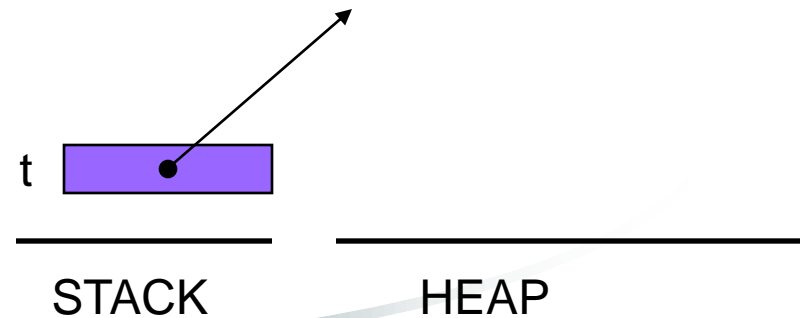
```
struct Table *t;
...
Table_free(t);
...
```



# Expanding Array: Free (5)

```
void Table_free(struct Table *t) {
 free(t->array);
 free(t);
}
```

```
struct Table *t;
...
Table_free(t);
...
```





# Expanding Array: hiệu năng

- Thời gian tính toán
  - Create:  $O(1)$ , nhanh
  - Add:  $O(1)$ , nhanh
  - Search:  $O(n)$ , chậm
  - Free:  $O(1)$ , nhanh
- Giải pháp khác: Sắp xếp mảng theo khóa
  - Create:  $O(1)$ , nhanh
  - Add:  $O(n)$ , chậm; cần phải dịch 1 cặp khóa/ giá trị để có chỗ thêm mới
  - Search:  $O(\log n)$ , chấp nhận được, có thể sử dụng tìm kiếm nhị phân
  - Free:  $O(1)$ , nhanh

## 5. Quản lý bộ nhớ động

Chương 3. Các kỹ thuật xây dựng chương trình phần mềm

**IV. Thiết kế chương trình và các kỹ thuật xây dựng hàm/thủ tục**

- Một số hạn chế khi sử dụng mảng trong C:
    - Kích thước của mảng cần biết trước
    - Kích thước của mảng không thể thay đổi trong chương trình
  - Cần:
    - Cấp phát bộ nhớ đúng như lượng cần khi chạy
    - Giải phóng bộ nhớ ngay khi không cần dùng
    - Chương trình không cần phải viết lại để xử lý lượng dữ liệu lớn hơn
- Quản lý bộ nhớ động (dynamic memory)
- C: Dùng malloc(), calloc(), realloc() và free()
  - C++: Dùng new và delete

## 5.1. Quản lý bộ nhớ động trong C

### ✿ Trong thư viện *stdlib.h*

✿ `malloc(size_t n), calloc(size_t m, size_t n)`

- Cấp phát một vùng nhớ mới.

✿ `realloc(void * ptr, size_t n)`

- Thay đổi kích thước cho một vùng nhớ đã được cấp phát

✿ `free(void * ptr)`

- Giải phóng cho vùng nhớ đã được cấp phát

✿ `size_t` là kiểu dữ liệu chứa kích thước đối tượng (số byte)

✿ `size_t sizeof(doi_tuong)`: Kích thước của `doi_tuong`

✿ Ví dụ: `sizeof(int) → 2`

## 5.1.1. Cấp phát vùng nhớ

✿ **`void *malloc(size_t n) ;`**

- ✿ Lấy về một vùng nhớ liên tục có kích thước ít nhất là n byte
- ✿ Chỉ xin cấp phát vùng nhớ chứ nội dung vùng nhớ vẫn chưa được xác định

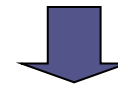
✿ **`void *calloc(size_t m, size_t n) ;`**

- ✿ Lấy về một vùng nhớ có kích thước ít nhất là m x n byte.
- ✿ Vùng nhớ này đủ để chứa một mảng gồm m phần tử, mỗi phần tử chiếm n byte.
- ✿ Mỗi byte nhớ được khởi tạo với giá trị 0

✿ Hai hàm trên trả về con trỏ null khi có lỗi trong quá trình cấp phát vùng nhớ

- ✿ Bộ nhớ động được quản lý bởi hệ điều hành trong môi trường đa nhiệm → được chia sẻ giữa hàng loạt các ứng dụng  
→ Có thể thiếu bộ nhớ

```
void main() {
 int i, kthuc; int *a;
 double *dPtr = malloc(sizeof(double));
 printf("Hay nhap vao kich thuc cua mang: ");
 scanf("%d", &kthuc);
 a = (int *) calloc(kthuc, sizeof(int));
 if (a == NULL || dPtr == NULL) //Khong du bo nho
 printf("Khong du bo nho");
 else // Da duoc cap phat du
 {
 *dPtr = -8.67;
 for (i=0;i<kthuc;i++)
 a[i] = i;
 a[kthuc-1] = *dPtr;
 for (i=0; i<kthuc; i++)
 printf("%d", a[i]);
 }
 getch();
}
```



```
Hay nhap vao kich thuc cua mang: 7
012345-8
```



## 5.1.1. Cấp phát vùng nhớ (2)

- ✿ Cấp phát vùng nhớ thành công sẽ trả về một con trỏ trỏ đến byte đầu tiên của vùng nhớ
- ✿ Bất cứ một kiểu đối tượng nào cũng có thể lưu vào vùng nhớ được cấp phát (con trỏ void).
- ✿ Vùng nhớ được cấp phát được giữ cho chương trình của bạn cho tới khi bạn giải phóng nó bằng việc gọi hàm `free( )` hoặc `realloc( )` hoặc khi kết thúc chương trình

## 5.1.2. Thay đổi và giải phóng bộ nhớ

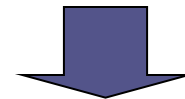
✚ **void free(void \* ptr) ;**

- ✚ Giải phóng vùng nhớ động bắt đầu từ địa chỉ ptr
- ✚ Nếu ptr là con trỏ null thì lời gọi hàm là vô nghĩa.

✚ **void \*realloc(void \* ptr, size\_t n) ;**

- ✚ Giải phóng vùng nhớ từ địa chỉ ptr và cấp phát một vùng nhớ mới n byte và trả về địa chỉ của vùng nhớ đó
- ✚ Vùng nhớ mới có thể bắt đầu giống địa chỉ của vùng nhớ cũ.
- ✚ Giữ nguyên nội dung của vùng nhớ cũ cho tới kích thước của vùng nhớ nào nhỏ hơn
  - Nếu vùng nhớ mới không bắt đầu từ địa chỉ của vùng nhớ cũ thì hàm realloc( ) sao chép nội dung vùng nhớ cũ sang vùng nhớ mới.
  - Nếu vùng nhớ mới lớn hơn vùng nhớ cũ thì giá trị của các byte dư ra không được xác định.
- ✚ Nếu ptr là con trỏ null thì hàm này giống như hàm malloc()

```
#include <string.h>
void main()
{
 char *p = malloc(17);
 if(p == NULL) {
 printf("Co loi khi cap phat bo nho\n");
 exit(1);
 }
 strcpy(p, "Xau gom 16 ky tu");
 p = realloc(p, 18);
 if(!p) {
 printf("Co loi khi cap phat bo nho\n");
 exit(1);
 }
 strcat(p, "!!");
 printf(p);
 free(p);
 getch();
}
```



Xau gom 16 ky tu!!

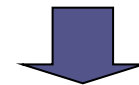
```

#include <string.h>
void main()
{
 char dong[80], *tbao;
 puts("Nhap vao mot dong: "); gets(dong);
 tbao = realloc(NULL, strlen(dong)+1);

 strcpy(tbao, dong); puts(tbao);
 puts("Nhap vao mot dong khac: "); gets(dong);
 tbao = realloc(tbao, (strlen(tbao)
 + strlen(dong)+1));

 strcat(tbao, dong);
 puts(tbao);
 getch();
}

```



```

Nhap vao mot dong:
Xin chao cac ban
Xin chao cac ban
Nhap vao mot dong khac:
Lap trinh C that la thu vi
Xin chao cac banLap trinh C that la thu vi

```

## 5.2. Quản lý bộ nhớ động trong C++

- ✿ Trong thư viện **<new>**

- ✿ Xin cấp phát bộ nhớ: Toán tử **new**

  - ✿ **bienConTro = new kieuDL;**

    - xin cấp phát một vùng nhớ cho một biến đơn

  - ✿ **bienConTro = new kieuDL[kichThuoc];**

    - xin cấp phát cho một mảng các phần tử có cùng kiểu với kiểu dữ liệu

  - ✿ Khi có lỗi về việc cấp phát bộ nhớ, toán tử new sẽ trả về con trỏ NULL.

- ✿ Giải phóng bộ nhớ: Toán tử **delete**

  - ✿ **delete bienConTro; // xóa 1 biến đơn**

  - ✿ **delete []bienMang; // xóa 1 biến mảng**

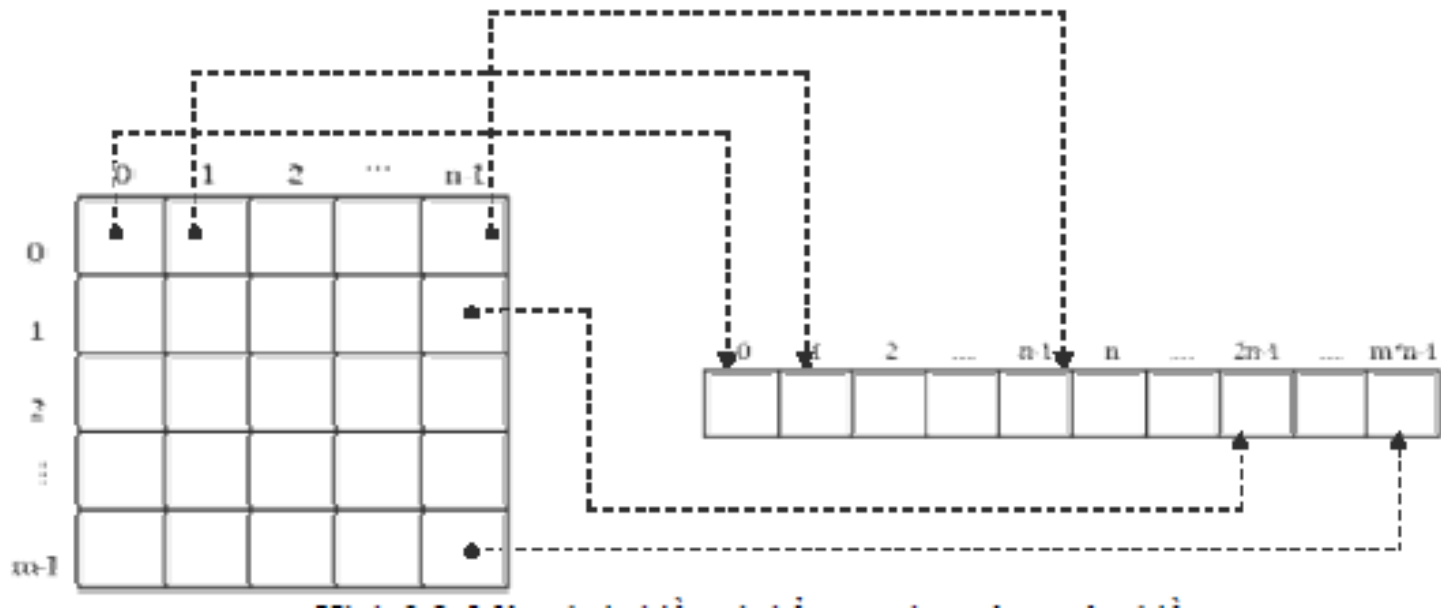
# Ví dụ - Quản lý bộ nhớ động trong C++

```
#include <iostream>

int main() {
 int i,n; long total=100,x,*ptr;
 cout << "Vao so ptu "; cin >> n;
 ptr = new long [n];
 if (ptr==NULL) exit(1);
 for (i=0;i<n;i++){
 cout <<"\n Vao so thu "<< i+1 <<" :";
 cin >> ptr[i]
 }
 cout << "Danh sach cac so : \n"
 for (i=0;i<n;i++) cout << ptr[i] << ",";
 delete [] ptr;
 return 0;
}
```

# Dùng bộ nhớ động cho mảng

Ta có thể coi một mảng 2 chiều là 1 mảng 1 chiều như hình sau:



Gọi X là mảng hai chiều có kích thước m dòng và n cột.  
A là mảng một chiều tương ứng ,thì  $X[i][j] = A[i*n + j]$