مقدمه

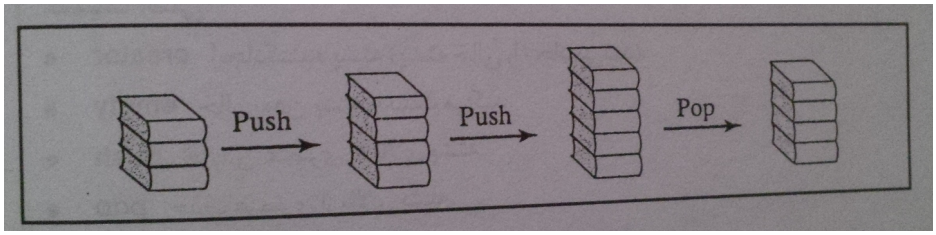
ساختمان داده[1] ی پشته به صورتی است که عمل حذف و اضافه کردن عناصر از یک طرف انجام می شود. یعنی عنصری که دیرتر از همه وارد شده، زودتر از بقیه خارج می شود[2].
مثال: تعدادی بشقاب روی هم می چینید، پشته ای از بشقاب ها ایجاد کرده اید. هنگام برداشتن بشقاب، آن که بالای بقیه قرار دارد را زودتر بر می دارید، درغیر این صورت ... .

نوع داده انتزاعی[3] پشته

داده	مجموعه ای از عناصر که از یک طرف قابل دستیابی اند و این طرف را، بالای[4] پشته می گویند.
عملیات	• پشته خالی ایجاد می کند. • خالی بودن پشته را تست می کند. • عنصری به بالای پشته اضافه می کند. • عنصری از بالای پشته برمی گرداند. (اما آن را از پشته حذف نمی کند.) • عنصری از بالای پشته حذف می کند.

تمامی عملیات بالا با توجه به چگونگی پیاده سازی پشته تفاوت های ناچیزی با هم دارند.
در شکل زیر آن‌ها را با گذاشتن و برداشتن کتاب، شبیه سازی کرده ایم:

1 data structure
2 **LIFO** - **L**ast **I**n **F**irst **O**ut
3 **ADT** - **A**bstract **D**ata **T**ype
4 top



پیاده سازی پشته

ساختمان داده پشته را به دو روش:

- خطی (آرایه[5])
- اشاره گر (لیست پیوندی[6])

می‌توان پیاده‌سازی کرد.

شکل زیر، پیاده سازی پشته با آرایه و چیدمان داده ها در حافظه را نشان می دهد:

شکل

مثال (پیاده سازی پشته با آرایه): در شکل زیر، نمونه ای از حذف و اضافه عناصر به پشته ای که با آرایه ساخته شده است را، بررسی می کنیم:

شکل

در شکل داریم:

1. طول آرایه[7] ثابت[8] و از قبل تعریف شده[9] است.
2. با افزودن عنصر 95 یکی به شمارنده *top* اضافه می شود.
3. اگر به اضافه کردن عنصر ادامه دهیم، سر ریز[10] رخ می دهد.
4. آرایه عنصر 80 را حذف نمی کند، اما طول پشته می تواند متغیر[11] باشد.

شمارنده *top* آخرین عنصر بالای پشته (نه آرایه) را نشان می دهد. اگر به حذف کردن عنصر ادامه دهیم، پاریز[12] رخ می دهد.

قطعه کد زیر کلاس[13] پیاده‌سازی پشته با آرایه در زبان *C++* را نشان می دهد:

```
#define SIZE 5
class stack {
   public:
      stack();
```

5 Array
6 Linked List
7 Array Size
8 Fixed
9 Predefined
10 Overflow
11 Dynamic
12 Underflow
13 C++ Class

```
        int empty();
        void push(int x);
        int pop();
        int top();
    private:
        int my_top;
        int items[SIZE];
};
```

مثال (پیاده سازی پشته با لیست پیوندی): از معایب پیاده سازی به روش آرایه:

- طول ثابت و از پیش تعریف شده آرایه
- همگن[14] بودن عناصر[15]

است. برای واقف آمدن بدین مشکل ها از لیست های پیوندی استفاده می کنیم. در این صورت عناصر می توانند ازهر نوعی (طول و اندازه ای) باشند. البته مشکل لیست پیوندی این است که با زیاد شدن تعداد عناصر، پیمایش[16] کردن آن ها زمان گیر خواهد شد.

در شکل زیر اضافه کردن مقدار 5 به پشته پیاده سازی شده با لیست پیوندی را بررسی می کنیم:

5. *jpg*

قطعه کد زیر کلاس پیاده‌سازی پشته با لیست پیوندی در زبان $C++$ را نشان می دهد:

```
class node {
    friend class stack;
    private:
        int info;
        node *next;
},
class stack {
    public:
        // member functions
    private:
        node *my_top;
}
```

کاربرد های پشته

در این قسمت دو تا از کاربرد های ساختمان داده ی پشته را بررسی می کنیم:

- فرا خوانی تابع[17]
- ارزیابی عبارات ریاضی[18]

14 Homogeneity
15 Element
16 Traverse
17 Function Call
18 Expression Evaluate

فراخوانی تابع

هر وقت تابعی فراخوانی شود، یک رکورد فعالیت[19]، برای آن تابع ایجاد می شود و محیط[20] فعلی (که شامل آدرس شروع تابع در حافظه است.) را برای آن تابع ذخیره می کند. رکورد فعالیت شامل اطلاعاتی چون:

- پارامتر[21] ها
- اطلاعات حالت فراخوان، مانند محتویات ثبات[22] ها، آدرس های برگشت[23].
- متغیر های محلی[24]، حافظه های موقت برای انجام محاسبات میانی.

معمولاً این توابع از نوع بازگشتی[25] هستند و بنابراین خود تابع چندین دفعه با پارامتر های متفاوت فراخوانی می شود. یا ممکن است به صورت توابع تو در تو[26] باشد. یعنی در دل یک تابع، تابع دیگری تعریف یا صدا زده، شده باشد.

6. *jpg*

چون ممکن است هر تابع، توابع دیگری را فراخوانی کند و اجرای خود آن تابع به تعویق[27] افتد، رکورد آن باید طوری ذخیره شود، که وقتی اجرای تابع از سر گرفته می شود، بتواند به رکورد فعالیت اش دست یابد و اجرای خود را از سر گیرد. ساختمان داده‌ای که رکورد های فعالیت را ذخیره می کند، باید رفتار *LIFO* داشته باشد. زیرا اولین تابعی که خاتمه یابد، آخرین تابعی است که فراخوانی شده است و رکورد فعالیت آن باید زودتر از همه بازیابی شود. به نظر میرسد که پشته، ساختمان داده مناسبی برای این کار است.و از آن جایی که این پشته در زمان اجرا دستکاری می شود. پشته زمان اجرا[28] نام دارد.

مراحل استفاده از پشته برای فراخوانی تابع

1. یک کپی از رکورد فعالیت تابع در پشته ذخیره می شود.
2. پارامتر ها را ذخیره می کند.
3. کنترل به آدرس شروع بدنه تابع منتقل می شود.
4. رکورد فعالیت بالای پشته زمان اجرا، مربوط به تابع در حال اجراست. وقتی تابع خاتمه می یابد، عمل *pop* رکود فعالیت آن را از پشته حذف می‌کند و رکورد

19 Activation Record
20 Workspace
21 Parameter
22 Register
23 Return Address
24 Local Variable
25 Recursive
26 Nested Functions
27 Postponed
28 Run-Time Stack

فعالیت تابع قبلی (که این تابع را فراخوانی کرده بود) بالای پشته قرار می گیرد.

مثال (تابع فاکتوریل[29]): برای نشان دادن پروسه فراخوانی تابع، اجرای تابع فاکتوریل را بررسی می کنیم:

```
long int fact (int n) {
    int x, y;
    // basis case
    if (n == 0)  return 1;
    // step case
    x = n - 1;
    y = fact(x);
    return n * y;
}
```

در شکل زیر پروسه اجرای تابع بازگشتی فاکتوریل بررسی می شود:

7.jpg

همان طور که در تابع بازگشتی و استفاده از ساختمان داده پشته مشاهده کردیم، اجرای هر تابع به اجرای تابع های قبلی وابسته[30] است. یعنی خروجی همه توابع را باید در حافظه نگه داری کنیم، چون بعدا به سراغ آن ها باز خواهیم گشت. این روند، زمان[31] اجرا و هم حافظه[32] زیادی مصرف می کند. این عیب بزرگ توابع بازگشتی است. در حالی که تنها حسن آن نوشتن تعداد خطوط دستور [33]کمتر توسط برنامه نویس است. در مثال فاکتوریل، تنها 2 دسته دستور مشاهده کردیم:

- حالت پایه[34]
- حالت بعدی[35] (فراخوانی همین تابع با پارامتر جدید)

این دسته دستورات مشابه 3 مرحله استقرای ریاضی[36] برای پیدا کردن مدل[37] (فرمول[38])، از روی بررسی روابط ورودی-خروجی[39] است:

- حالت پایه[40]

29 Factorial
30 Multiple Chained Dependencies
31 Time Consumer
32 Space Consumer
33 LOC: Lines Of Code
34 Basis Case
35 Step Case
36 Inductive Reasoning
37 Model
38 Formula
39 Observed Input/Output Samples
40 Base

- فرض استقرا[41]
- حکم استقرا[42]

این در حالی است که، در توابع غیر بازگشتی[43]، تمامی دستورات باید توسط برنامه نویس به ماشین آموزش[44] داده شود. بنابراین برنامه ها بسیار طولانی از نظر تعداد خطوط دستور هستند. اما حسن آن این است که ماشین فقط طبق مراحل دستورات پیش می رود.
بنابراین هم سرعت اجرای برنامه بالا (زمان اجرا پایین) است و هم حافظه کمتری استفاده می شود.
در مسائل مربوط به شبکه های عصبی[45]، یادگیری عمیق[46]، یادگیری تقویتی[47] و ... از توابع بازگشتی استفاده می شود. زیرا منطق فکر کردن و نتیجه گیری را بر عهده ماشین می گذارند که همچون انسان رفتار کند.

ارزیابی عبارات ریاضی

برای نشان دادن عبارات ریاضی 3 روش وجود دارد:

- میانوندی[48]
- پیشوندی[49]
- پسوندی[50]

میانوندی

نمایش عبارات ریاضی به این روش برای انسان راحت تر قابل درک و فهم[51] است.

$$in-order-form:\ a\ +\ b$$

پیشوندی

پیاده سازی یک عبارت ریاضی به روش پیشوندی در برنامه نویسی راحت تر است. به طور مثال، عملگر + را در نظر گرفته که دو عملوند چپ و راست خود را (اگر از نوع عدد باشند) با هم جمع می کند. برای پیاده سازی آن، به صورت زیر کد نویسی می کنیم:

41 Induction Hypothesis
42 Final Step
43 Iterated Function
44 Instructed
45 Artificial Neural Networks
46 Deep Learning
47 Reinforcement Learning
48 In-Order
49 Pre-Order
50 Post-Order
51 Natural

$$function \quad add(a,b) \ \{...\}$$

با مقایسه کد بالا با نمایش پیشوندی به صورت:

$$pre-order-form: \ +ab$$

متوجه می‌شویم که عملگر + همان تابع add در کد و عملوند ها به عنوان آرگومان[52] به تابع فرستاده می شوند.

پسوندی

برای کامپایلر ها تفسیر کردن[53] عبارت های ریاضی به فرم پسوندی، کارآمد[54] تر خواهد بود تا تفسیر آن‌ها به فرم میانوندی. با تبدیل فرم میانوندی به پسوندی، دیگر نیازی به پرانتز های عبارت ریاضی نیست.

$$post-order-form: \ ab+$$

عبارت پسوندی (و همچنین پیشوندی) فاقد پرانتز بوده و فقط عملگر ها و عملوند ها به داخل پشته فرستاده می شوند. از طرف دیگر در فرم میانوندی برای ارزیابی عبارات ریاضی می‌توان با جا به جا کردن پرانتز[55] ها، اولویت[56] انجام عملگر ها را تغییر داد. در مثال زیر عبارت ریاضی بدون پرانتز و با پرانتز جواب های متفاوتی بر می گرداند.[57]

مثال: عبارت میانوند زیر را در نظر بگیرید که در آن تقدم عملگر ضرب * زودتر از عملگر جمع + است:

```
int a = 1;
int b = 2;
int c = 4;
cout << “out-put:” << endl;
cout << a + b * c << endl;
cout << (a + b) * c << endl;
cout << a + (b * c) << endl;

out-put:
9
12
9
```

از آن جایی که کامپایلر از فرم پسوندی (فاقد پرانتز) برای تفسیر عبارات ریاضی استفاده می کند، پارسر اش باید جدولی تحت عنوان جدول اولویت[58] نگه داری کند. از روی این جدول، مشخص می‌شود که کدام عملگر را زود تر انجام دهد (وارد پشته کند). نمونه‌ای از این جدول در شکل آمده است:

52 Argument
53 Parsing
54 Efficient
55 Parentheses
56 Operator Precedence
57 Who wants to remember the rules for operator precedence? If there might be any doubt, use parentheses to clarify expressions.
58 Precedence Table

8. *jpg*

در مثال های بعدی کاربرد و طرز عملکرد پشته را برای ارزیابی عبارات ریاضی و کنترل تعداد پرانتز ها، بررسی خواهیم کرد.

مثال (تبدیل عبارت میانوندی بدون پرانتز به پسوندی):

9. *jpg*

مثال (تبدیل عبارت میانوندی پرانتز دار به پسوندی):

10. *jpg*

تمرین: تبدیل عبارات زیر از میانوندی به پسوندی و بر عکس را، بررسی کنید.

11. *jpg*

مثال (جای گذاری مقادیر در عبارت پسوندی شده):

12. *jpg*

محدودیت های برنامه ها و مثال هایی که تا به حال بررسی کردیم، به قرار زیر هستند:

- فرض شده که عبارات پسوندی معتبر هستند. مثلا تعداد پرانتز های باز و بسته عبارت میانوندی مساوی است.
- عملگر ها یک کاراکتری هستند. مثلاً عملگر جمع + یک کاراکتری است. در حالی که این امکان وجود دارد در یک عبارت ریاضی، عملگر بیش از یک کاراکتر باشد. به طور مثال عملگر XOR که 3 کاراکتری است.
- عملوند ها یک رقمی هستند. مثلا عملوند 3 یا 8 یک رقمی هستند. در حالی که این امکان وجود دارد در یک عبارت ریاضی، عملوند بیش از یک رقم باشد. به طور مثال عدد 35 یا عدد 49 که دو رقمی اند.

راه حل محدودیت های بالا، استفاده از لیست پیوندی است. یعنی هر عملگر یا عملوند را در یک گره[59] قرار دهیم.

تمرین: یکی از کاربرد های پشته تشخیص توازن[60] یک عبارت ریاضی است. یعنی ارزیابی شود آیا تعداد پرانتز های باز و بسته، مساوی هستند یا خیر.

به عنوان مثال عبارت زیر متوازن است، زیرا تعداد پرانتز های باز و بسته مساوی است:

$$(a + (b + c))$$

در حالی که عبارت زیر نامتوارن است، زیرا تعداد پرانتز های باز و بسته برابر نیست:

$$a + (b + c))$$

با توجه به شبه کد زیر، برنامه ای بسازید که متوازن بودن یا نبودن یک عبارت ریاضی را مشخص کند.

1. عبارت را از ورودی بخوان.

59 Node
60 Balanced Parentheses

2. نمادی را از عبارت جدا کن.
3. اگر نماد، پرانتز باز است: در پشته بیانداز.
4. اگر نماد، عملوند باشد: آن را رد کن.
5. اگر نماد، عملگر باشد: یک پرانتز باز از پشته بردار. اگر:
    - پشته خالی شد، آن گاه عبارت متوازن است.
    - اگرپشته خالی نشد یا نتوانیم مقداری ا ز پشته برداریم، عبارت متوازن نیست.