

کمپیوٹیشنل تھنکنگ

اس باب کا مطالعہ کرنے کے بعد طلبہ اس قابل ہو جائیں گے وہ:

- کمپیوٹیشنل تھنکنگ اور اس کے کلیدی اجزاء کی وضاحت کر سکیں: پرابلم بریک ڈاؤن، پیٹرن کی شناخت، اسٹرکشن اور الگورتھم۔
- کمپیوٹیشنل تھنکنگ کے اصولوں کی وضاحت کر سکیں، بشمول مسئلہ کی تفہیم، مسئلہ کو آسان بنانا، اور حل کا انتخاب اور ڈیزائن۔
- الگورتھم ڈیزائن کے طریقوں، خاص طور پر فلو چارٹ اور سوڈوکوڈ کی وضاحت کریں، اور ان کے مابین اختلافات کو سمجھ سکیں۔
- الگورتھم کو بصری طور پر پیش کرنے کے لیے فلو چارٹ بنائیں اور ان کی تشریح کر سکیں۔
- ایک منظم، انسانی پڑھنے کے قابل فارمیٹ میں الگورتھم کا خاکہ پیش کرنے کے لیے ”سوڈوکوڈ“ لکھ سکیں۔
- الگورتھم سرگرمیوں میں مشغول رہیں جیسے ڈیزائن اور تشخیص کی تکنیک۔
- ان کی درستی کی دستی طور پر تصدیق کرنے کے لیے فلو چارٹ اور سوڈوکوڈ کے ڈرائی رن کریں۔
- LARP کے تصور اور اہمیت کو سمجھ سکیں (منطق کی منطق) مسائل کے حل کے لیے الگورتھم۔
- لکھنے کے الگورتھم اور ڈرائنگ فلو چارٹ کی مشق کرنے کے لیے LARP سرگرمیوں کو نافذ کر سکیں۔
- الگورتھم میں مختلف قسم کی غلطیوں کی نشاندہی کر سکیں، بشمول ٹیکسٹ کی غلطیاں، منطقی غلطیاں، اور رن ٹائم غلطیاں۔
- الگورتھم میں غلطیوں کو تلاش کرنے اور ٹھیک کرنے کے لیے ڈیبگنگ تکنیک کا اطلاق کر سکیں۔
- LARP کے دوران پیش آنے والے عام غلطی کے پیغامات کو پہچان کر ان کو حل کرنے کا طریقہ سیکھ سکیں۔
- حقیقی دنیا کے منظر نامے پر کمپیوٹیشنل تھنکنگ کے اصولوں کو لاگو کر کے مسئلہ حل کرنے کی مہارت کا مظاہرہ کر سکیں۔
- مختلف الگورتھم کی کارکردگی کا اندازہ کر سکیں اور کارکردگی کے تجزیہ کی بنیاد پر انھیں بہتر بنائیں۔

تعارف

کمپیوٹیشنل تھنکنگ ایک اہم مہارت ہے جو افراد کو پیچیدہ مسائل کو حل کرنے کے قابل بناتی ہے اور یہ ان طریقوں کا استعمال کرتی ہے جو کمپیوٹر سائنس کے عمل سے مشابہت رکھتے ہیں۔ یہ باب کمپیوٹیشنل تھنکنگ کی تعریف کرنے اور اسے اس کے بنیادی اجزاء میں تقسیم کرنے سے شروع ہوتا ہے جیسے تقسیم، تجزیہ، پیٹرن کی شناخت، تجرید اور الگورتھم وغیرہ۔ یہ اجزاء پیچیدہ مسائل کو آسان بنانے، اور ایسے پیٹرن کی نشاندہی کرنے کے لیے ضروری ہیں جو حل کا باعث بن سکتے ہیں، غیر ضروری مسائل کو نظر انداز کرتے ہوئے متعلقہ تفصیلات پر توجہ مرکوز کرتے ہیں، اور مسائل کو حل کرنے کے لیے مرحلہ وار طریقہ کار تشکیل دیتے ہیں۔ ان تصورات کو سمجھنا نہ صرف کمپیوٹر سائنس دانوں کے لیے مفید ہے بلکہ ہر اس شخص کے لیے بھی فائدہ مند ہے جو مختلف شعبوں میں اپنے مسائل کو حل کرنے کی مہارت کو بہتر بنانا چاہتا ہے۔

کمپیوٹیشنل تھننگ کی وضاحت کرنے کے علاوہ یہ باب ان اصولوں کا بھی جائزہ لیتا ہے جو رہنمائی فراہم کرتے ہیں، جیسے مسئلے کو سمجھنا، اسے زیادہ منظم بنانے کے لیے آسان بنانا، اور بہترین حل اور ڈیزائن کا انتخاب کرنا۔ اس باب میں الگورتھم ڈیزائن کرنے کے مختلف طریقوں کو بھی متعارف کرایا گیا ہے، بشمول فلو چارٹ اور سوڈو کوڈ کا استعمال، اور ان دونوں طریقوں کے درمیان فرق کی بھی وضاحت کی گئی ہے۔ مزید برآں، یہ LARP (لا جک آف الگورتھم فار ریزولوشن آف پرائلمز) جیسی سرگرمیوں کے ذریعے الگورتھم ڈیزائن اور تشخیص کی مشق کرنے کی اہمیت پر بھی زور دیتا ہے۔ آخر میں، باب غلطی کی شناخت اور ڈیبگنگ کے ضروری پہلوؤں کا احاطہ کرتا ہے، الگورتھم کے نفاذ کے دوران پیش آنے والی عام غلطیوں کو پہچاننے اور ٹھیک کرنے کے لیے تکنیک فراہم کرتا ہے۔ ان مہارتوں کو سیکھنے سے، طلبہ کمپیوٹیشنل پرائلمز کی ایک وسیع رینج سے مؤثر طریقے سے نمٹنے کے لیے اچھی طرح سے لیس ہوں گے۔

7.1 کمپیوٹیشنل تھننگ کی تعریف

کمپیوٹیشنل تھننگ (CT) کسی مسئلہ کو حل کرنے کا ایک عمل ہے جس میں پیچیدہ مسائل کو اس طرح حل کرنے کے لیے مہارتوں اور تکنیکوں کا ایک مجموعہ شامل ہے جس پر کمپیوٹر بھی عمل درآمد کر سکتا ہے۔ اس طریقہ کار کو کمپیوٹر سائنس سے باہر بھی مختلف شعبوں میں استعمال کیا جاسکتا ہے، جیسے بیالوجی، ریاضی اور یہاں تک کہ روزمرہ کی زندگی۔ آئیے کمپیوٹیشنل تھننگ کو اس کے بنیادی اجزاء میں تقسیم کرتے ہیں:

کیا آپ جانتے ہیں؟

کمپیوٹیشنل تھننگ صرف کمپیوٹر سائنس تک محدود نہیں ہے۔ یہ روزمرہ کے مسائل کو حل کرنے میں استعمال ہوتی ہے، جیسے ایک سفر کی منصوبہ بندی کرنا یا کاموں کی ترتیب دینا۔

7.1.1 تقسیم یا تجزیہ (Decomposition)

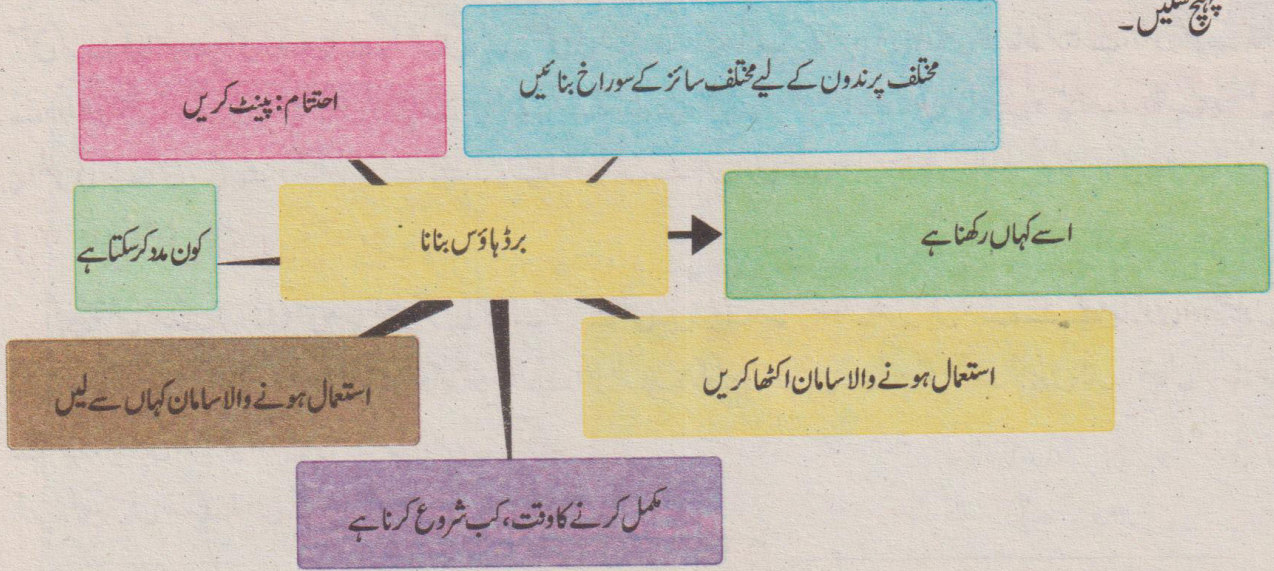
تقسیم ایک پیچیدہ مسئلے کو چھوٹے، زیادہ منظم حصوں میں تقسیم کرنے کا عمل ہے۔ مثال کے طور پر، اسکول کی تقریب کی منصوبہ بندی کرتے وقت، آپ کام کو کچھ حصوں میں تقسیم کر سکتے ہیں جیسے تاریخ کا انتخاب کرنا، مقام کی بکنگ کرنا، دعوت نامے بھیجنا اور سرگرمیوں کا اہتمام کرنا۔

کمپیوٹیشنل تھننگ تقسیم میں ایک اہم قدم ہے۔ اس میں ایک پیچیدہ مسئلے کو چھوٹے، قابل انتظام حصوں میں تقسیم کرنا شامل ہے۔ آئیے! پرندوں کے گھر کی تعمیر کی مثال لیتے ہیں۔ یہ کام شروع میں مشکل لگ سکتا ہے، لیکن اگر ہم اسے چھوٹے حصوں میں تقسیم کر دیں تو، ایک وقت میں ہم ایک حصے کو سنبھال سکتے ہیں۔

یہاں بتایا گیا ہے کہ ہم پرندوں کے گھر کی تعمیر کے کام کو کس طرح تقسیم کر سکتے ہیں۔ تصویر 1.7 پرندوں کے گھر کی تعمیر کے لیے تقسیم کو ظاہر کرتی ہے۔

- برڈ ہاؤس ڈیزائن کریں: سائز، شکل اور ڈیزائن پر فیصلہ کریں۔ ایک منصوبہ بنائیں اور تمام ضروری پیمائش جمع کریں۔
- مواد جمع کریں: لکڑی، کیل، پینٹ اور تھوڑے اور آری جیسے اوزار سمیت تمام ضروری مواد کی فہرست بنائیں۔
- لکڑی کو کاٹیں: ڈیزائن کے مطابق لکڑی کی پیمائش کریں اور مطلوبہ حصوں میں کاٹیں۔

- (iv) حصوں کو جوڑنا: پرندوں کے گھر کی ساخت بنانے کے لیے لکڑی کے حصوں کو جوڑنے کے منصوبے پر عمل کریں۔
- (v) رنگ کریں اور سجائیں: پرندوں کے گھر کو رنگ کریں اور اسے پرندوں کے لیے پرکشش بنانے کے لیے آرائشی اشیاء لگائیں۔
- (vi) برڈ ہاؤس نصب کریں: ایک مناسب جگہ تلاش کریں اور محفوظ طریقے سے برڈ ہاؤس نصب کریں جہاں پرندے آسانی سے اس تک پہنچ سکیں۔



تصویر 7.1: پرندوں کے گھر کی تعمیر

سرگرمی

کسی کام کو ڈی کمپوز کریں۔ ایک پیچیدہ کام کے بارے میں سوچیں جو آپ باقاعدگی سے کرتے ہیں، جیسے اسکول کی تقریب کا اہتمام کرنا یا کھانا پکانا۔ اسے چھوٹے، قابل انتظام حصوں میں تقسیم کریں۔ ہر مرحلے کو لکھیں اور اپنے ہم جماعتوں کے ساتھ تبادلہ خیال کریں کہ کس طرح تقسیم کام کو سنبھالنے میں آسان بناتی ہے۔

7.1.2 پیٹرن کی شناخت

پیٹرن کی شناخت میں مسائل کے درمیان اور ان کے اندر مماثلت یا نمونے تلاش کرنا شامل ہے۔ مثال کے طور پر، اگر آپ محسوس کرتے ہیں کہ آپ ہمیشہ سوموار کو اپنا ہوم ورک بھول جاتے ہیں تو آپ ایک پیٹرن کو پہچان سکتے ہیں اور خاص طور پر اتوار کے لیے ایک یاد دہانی ترتیب دے سکتے ہیں۔

پیٹرن کی شناخت کمپیوٹیشنل تھنکنگ کا ایک اہم جزو ہے۔ اس میں کسی بھی ڈیٹا یا مسئلے کے اندر ترتیب، باقاعدگی یا تکرار کو سمجھنا شامل ہے۔ آئیے مربعوں کے رقبے میں پیٹرن تلاش کرنے کی مثال پر غور کریں:

تصویر 7.2 میں اوپری قطار مربعوں کی سائیڈ کی لمبائی کی نمائندگی کرتی ہے، جو 1 سے 7 تک ہے۔ نچلی قطار ان مربعوں کے رقبہ کو ظاہر کرتی ہے۔ یہاں ہم ایک نمونہ دیکھ سکتے ہیں کہ کس طرح رقبہ میں اضافہ ہوتا ہے۔

• سائیڈ لمبائی 1: $1 = 1^2$ = رقبہ

- سائیڈ لمبائی 2: $2^2 = 4$ (1+3) = رقبہ
- سائیڈ لمبائی 3: $3^2 = 9$ (1+3+5) = رقبہ
- سائیڈ لمبائی 4: $4^2 = 16$ (1+3+5+7) = رقبہ
- سائیڈ لمبائی 5: $5^2 = 25$ (1+3+5+7+9) = رقبہ
- سائیڈ لمبائی 6: $6^2 = 36$ (1+3+5+7+9+11) = رقبہ
- سائیڈ لمبائی 7: $7^2 = 49$ (1+3+5+7+9+11+13) = رقبہ

ہم دیکھ سکتے ہیں کہ ہر مربع کے رقبے کا حساب لگاتار طاق نمبروں کو جمع کر کے لگایا جاسکتا ہے۔ مثال کے طور پر، 3 کی سائیڈ لمبائی کے ساتھ ایک مربع کا رقبہ پہلے تین طاق نمبروں کو جمع کر کے حاصل کیا جاسکتا ہے:

$$1 + 3 + 5 = 9$$

Visual/Numerical Pattern

Goes up by 1

		+1	+1	+1	+1	+1	+1
Side	1	2	3	4	5	6	7
Area	1	4	9	16	25	36	49
		+3	+5	+7	+9	+11	+13

تصویر 7.2: 1 سے 7 تک اطراف والے مربعوں کے رقبے کا پیٹرن

کیا آپ جانتے ہیں؟

1 سے 10 تک سائیڈ لمبائی کے ساتھ ایک ٹیبل بنائیں۔ لگاتار طاق نمبروں کو جمع کرنے کے پیٹرن کا استعمال کرتے ہوئے مربعوں کے رقبوں کا حساب لگائیں۔ سائیڈ کی لمبائی کو ترتیب دے کر اپنے نتائج کی تصدیق کریں اور دیکھیں کہ آیا پیٹرن برقرار ہے یا نہیں۔

7.1.3 تجرید (Abstraction)

تجربہ مسئلہ حل کرنے میں، خاص طور پر کمپیوٹر سائنس میں، ایک بنیادی تصور ہے۔ اس میں پیچیدہ مسائل کو چھوٹے، زیادہ منظم حصوں میں تقسیم کر کے آسان بنانا، اور غیر ضروری حصوں کو نظر انداز کرنا۔ صرف ضروری تفصیلات پر توجہ مرکوز کرنا شامل ہے۔ اس سے مسائل کو زیادہ مؤثر طریقے سے سمجھنے، ڈیزائن کرنے اور حل کرنے میں مدد ملتی ہے۔

تعریف:

تجربہ پیچیدہ حقیقت کو چھپانے اور صرف ضروری عناصر کو نمایاں کرنے کا عمل ہے۔ یہ ہمیں تفصیلات میں گم ہوئے بغیر اعلیٰ سطح کے جائزہ پر توجہ مرکوز کرتے ہوئے پیچیدگی کو کم کرنے میں مدد کرتا ہے۔

مثال: چائے کا کپ بنانا:

بلند سطح کے اقدامات:

(i) پانی ابالیں۔

(ii) چائے کی پتی یا چائے کا بیگ شامل کریں۔

(iii) چند منٹ کے لیے دم دینے دیں۔

(iv) ایک کپ میں ڈالیں اور اگر چاہیں تو دودھ/چینی شامل کریں۔

یہ مثال دکھاتی ہے کہ ہم تفصیلات جسے چائے کے برتن کی قسم، پانی کی مقدار، اور دیگر غیر ضروری تفصیلات کو نظر انداز کر کے صرف بنیادی مراحل پر توجہ دیتے ہیں جو تجدید کا بنیادی اصول ہے۔

دلچسپ معلومات:

یہ پیچیدہ مسائل کو حل کرتے وقت انہیں چھوٹے حصوں میں توڑنے کی کوشش کریں اور اہم مراحل پر توجہ مرکوز کریں۔ یہ آپ کو مسئلے کو بہتر طور پر سمجھنے میں مدد دے گا اور حل تلاش کرنا زیادہ آسان بنائے گا۔ اب straction کا استعمال کرتے ہوئے، ہم پیچیدہ مسائل کا سامنا کر سکتے ہیں۔

7.1.3 الگورتھم

HOW TO BAKE A CAKE?

- 1) Preheat the oven
- 2) Gather the ingredients
- 3) Measure out the ingredients
- 4) Mix together the ingredients to make the batter
- 5) Grease a pan
- 6) Pour the batter into the pan
- 7) Put the pan in the oven
- 8) Set a timer
- 9) When the timer goes off, take the pan out of the oven
- 10) Enjoy!



الگورتھم کسی مسئلے کو حل کرنے یا کسی کام کو مکمل کرنے کے لیے مرحلہ وار ہدایات کا ایک مجموعہ ہوتا ہے۔ بالکل ایک ایک پکانے کی ترکیب پر عمل کرنے کی طرح۔ الگورتھم ایک منظم اور واضح ترتیب میں دی گئی ہدایات کا مجموعہ ہوتا ہے۔ جس پر عمل کر کے کسی خاص مقصد کو حاصل کیا جاسکتا ہے۔ یہ ایک ترکیب یا ہدایات کے ایک سیٹ کی طرح ہے جو آپ کو بتاتا ہے کہ کسی کام کو کس طرح کرنا ہے۔

مثال 1: کیک بنانا:

شکل 7.3

تصویر میں ہم کیک بنانے کی ترکیب دیکھتے ہیں۔ ترکیب اجزا کی ایک

فہرست فراہم کرتی ہے اور انہیں ملانے اور کیک پکانے کے لیے مرحلہ وار ہدایات فراہم کرتی ہے۔ یہ الگورتھم کی ایک مثال ہے کیونکہ یہ کیک بنانے کے مقصد کو حاصل کرنے کے لیے اقدامات کی ایک واضح ترتیب کا خاکہ پیش کرتا ہے۔

مثال 2: درخت لگانا: یہاں درخت لگانے کے لیے ایک آسان الگورتھم دیا گیا ہے جو ایک مفید اور بامعنی سرگرمی ہو سکتی ہے:

(i) اپنے باغ میں ایک مناسب جگہ کا انتخاب کریں۔

(ii) ایک گڑھا کھودیں جو درخت کی جڑ سے دو گنا چوڑا ہو۔

(iii) درخت کو گڑھے میں رکھیں، اس بات کو یقینی بنائیں کہ درخت سیدھا ہے۔

(iv) گڑھے کو مٹی سے بھریں، ہوا کو ہٹانے کے لیے اسے آہستہ سے دبائیں۔

- (v) درخت کو دل کھول کر پانی دیں تاکہ وہ مٹی میں جڑ پکڑ سکے۔
- (vi) نمی کو برقرار رکھنے کے لیے درخت کی جڑوں کے ارد گرد نامیاتی کھاڈا لیں۔
- (vii) درخت کو باقاعدگی سے پانی دیں جب تک کہ یہ بڑھنا شروع نہ ہو جائے۔
- یہ الگورتھم درخت لگانے کے بارے میں واضح ہدایات دیتا ہے، جس سے کسی کے لیے بھی اس پر عمل کرنا آسان ہو جاتا ہے۔

سرگرمی

آئیں ایک الگورتھم بنائیں!

کسی ایسے کام کے بارے میں سوچیں جو آپ ہر روز کرتے ہیں، جیسے دانتوں کو برش کرنا یا اسکول بیگ کو پیک تیار کرنا۔ آپ جن اقدامات پر عمل کرتے ہیں انہیں ایک ایک کر کے لکھیں۔ اپنی کلاس کے ساتھ اپنے الگورتھم کا اشتراک کریں اور دیکھیں کہ آیا آپ کے دوست اس پر عمل کر سکتے ہیں!

کیا آپ جانتے ہیں؟

الگورتھم صرف کمپیوٹرز میں ہی استعمال نہیں ہوتے ہیں؟ وہ ہر جگہ موجود ہیں! جب آپ اپنے دوست کے گھر جانے کے لیے ہدایات پر عمل کرتے ہیں یا قواعد کے ساتھ بورڈ گیم کھیلتے ہیں تو آپ الگورتھم استعمال کر رہے ہوتے ہیں۔ الگورتھم ہمیں منطقی طریقے سے مسائل کو حل کرنے میں مدد کرتے ہیں۔

سرگرمی

- جماعت نہم 9 کے امتحان کے لیے بورڈ آف انٹرمیڈیٹ اینڈ سیکنڈری ایجوکیشن (BISE) میں درخواست دینے کے لیے ایک الگورتھم کا خاکہ تیار کریں۔

الگورتھم چیلنج

- ایک عام کام کے لیے الگورتھم بنانے کے لیے دو دو کے گروپوں میں کام کریں، جیسے سینڈوچ بنانا یا اسکول کے لیے تیار ہونا۔ ہر مرحلہ کو واضح طور پر لکھیں اور پھر دوسرے گروپ کے ساتھ الگورتھم کا تبادلہ کریں۔ ان کے الگورتھم پر بالکل اسی طرح عمل کریں جیسے لکھا گیا ہے اور دیکھیں کہ آیا آپ کام مکمل کر سکتے ہیں۔

7.2 کمپیوٹیشنل تھنکنگ کے اصول

کمپیوٹیشنل تھنکنگ میں کئی اہم اصول شامل ہیں جو منظم انداز میں مسئلہ حل کرنے کے عمل کی رہنمائی کرتے ہیں۔

7.2.1 مسئلہ کی تفہیم

کسی مسئلہ کو سمجھنے میں بنیادی مسئلہ کی شناخت کرنا، ضروریات کی وضاحت کرنا، اور مقاصد کا تعین کرنا شامل ہے۔ مسئلہ کو سمجھنا مسئلہ کو حل کرنے میں پہلا اور سب سے اہم مرحلہ ہے، خاص طور پر کمپیوٹیشنل تھنکنگ میں۔ اس میں حل تلاش کرنے کی کوشش کرنے سے پہلے اس کے بنیادی اجزاء اور ضروریات کی شناخت کرنے کے لیے مسئلہ کا مکمل تجزیہ کرنا شامل ہے۔

”اگر میرے پاس کسی مسئلے کو حل کرنے کے لیے ایک گھنٹہ ہوتا تو میں 55 منٹ اس مسئلے کے بارے میں سوچنے میں گزارتا اور 5 منٹ اس کے حل کے بارے میں سوچنے میں صرف کرتا۔“ (البرٹ آئن سٹائن)

مسئلے کی تفہیم کی اہمیت:

- (i) وضاحت اور توجہ: مسئلے کو مکمل طور پر سمجھنے سے آپ کو معلوم ہو جاتا ہے کہ کیا حل کرنے کی ضرورت ہے۔ اس سے آپ غیر ضروری تفصیلات میں الجھنے کی بجائے اصل مسئلہ پر توجہ مرکوز کر سکتے ہیں۔
- (ii) اہداف کا تعین: مسئلے کی مناسب سمجھ سے آپ واضح اور قابل حصول اہداف طے کر سکتے ہیں۔ آپ اس بات کا تعین کر سکتے ہیں کہ حتمی نتیجہ کیسا ہونا چاہیے اور اس نتیجے تک پہنچنے کے لیے کون سے مخصوص اقدامات ضروری ہیں۔
- (iii) مؤثر حل: جب آپ مسئلے کو اچھی طرح سے سمجھتے ہیں تو آپ زیادہ مؤثر حل بنا سکتے ہیں۔ آپ مسئلے کو حل کرنے میں وقت اور وسائل کی بچت کرنے کے لیے بہترین طریقوں اور آلات کا انتخاب کر سکتے ہیں۔
- (iv) غلطیوں سے بچنا: مسئلے کو اچھی طرح سے سمجھ کر آپ عام نقصانات اور غلطیوں سے بچ سکتے ہیں۔ مسئلے کو غلط سمجھنا غلط حل اور محنت کے ضائع ہونے کا باعث بن سکتا ہے۔

مثال: اسکول کی ویب سائٹ بنانا

فرض کریں کہ آپ کو اپنے اسکول کے لیے ایک ویب سائٹ بنانے کے لیے کہا جاتا ہے۔ کوڈنگ میں جانے سے پہلے، آپ کو مسئلہ سمجھنے کی ضرورت ہے:

- (i) ضروریات کی شناخت: ویب سائٹ میں کن خصوصیات کی ضرورت ہے؟ مثال کے طور پر، خبروں، واقعات، کلاس شیڈول، اور رابطے کی معلومات کے لیے صفحات۔
- (ii) صارف کی ضروریات: ویب سائٹ کا استعمال کون کرے گا؟ طالب علم، اساتذہ یا والدین؟ اپنے سامعین کو سمجھنے سے ایک صارف دوست انٹرفیس ڈیزائن کرنے میں مدد ملے گی۔
- (iii) تکنیکی رکاوٹیں: کون سے وسائل اور آلات دستیاب ہیں؟ کیا آپ کو ویب سرور اور ضروری سافٹ ویئر تک رسائی حاصل ہے؟ ان پہلوؤں کو سمجھ کر، آپ ایک ویب سائٹ کی منصوبہ بندی اور تعمیر کر سکتے ہیں جو آپ کے اسکول کی ضروریات کو مؤثر طریقے سے پورا کرے۔

دلچسپ معلومات:

کسی مسئلے کو حل کرنا شروع کرنے سے پہلے ہمیشہ اسے اچھی طرح سے سمجھنے کے لیے وقت نکالیں۔ سوالات پوچھیں، معلومات جمع کریں اور کسی بھی شک کی وضاحت کریں۔ یہ بنیادی قدم بہتر اور زیادہ مؤثر حل کا باعث بنے گا۔

7.2.2 مسئلے کو آسان بنانا

کسی مسئلے کو آسان بنانے کا مطلب ہے کہ اسے چھوٹے، زیادہ قابل انتظام ذیلی مسائل میں تقسیم کیا جائے۔ مثال کے طور پر: ویب سائٹ ڈیزائن کرنے کے لیے، کاموں کو لے آؤٹ ڈیزائن کرنے، مواد بنانے اور کاموں کو کوڈ کرنے میں تقسیم کریں۔

7.2.3 حل کا انتخاب اور ڈیزائن

بہترین حل کا انتخاب کرنے میں مختلف طریقوں کا جائزہ لینا اور سب سے زیادہ مؤثر طریقے کا انتخاب کرنا شامل ہے۔ حل کو ڈیزائن کرنے کے لیے ایک تفصیلی منصوبہ یا الگورتھم بنانے کی ضرورت ہوتی ہے۔

7.3 الگورتھم ڈیزائن کرنے کے طریقے

الگورتھم ڈیزائن کے طریقے مختلف کمپیوٹیشنل مسائل سے مؤثر طریقے سے نمٹنے کے لیے مختلف ٹولز اور تکنیک فراہم کرتے ہیں۔ ہر طریقہ کار کی اپنی طاقت اور کمزوریاں ہیں، جو اسے مختلف قسم کے مسائل کے لیے موزوں بناتی ہیں۔ مختلف طریقوں کو سمجھنے سے ایک مسئلے کے لیے سب سے مناسب طریقہ منتخب کرنے میں مدد ملتی ہے، جس سے زیادہ مؤثر اور بہتر حل سامنے آتے ہیں۔ آئیے ان میں سے دو طریقوں پر بات کرتے ہیں۔

7.3.1 فلو چارٹ

فلو چارٹ کسی عمل یا نظام کے مراحل کی بصری نمائندگی ہوتے ہیں، جو تیروں سے منسلک مختلف علامتوں کا استعمال کرتے ہوئے بنائے جاتے ہیں۔ یہ وسیع پیمانے پر مختلف شعبوں میں استعمال ہوتے ہیں، بشمول کمپیوٹر سائنس، انجینئرنگ، کاروبار تاکہ عمل کو ماڈل کیا جاسکے، نظاموں کو ڈیزائن کیا جاسکے اور پیچیدہ ورک فلو کو واضح اور مؤثر انداز میں بیان کی جاسکے۔

7.3.1.1 فلو چارٹ کی اہمیت

- وضاحت: فلو چارٹ کسی بھی عمل کو واضح اور مختصر انداز میں پیش کرتے ہیں، جس سے اسے ایک نظر میں سمجھنا آسان ہو جاتا ہے۔
- بلاغ: یہ پیچیدہ عمل کو وسیع سامعین تک پہنچانے کے لیے بہترین ذریعہ ہیں تاکہ سب کو ایک جیسی سمجھ حاصل ہو۔
- مسئلہ حل کرنا: فلو چارٹ کسی عمل میں رکاوٹوں اور غیر مؤثر حصوں کی نشاندہی کرنے میں مدد کرتے ہیں، جس سے مسئلہ حل کرنے اور بہتری کرنے میں مدد ملتی ہے۔
- دستاویزات: یہ نظام اور عمل کے لیے ایک ضروری دستاویزات کے طور پر کام کرتے ہیں جو تربیت اور حوالہ کے مقاصد کے لیے مفید ہوتے ہیں۔

7.3.1.2 فلو چارٹ کی علامتیں

فلو چارٹ کی علامتیں بصری نمائندگی ہوتی ہیں جو کسی عمل یا نظام کے مراحل اور بہاؤ کی وضاحت کرنے کے لیے استعمال ہوتی ہیں جیسا کہ ٹیبل 7.1 میں دکھایا گیا ہے۔

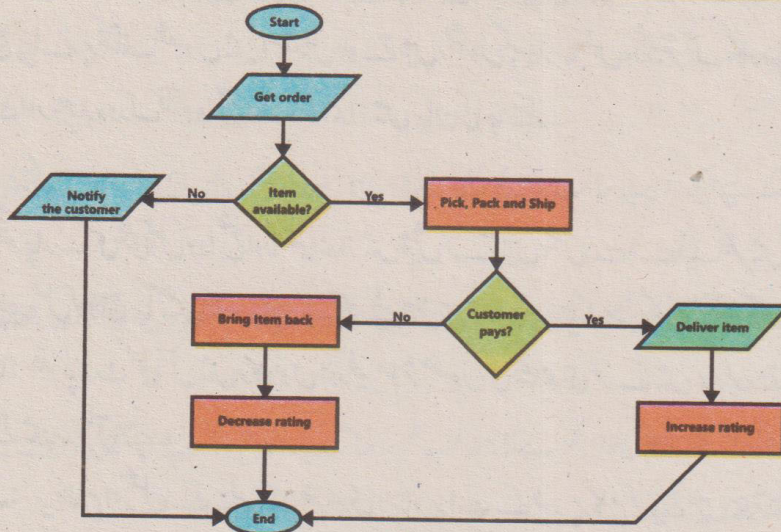
بیان	نام	علامت
کسی عمل کے آغاز یا اختتام کی نمائندگی کرتا ہے۔	بیضوی (ٹرینل)	
ایک عمل، کام یا آپریشن کی نمائندگی کرتا ہے جسے انجام دینے کی ضرورت ہے۔	مستطیل (عمل)	

	متوازی گرام (Input/Output)	ڈیٹا ان پٹ یا آؤٹ پٹ کی نمائندگی کرتا ہے (مثال کے طور پر صارف سے ان پٹ حاصل کرنا یا اسکرین پر آؤٹ پٹ دکھانا۔)
	ڈامنڈ (فیصلہ سازی)	یہ عمل میں ایک فیصلہ کی جگہ کی نمائندگی کرتا ہے جہاں بہاؤ ہاں نہیں سوال، یا، صحیح غلط کی بنیاد پر مختلف ہو سکتا ہے۔
	تیر (فلو لائن)	فلو چارٹ کے اندر بہاؤ کی سمت دکھاتا ہے اور اقدامات کی ترتیب کی نشان دہی کرنے کے لیے علامتوں کو جوڑتا ہے۔

نیل 7.1: فلو چارٹ علامتیں

کیا آپ جانتے ہیں؟

فلو چارٹ کو کمپیوٹر سائنس دانوں جیسے کہ جان وان نیومین اور ہرمن گولڈسٹائن نے کمپیوٹنگ کے ابتدائی دنوں میں مقبول بنایا۔



شکل 7.4: دکان کے آرڈر کے عمل کا فلو چارٹ

سرگرمی

ایک روزمرہ کی سرگرمی کے لیے ایک فلو چارٹ بنائیں، جیسے اسکول کے لیے تیار ہونا۔ جیسے اس میں فیصلہ کن نکات شامل کریں۔ موسم کی بنیاد پر کیا پہننا ہے۔

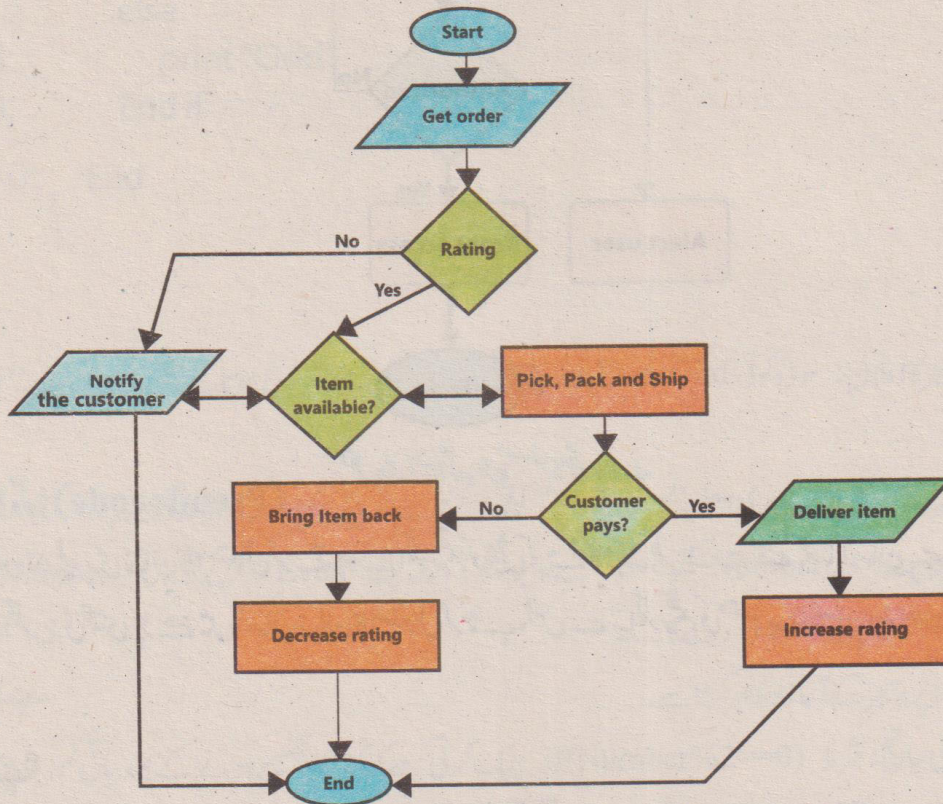
مثال: آپ کے گھر کے قریب ایک دکان: فرض کریں کہ دکان موبائل فون پیغامات کے ذریعے آرڈر لیتی ہے۔ شکل 7.4 میں فلو چارٹ آرڈر پروسیسنگ کے مراحل کا خاکہ پیش کرتا ہے۔ ان پٹ آرڈر ہے، اور آؤٹ پٹ آئٹم ڈیلیوری ہے یا اطلاع ہے اگر آئٹم دستیاب نہیں ہے۔

فیصلے آئٹم کی دستیابی اور گاہک کی ادائیگی کے حوالے سے کیے جاتے ہیں۔ اگر گاہک آئٹم قبول نہیں کرتا ہے یا ادائیگی نہیں کرتا ہے تو، آئٹم کو دکان پر واپس کر دیا جاتا ہے، اور گاہک کی درجہ بندی 1 نمبر کم کر دی جاتی ہے۔ اگر وہ آئٹم کے لیے ادائیگی کرتے ہیں تو گاہک کی درجہ

بندی میں 1 نمبر کا اضافہ کر دیا جاتا ہے۔ اگر سامان دستیاب نہیں ہے تو، دکان گاہک کو مطلع کرتی ہے، بصورت دیگر، دکان سامان اٹھاتی ہے، پیک کرتی ہے اور بھیجتی ہے۔

گاہک کی درجہ بندی کا استعمال کرتے ہوئے فلو چارٹ میں بہتری

نوٹ کریں کہ اگرچہ گاہک کی درجہ بندی شکل 7.4 میں دکھائے گئے فلو چارٹ میں شامل ہے، لیکن اس کا استعمال نہیں کیا گیا ہے۔ آئیے ! فلو چارٹ پر نظر ثانی کریں تاکہ یہ یقینی بنایا جاسکے کہ صرف 0 سے زیادہ درجہ بندی والے صارفین ہی آرڈر کر سکیں۔ نظر ثانی شدہ فلو چارٹ شکل 7.5 میں دکھایا گیا ہے۔

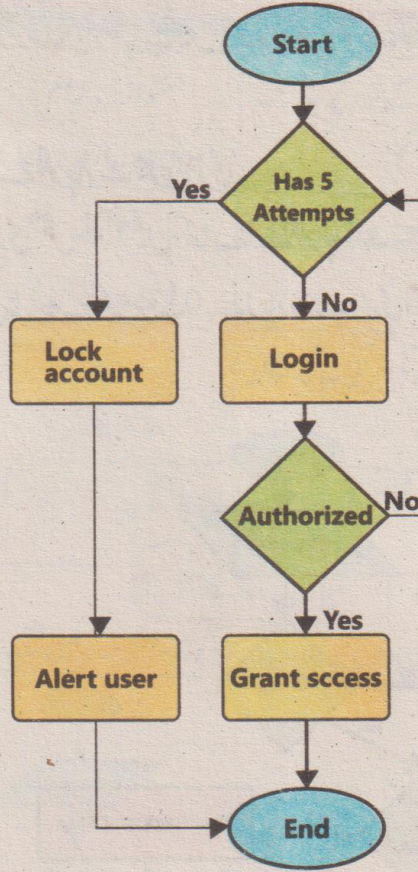


شکل 7.5: گاہک کی درجہ بندی کا استعمال کرتے ہوئے دکان کے آرڈر کا فلو چارٹ

سرگرمی

شکل 7.5 کو ترمیم کریں تاکہ صارف کے ریٹنگز درست حد 0 سے 5 کے درمیان ہوں، بشمول۔ ریٹنگز منفی نہیں ہو سکتی ہیں یا 5 سے زیادہ نہیں ہو سکتی ہیں۔

مثال: لاگ ان سسٹم کے لئے ایک فلو چارٹ، جس میں صارف کا نام اور پاس ورڈ داخل کرنے، ان کی تصدیق کرنے، اور رسائی دینے کے مراحل دکھائے گئے ہیں جیسا کہ شکل 7.6 میں دکھایا گیا ہے۔ صارف زیادہ سے زیادہ پانچ کوششیں کر سکتا ہے۔



شکل 7.6: لاگ ان سسٹم کا فلو چارٹ

7.3.2 سوڈو کوڈ (Pseudocode)

سوڈو کوڈ سادہ اور غیر رسمی زبان کا استعمال کرتے ہوئے الگورتھم کو پیش کرنے کا ایک طریقہ ہے جسے سمجھنا آسان ہے۔ یہ پروگرامنگ کی ساخت کو سادہ انگریزی جیسی پڑھنے میں آسان زبان کا استعمال کرتا ہے، جس سے یہ الگورتھم کی منصوبہ بندی اور وضاحت کے لیے ایک مفید ذریعہ بن جاتا ہے۔

سوڈو کوڈ کیا ہے؟

سوڈو کوڈ کوئی اصل میں کوڈ نہیں ہوتا ہے جسے کمپیوٹر پر چلایا جاسکے، بلکہ الگورتھم کے مراحل کو اس انداز میں بیان کرنا ہے جسے سمجھنا آسان ہو۔ یہ پروگرامرز اور طالب علموں کو کسی مخصوص پروگرامنگ زبان کے ٹیکسٹ میں پھنسے بغیر الگورتھم کی منطق پر توجہ مرکوز کرنے میں مدد کرتا ہے۔ مثال 1: اس بات کا تعین کرنا کہ آیا کوئی عدد جفت ہے یا طاق، پروگرامنگ اور کمپیوٹر سائنس میں ایک بنیادی کام ہے۔ ایک طاق عدد کو 2 سے تقسیم کیا جاتا ہے تو باقی 1 بچ جاتا ہے۔

ذیل میں اس عمل کے لیے سوڈو کوڈ ہے، جس کے بعد ایک وضاحت ہے:

الگورتھم 1 سوڈو کوڈ اس بات کا تعین کرنے کے لیے کہ آیا کوئی نمبر جفت ہے یا طاق ہے۔


```

1: Procedure CheckEvenOdd(number)
2:   Input: number {The number to be checked}
3:   Output: "Even" if number is even, "Odd" if number is odd
4:   Begin
5:     if (number % 2 == 0) then
6:       print "Even"
7:     else
8:       print "Odd"
9:   End if
10: End

```

وضاحت

- (i) پروسیجر کا اعلان: سوڈو کوڈ، CheckEvenOdd کی ڈیکلریشن سے شروع ہوتا ہے جو ایک ان پٹ "Number" لیتا ہے۔
 - (ii) ان پٹ: عمل ایک ویری ایبل "Number" کو قبول کرتا ہے جو چیک کیا جانے والا عدد ہے۔
 - (iii) آؤٹ پٹ: اگر "Even" Number ہے تو پروسیجر آؤٹ پٹ "Even" دے گا اور اگر "Odd" Number ہے تو پروسیجر آؤٹ پٹ "Odd" دے گا۔
 - (iv) آغاز: پروسیجر کے آغاز کو ظاہر کرتا ہے۔
 - (v) شرط کی جانچ: شرط $\text{if}(\text{Number} \% 2 == 0)$ چیک کرتی ہے کہ کیا 2 پر تقسیم ہونے کے بعد باقی نمبر 0 بچتا ہے۔ اس مقصد کے لیے موڈیول آپریٹر (%) استعمال کیا جاتا ہے۔
 - (vi) Even کیس: اگر شرط درست ہے تو طریقہ کار "Even" پرنٹ کرتا ہے۔
 - (vii) Odd کیس: اگر شرط غلط ہے تو طریقہ کار "Odd" پرنٹ کرتا ہے۔
 - (viii) اختتام: عمل کے اختتام کو ظاہر کرتا ہے۔
- مثال 2: اس بات کا تعین کرنا کہ آیا کوئی نمبر مفرد ہے یا نہیں، نمبر تھیوری اور کمپیوٹر سائنس میں ایک بنیادی کام ہے۔ ایک مفرد عدد ایک قدرتی عدد ہے جو 1 سے بڑا ہے اور جو اپنے 1 کے علاوہ کسی پر تقسیم نہیں ہوتا۔ ذیل میں اس عمل کے لیے سوڈو کوڈ ہے، جس کے بعد جس کی وضاحت ہے۔
- الگورتھم 2 سوڈو کوڈ اس بات کا تعین کرنے کے لیے کہ آیا کوئی عدد مفرد ہے۔


```

1: Procedure Is Prime(number)
2:   Input: number {The number to be checked}
3:   Output: True if number is prime, False otherwise
4:   Begin
5:     if (number <= 1) then
6:       return False
7:     end if
8:     for i from 2 to sqrt(number) do
9:       if (number % i == 0) then
10:        return False
11:      end if
12:    end for
13:    return True
14:  End

```

وضاحت

- (i) پروسیجر کا اعلان: سوڈو کوڈ پر ویسیجر "isPrime" کی ڈیکلریشن سے شروع ہوتا ہے جو ایک ان پٹ "Number" لیتا ہے۔
- (ii) ان پٹ: پروسیجر ایک ویری ایبل "Number" کو قبول کرتا ہے جو چیک کیا جانے والا عدد ہے۔
- (iii) آؤٹ پٹ: اگر نمبر پرائم ہے تو طریقہ کار "True" اور دوسری صورت میں "False" آؤٹ پٹ دے گا۔
- (iv) آغاز: پروسیجر کے آغاز کو ظاہر کرتا ہے۔
- (v) ابتدائی چیک: شرط: $\text{if}(\text{Number} \leq 1)$ چیک کرتی ہے کہ آیا عدد 1 سے کم یا برابر ہے۔ اگر یہ درست ہے تو پروسیجر "False" واپس کرتا ہے کیونکہ 1 سے کم یا اس کے برابر نمبر مفرد عدد نہیں ہیں۔
- (vi) ممکنہ تقسیم کاروں کی تلاش کے لیے لوپ: "فار لوپ عدد کی جزرتک چلتا ہے۔ اس کی وجہ یہ ہے کہ کسی عدد کے بڑے Factors کا اس کے چھوٹے Factors کے ملٹی پل ہوتے ہیں جسے پہلے ہی چیک کیا جا چکا ہے۔
- (vii) تقسیم کی جانچ: لوپ کے اندر شرط $\text{if}(\text{Number} \% i = 0)$ چیک کرتی ہے کہ آیا عدد "i" پر پورے طور پر تقسیم ہو رہا ہے یا نہیں۔ اگر یہ "True" ہے تو پروسیجر "False" واپس کرتا ہے کیونکہ عدد کا 1 کے علاوہ ایک اور بھی تقسیم کار ہے۔
- (viii) پرائم کنفرمیشن: اگر لوپ میں کوئی تقسیم کار نہیں پایا جاتا ہے تو، پروسیجر "True" واپس کرتا ہے، جس سے اس بات کی تصدیق ہوتی ہے کہ مفرد عدد ہے۔
- (ix) اختتام: پروسیجر کے اختتام کو ظاہر کرتا ہے۔

سرگرمی

اپنا سوڈو کوڈ بنائیں:

طالب علموں کو چھوٹے گروپوں میں تقسیم کریں اور ہر گروپ کو ایک مختلف آسان مسئلہ تفویض کریں، جیسے کسی فہرست میں زیادہ سے زیادہ تعداد تلاش کرنا یا کسی نمبر کی "Factorial" کا حساب لگانا۔ انھیں اپنے تفویض کردہ مسئلے کے لیے سوڈو کوڈ لکھنے کے لیے کہیں اور پھر اسے کلاس میں پیش کریں۔

کیا آپ جانتے ہیں؟

اصل کوڈ لکھنے سے پہلے سافٹ ویئر کی تیاری میں سوڈو کوڈ کو اکثر استعمال کیا جاتا ہے تاکہ اس بات کو یقینی بنایا جاسکے کہ منطق درست ہے یہ ٹیم کے افراد کے مابین رابطے کو بھی آسان بناتا ہے خاص طور پر جب وہ رابطے مختلف پروگرامنگ لینگویجز استعمال کر رہے ہیں۔

سوڈو کوڈ کیوں استعمال کریں؟

سوڈو کوڈ کے استعمال کے متعدد فوائد ہیں:

- وضاحت: یہ ٹیکسٹ کے بارے میں فکر کیے بغیر الگورتھم کی منطق کو سمجھنے میں مدد کرتا ہے۔
- منصوبہ بندی: یہ پروگرامرز کو اپنے خیالات کا خاکہ پیش کرنے اور الگورتھم کے مراحل کی منصوبہ بندی کرنے میں مدد دیتا ہے۔
- ابلاغ: یہ الگورتھم کے مراحل کو بیان کرنے کا ایک عالم گیر طریقہ ہے، جس سے دوسروں کے ساتھ تبادلہ خیال کرنا آسان ہو جاتا ہے۔

7.3.3 فلو چارٹ اور سوڈو کوڈ میں فرق

فلو چارٹ اور سوڈو کوڈ دونوں الگورتھم کی وضاحت کرنے کے لیے استعمال ہونے والے ذرائع ہیں، لیکن وہ مختلف طریقوں سے ایسا کرتے ہیں۔ ان کے درمیان فرق کو سمجھنے سے آپ کو یہ فیصلہ کرنے میں مدد مل سکتی ہے کہ کس صورت میں کون سا طریقہ استعمال کرنا زیادہ بہتر ہے۔

سوڈو کوڈ

- سوڈو کوڈ الگورتھم کے مراحل کی وضاحت کرنے کے لیے سادہ زبان اور اسٹرکچرڈ فارمیٹ کا استعمال کرتا ہے۔
- یہ ایک کہانی کی طرح پڑھا جاتا ہے، جہاں ہر مرحلہ ترتیب وار لکھا جاتا ہے۔
- سوڈو کوڈ تفصیلی اور بیانیہ طرز میں الگورتھم کے مراحل کو واضح کرتا ہے۔
- یہ خاص طور پر الگورتھم کی ایسی دستاویز سازی کے لیے مفید ہے جسے کسی بھی پروگرامنگ لینگویج کے اصل کوڈ میں آسانی سے تبدیل کیا جاسکتا ہے۔

فلو چارٹ

- فلو چارٹ الگورتھم کے بہاؤ کو ظاہر کرنے کے لیے گرافیکل علامتوں اور تیروں (arrows) کا استعمال کرتے ہیں۔
- یہ ایک فلم دیکھنے کی طرح ہے، جہاں ہر علامت (جیسے مستطیل، ڈائمنڈ اور بیضی) ایک مختلف قسم کے عمل یا فیصلے کو ظاہر کرتی ہے،

اور تیر مرحلوں کے درمیان تعلق اور سمت کی نشاندہی کرتے ہیں۔

(iii) فلو چارٹ ایک بصری انداز میں عمل کو واضح کرتا ہے، جو مجموعی بہاؤ اور ساخت کو سمجھنے کے لیے زیادہ مؤثر ہو سکتا ہے۔

(v) یہ ایک نظر میں الگورتھم میں شامل مراحل اور فیصلوں کی فوری شناخت کرنے کے لیے مفید ہے۔

مثال 3

الگورتھم 3 صارفین کے درست نام اور پاس ورڈ کی جانچ پڑتال کے لیے سوڈو کوڈ پیش کرتا ہے۔

1. **Procedure** CheckCredentials(username, password)
2. **Input:** username, password
3. **Output:** Validity message
4. **Begin**
5. validUsername = "user123" {Replace with the actual valid username}
6. validPassword = "pass123" {Replace with the actual valid password}
7. if (username == validUsername) then
- 8: if (password == validPassword) then
- 9: print "Login successful"
- 10: else
- 11: print "Invalid password"
- 12: end if
- 13: else
- 14: print "Invalid username"
- 15: end if
- 16: End

7.4 الگورتھم کی سرگرمیاں

7.4.1 ڈیزائن اور تشخیص کی تکنیک

الگورتھم کا جائزہ لینا یہ سمجھنے کے لیے اہم ہے کہ یہ مسائل کو کتنے مؤثر طریقے سے حل کرتے ہیں۔ اس سیکشن میں، ہم الگورتھم کا جائزہ لینے کے لیے مختلف تکنیکوں کو دیکھیں گے، ان کے لیے درکار وقت اور جگہ کی پیچیدگیوں پر توجہ مرکوز کریں گے۔

7.4.1.1 ٹائم کمپلیکسٹی

ٹائم کمپلیکسٹی اس بات کی پیمائش کا ایک طریقہ ہے کہ الگورتھم کس قدر تیز یا سست کارکردگی کا مظاہرہ کرتا ہے۔ یہ ہمیں بتاتا ہے کہ ان پٹ کے سائز میں اضافے کے ساتھ الگورتھم کے چلنے کا وقت کیسے تبدیل ہوتا ہے۔ اسے سمجھنے کا ایک آسان طریقہ یہ ہے: فرض کریں کہ آپ کے پاس ناموں کی ایک فہرست ہے، اور آپ ایک مخصوص نام تلاش کرنا چاہتے ہیں۔ اگر آپ کے پاس 10 نام ہوں تو آپ کو فہرست کو دیکھنے میں کچھ سیکنڈ لگ سکتے ہیں۔ اگر آپ کے پاس 100 نام ہوں تو یہ وقت کیا ہوگا؟ یا 1,000 نام ہوں تو؟ جیسے جیسے فہرست لمبی ہوتی جائے گی نام تلاش کرنے میں لگنے والا وقت بڑھتا جائے گا۔ درکار وقت کی پیچیدگی ہمیں اس اضافے کو سمجھنے میں مدد دیتی ہے۔

کیا آپ جانتے ہیں؟

ٹائم کمپلیکسٹی عموماً Big O notation کے ذریعے ظاہر کی جاتی ہے، جیسے $O(n)$ ، $O(\log n)$ یا $O(n^2)$ ۔ یہ ہمیں مختلف الگورتھمز کا موازنہ کرنے میں مدد دیتی ہے تاکہ ہم دیکھ سکیں کہ کون سا تیز ہے!

دلچسپ معلومات:

جب آپ ایک الگورتھم لکھ رہے ہوتے ہیں تو اس بات کو مد نظر رکھیں کہ وہ ٹاسک کتنے مراحل میں مکمل کرتا ہے۔ کم مراحل سے مراد تیز الگورتھم ہے۔

سرگرمی

ایک آسان کام کے بارے میں سوچیں، جیسے کسی فہرست میں سب سے بڑے عدد کو تلاش کرنا۔ ایسا کرنے کے لیے آپ جو اقدامات کریں گے اسے لکھیں۔ اب تصور کریں کہ فہرست میں 10 نمبر ہیں، پھر 100 نمبر ہیں۔ اقدامات کیسے بدلتے ہیں؟

کیا آپ جانتے ہیں؟

کچھ الگورتھم ایک ہی کام کو دوسروں کے مقابلے میں بہت تیزی سے انجام دے سکتے ہیں۔ مثال کے طور پر، 100 اشیاء کی فہرست کو ترتیب دینے میں ایک الگورتھم 1 سیکنڈ جب کہ دوسرا الگورتھم 10 سیکنڈ کا وقت لے سکتا ہے!

7.4.1.2 سپیس کمپلیکسٹی

سپیس کمپلیکسٹی ان پٹ سائز کے لحاظ سے الگورتھم میں استعمال ہونے والی میموری کی مقدار کی پیمائش کرتی ہے۔ ان پٹ کے لیے درکار میموری اور الگورتھم کے ذریعے استعمال ہونے والی کسی بھی اضافی میموری دونوں پر غور کرنا ضروری ہے۔ الگورتھم کو ڈیزائن کرنے اور تشخیص کرنے میں ڈرائی رن اور سیمولیشن جیسی سرگرمیاں شامل ہیں تاکہ یہ یقینی بنایا جاسکے کہ وہ مطلوبہ نتائج فراہم کرتے ہیں۔

7.5 ڈرائی رن (Dry Run)

ڈرائی رن میں کسی بھی غلطی کی نشاندہی کرنے کے لیے نمونے کے ڈیٹا کے ساتھ الگورتھم کو دستی طور پر چلایا جاتا ہے۔

7.5.1 فلو چارٹ کا ڈرائی رن

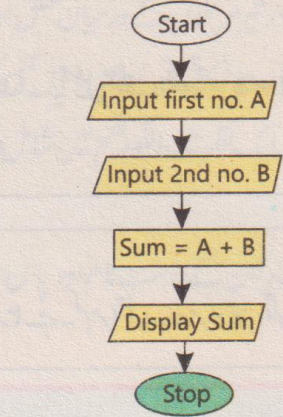
فلو چارٹ کے ڈرائی رن میں دستی طور پر فلو چارٹ سے قدم بہ قدم چلنا شامل ہے تاکہ یہ سمجھا جاسکے کہ کمپیوٹر کا استعمال کیے بغیر الگورتھم کس طرح کام کرتا ہے۔ اس سے کسی بھی منطقی غلطیوں کی نشاندہی کرنے اور کنٹرول کے بہاؤ کو سمجھنے میں مدد ملتی ہے۔

مثال: دو نمبروں کی جمع

دو نمبروں کو جمع کرنے کے لیے تصویر 7.7 میں دیئے گئے فلو چارٹ پر غور کریں: اس فلو چارٹ کو ڈرائی رن کرنے کے اقدامات:

اقدامات:

1. Start
2. Input the first number(e.g. 3)
3. Input the second number(e.g. 5)
4. Add the two numbers($3+5=8$)
5. Output the result(8)
- 6 Stop



شکل 7.7 دو نمبروں کو جمع کرنے کا فلو چارٹ

سرگرمی

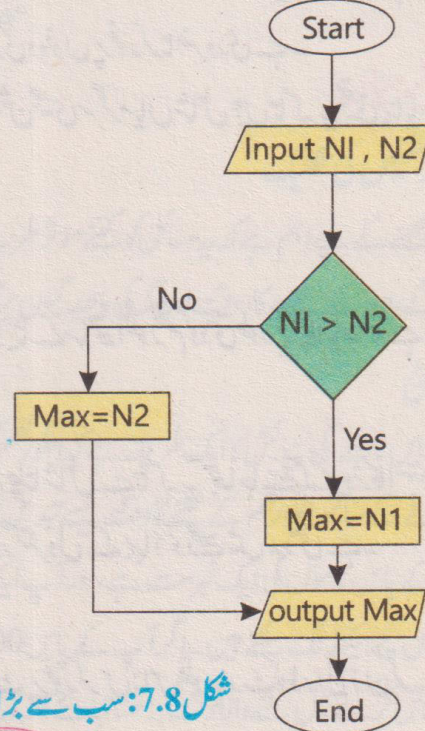
ڈرائی رن کا ایک فلو چارٹ

دو نمبروں میں سے بڑا نمبر معلوم کرنے کے لیے ایک فلو چارٹ بنائیں۔ نمبر 7 اور 4 کے لیے ڈرائی رن کریں۔ ہر قدم اور متغیرات کی قیمتیں (Values of variables) لکھیں۔

7.5.2 سوڈو کوڈ کا ڈرائی رن

سوڈو کوڈ کے ڈرائی رن کا مطلب اسے لائن بہ لائن دستی طور پر چلا کر اس کے عمل کو سمجھا جائے۔ یہ طریقہ الگورتھم کی منطق اور درستی کی تصدیق کرنے میں مدد دیتا ہے۔

اس سوڈو کوڈ کو ڈرائی رن کرنے کے مراحل:



Algorithm 4 FindMax

1. Input: num1, num2
2. if num1 > num2 then
3. max = num1
4. else
5. max = num2
6. end if
7. Output: max

شکل 7.8: سب سے بڑا نمبر ڈھونڈنے کا الگورتھم

کیا آپ جانتے ہیں؟

اپنے کوڈ یا الگورتھم کی ڈرائی رنگ کرنے سے ڈویلپمنٹ کے عمل میں غلطیوں کو جلد پکڑنے میں مدد ملتی ہے، جس سے وقت اور محنت کی بچت ہوتی ہے۔
بہت سے پیشہ ور پروگرامرز اور کمپیوٹر سائنسدان اپنے الگورتھم کو صحیح طریقے سے کام کرنے کو یقینی بنانے کے لیے ڈرائی رنگ کو ڈیبلنگ تکنیک کے طور پر استعمال کرتے ہیں!

7.5.3 سیمولیشن (Simulation)

سیمولیشن ایک ایسا طریقہ ہے جس میں ہم کمپیوٹر پروگراموں کو حقیقی دنیا کے عمل یا نظام کا ماڈل بنانے کے لیے استعمال کرتے ہیں۔ اس سے ہمیں یہ سمجھنے میں مدد ملتی ہے کہ حقیقی زندگی میں انہیں آزمانے کی ضرورت کے بغیر مختلف الگورتھمز کی جانچ کر کے چیزیں کس طرح کام کرتی ہیں۔ سیمولیشن استعمال کرنے کی وجہ:

- (i) ٹیسٹنگ الگورتھم: ہم یہ دیکھنے کے لیے سیمولیشن کا استعمال کر سکتے ہیں کہ ایک الگورتھم مختلف قسم کے ڈیٹا کے ساتھ کتنی اچھی طرح کام کرتا ہے۔ مثال کے طور پر، اگر ہم نمبروں کو ترتیب دینے کا ایک نیا طریقہ آزمانا چاہتے ہیں تو، ہم اسے نمبروں کے مختلف سیٹوں کے ساتھ سیمولیٹ کر سکتے ہیں تاکہ یہ دیکھا جاسکے کہ یہ کتنا تیز اور صحیح ہے۔
- (ii) ممکنہ حالات کا تجزیہ: سیمولیشن ہمیں مختلف حالات پیدا کرنے کی سہولت دیتا ہے تاکہ معلوم ہو سکے کہ ان تبدیلیوں سے کیا نتائج برآمد ہوتے ہیں۔ مثال کے طور پر پودوں کی نشوونما کے بارے میں ایک سائنسی تجربے میں ہم پانی یا سورج کی روشنی کی مختلف مقدار کو سیمولیٹ کر سکتے ہیں تاکہ یہ معلوم کیا جاسکے کہ کون سے حالات پودوں کی نشوونما میں بہترین مدد کرتے ہیں۔

سیمولیشن کے فوائد

- (i) کم لاگت: حقیقی تجربات کرنے کے مقابلے میں سیمولیشن چلانا عموماً سستا اور تیز تر ہوتا ہے۔
- (ii) محفوظ: ہم کسی کو بھی خطرے میں ڈالے بغیر، عمارت میں آگ جیسے خطرناک حالات کی جانچ کر سکتے ہیں۔
- (iii) دہرانے کے قابل: ہم ایک ہی سیمولیشن کو مختلف ترتیب کے ساتھ کئی بار چلا سکتے ہیں یہ دیکھنے کے لیے کہ چیزیں کس طرح تبدیل ہوتی ہیں۔

سیمولیشن کی مثالیں

- (i) موسم کی پیش گوئی: ماہرین موسمیات موسم کی پیش گوئی کرنے کے لیے سیمولیشن کا استعمال کرتے ہیں۔ وہ درجہ حرارت، نمی اور ہوا کی رفتار کے بارے میں ڈیٹا کو ایک کمپیوٹر ماڈل میں ان پٹ کرتے ہیں تاکہ یہ دیکھا جاسکے کہ اگلے چند دنوں میں موسم کس طرح تبدیل ہو سکتا ہے۔
- (ii) ٹریفک کا بہاؤ: شہر کے منصوبہ ساز ٹریفک کی نقل کر سکتے ہیں یہ دیکھنے کے لیے کہ سڑکوں یا ٹریفک لائٹس میں تبدیلیاں گاڑیوں کے بہاؤ کو کس طرح متاثر کر سکتی ہیں۔ اس سے انہیں سڑکوں کو بہتر ڈیزائن کرنے اور ٹریفک جام کو کم کرنے میں مدد ملتی ہے۔

7.6 لاجک آف الگورتھمز فار یزولیشن آف پرابلمز (LARP) کا تعارف

LARP کا مطلب مسئلے کے حل کے لیے الگورتھم کی منطق ہے۔ یہ سیکھنے کا ایک دلچسپ اور انٹرایکٹو طریقہ ہے کہ الگورتھم کو عملی طور پر چلا کر ان کے نتائج دیکھے جاسکتے ہیں۔ اسے کھیل کے میدان کے طور پر سوچیں جہاں آپ مختلف الگورتھم کے ساتھ تجربہ کر سکتے ہیں اور سمجھ سکتے ہیں کہ وہ ڈیٹا پر کس طرح عمل کرتے ہیں۔

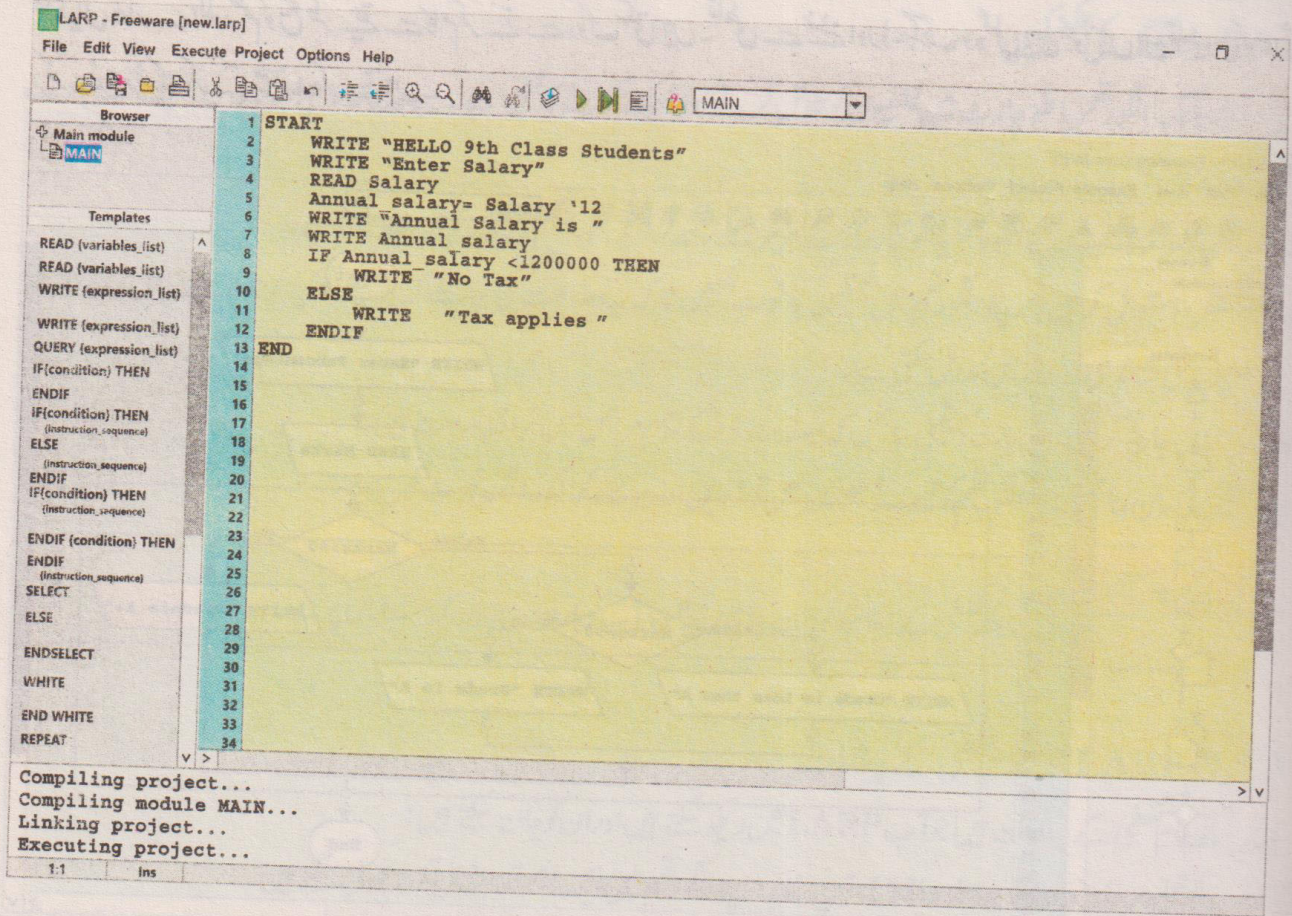
7.6.1 LARP کیوں اہم ہے؟

- (i) یہ جانیں کہ الگورتھم کیسے کام کرتا ہے۔ مثال کے طور پر تصویر 7.9 کو دیکھیں جو ایک الگورتھم کی وضاحت کرتی ہے جو کسی شخص کی سالانہ تنخواہ پر ٹیکس کے اطلاق کا تعین کرنے کے لیے بنایا گیا ہے۔
- (ii) مختلف ان پٹ کے آؤٹ پٹ پر اثرات کا جائزہ لیں۔
- (iii) اپنے الگورتھم لکھنے اور انھیں بہتر بنانے کی مشق کریں۔

7.6.2 الگورتھم لکھنا

LARP کا استعمال کرتے ہوئے الگورتھم لکھنا آسان ہے جس کے ذریعے مسائل کے لیے منطقی حل تیار کیے جاسکتے ہیں۔ LARP ایک واضح ٹیکسٹ استعمال کرتا ہے جو اسٹارٹ کمانڈ سے شروع ہوتا ہے اور اینڈ کمانڈ کے ساتھ ختم ہوتا ہے، اس بات کو یقینی بناتا ہے کہ الگورتھم کے ہر مرحلے پر عمل کرنا آسان ہو۔ اس فریم ورک کے اندر، ہدایات سیدھے سادے انداز میں فراہم کی جاتی ہیں، جیسے پیغامات کو ظاہر کرنے کے لیے write کا استعمال کرنا، ان پٹ قیمتوں کے لیے read اور فیصلہ سازی کے لیے if-then-else کے جیسی سٹریٹجی استعمال کرتے ہیں، پیچیدہ مسائل کو قابل انتظام مراحل میں تقسیم کر کے، LARP سیکھنے والوں کو پیچیدہ کوڈنگ کے ٹیکسٹ پر پھنسے بغیر الگورتھم کے منطقی بہاؤ پر توجہ مرکوز کرنے کی اجازت دیتا ہے۔ یہ طریقہ نہ صرف الگورتھم ڈیزائن کے بنیادی تصورات کو سمجھنے میں مدد دیتا ہے بلکہ واضح اور منطقی تھنکنگ کی حوصلہ افزائی کر کے مسئلہ حل کرنے کی مہارت کو بھی بڑھاتا ہے۔

یہاں ایک سادہ الگورتھم کی مثال ہے یہ چیک کرنے کے لیے کہ آیا کوئی نمبر جفت ہے یا طاق:



شکل 7.9: LARP سافٹ ویئر

7.6.3 LARP میں فلو چارٹ بنانا

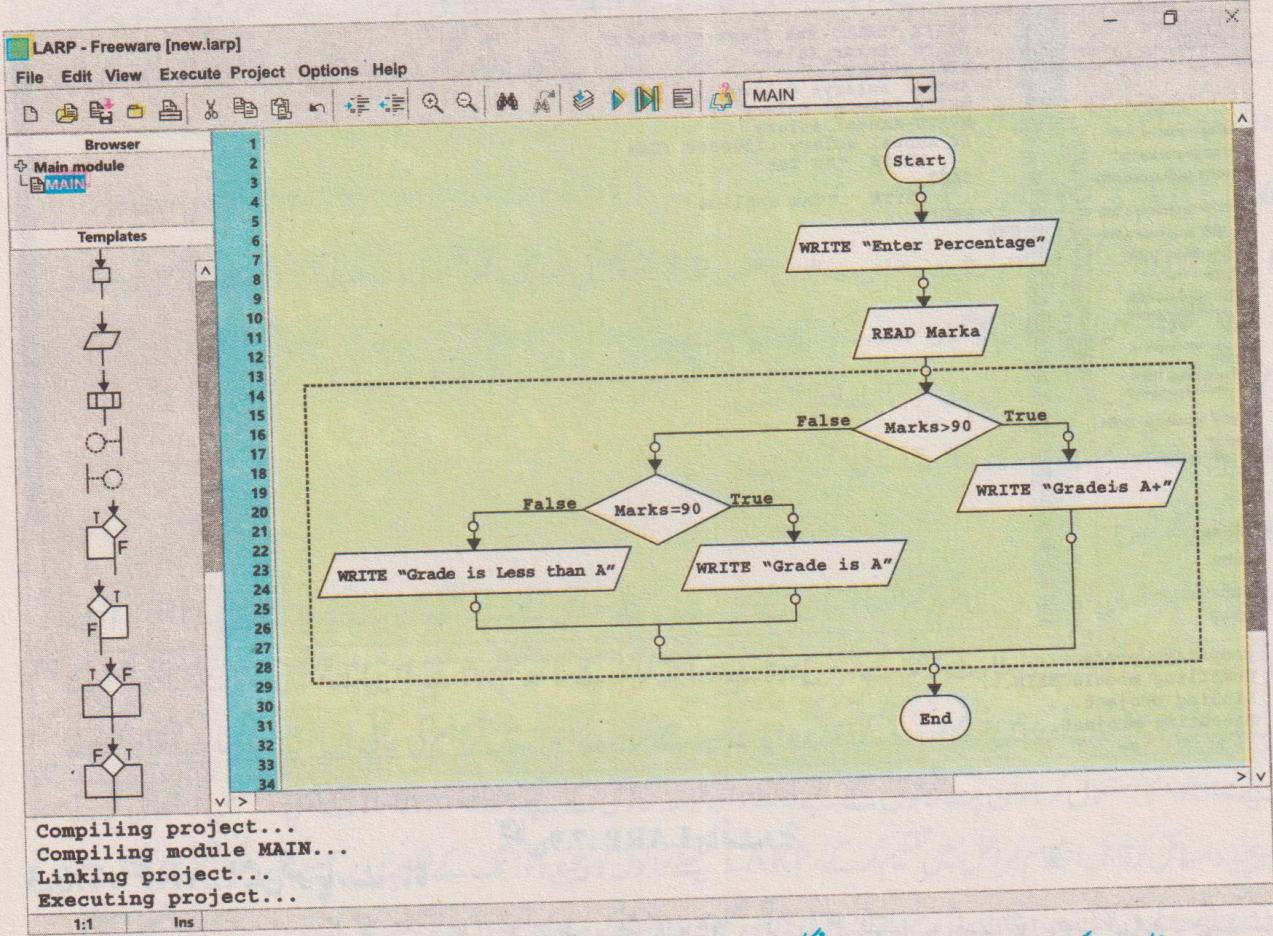
LARP میں فلو چارٹ بنانا الگورتھم کے مراحل کو بصری طور پر پیش کرنا شامل ہے جس میں فلو چارٹ کی علامتوں جیسا کہ پروسیسنگ کے لیے مستطیل، فیصلوں کے لیے ڈائمنڈ اور ان پٹ/آؤٹ پٹ آپریشنز کے لیے پیرالیلوگرامز شامل ہیں۔ ایک بار فلو چارٹ بنانے کے بعد، اس فلو چارٹ کو LARP کے ٹیکسٹ میں ترجمہ کر کے LARP میں چلایا جاتا ہے، جس میں Start، Write، Read، if، else جیسے سیدھے احکامات کا استعمال ہوتا ہے۔

یہ عمل طالب علموں کو اپنے الگورتھم کی منطق کا تصور کرنے اور اس کے قدم بہ قدم عملدرآمد کو دیکھنے کی اجازت دیتا ہے۔ مثال کے طور پر شکل 7.10 کسی طالب علم کے گریڈ کو "A" سے اوپر کرنے یا نہ کرنے کا تعین کرنے کے لیے ایک فلو چارٹ دکھاتا ہے۔ ہم اس کے درست ہونے کی تصدیق کرنے کے لیے فلو چارٹ کو چلا سکتے ہیں۔ یہ دستی نقطہ نظر فلو چارٹ کے کام کے بارے میں سیکھنے اور جاننے میں مدد دیتا ہے۔

7.7 غلطی کی شناخت اور ڈیبگنگ

جب ہم LARP میں الگورتھم لکھتے ہیں یا فلو چارٹ بناتے ہیں تو ہم کبھی کبھی غلطیاں کرتے ہیں جنہیں غلطیاں یا بگز کہا جاتا ہے۔ یہ

غلطیاں ہمارے الگورتھم کو صحیح طریقے سے کام کرنے سے روک سکتی ہیں۔ غلطی سے نمٹنے اور ڈیبگنگ وہ عمل ہیں جو ہمیں ان غلطیوں کو تلاش کرنے اور ٹھیک کرنے میں مدد کرتے ہیں۔



شکل 7.10 LARP 7.10 میں فلو چارٹ

7.7.1 غلطیوں کی اقسام

آپ کو تین اہم قسم کی غلطیوں کا سامنا کرنا پڑ سکتا ہے:

- سنٹیکس (Syntax) کی غلطیاں:** یہ اس وقت ہوتی ہیں جب ہم اپنے الگورتھم یا فلو چارٹ میں کچھ غلط لکھتے ہیں۔ مثال کے طور پر ایک سٹیپ کو چھوڑ دینا یا غلط علامت کا استعمال کرنا۔
- رن ٹائم (Run-time) غلطیاں:** یہ اس وقت ہوتی ہیں جب الگورتھم یا فلو چارٹ پر عمل درآمد کیا جا رہا ہوتا ہے۔ مثال کے طور پر، صفر سے تقسیم کرنے جیسے ناممکن آپریشن کو انجام دینے کی کوشش کرنا۔
- منطقی (Logical) غلطیاں:** یہ الگورتھم کی منطقی غلطیاں ہیں۔ مثال کے طور پر، فیصلے کے مرحلے پر غلط سیمبلز کا استعمال کرنا۔

کیا آپ جانتے ہیں؟

سنٹیکس کی غلطیوں کو تلاش کرنا سب سے آسان ہے کیونکہ LARP ٹول عام طور پر ان کی نشاندہی کرتا ہے۔ تاہم منطقی غلطیوں کو تلاش کرنا سب سے مشکل ہے کیونکہ الگورتھم اب بھی چلتا ہے لیکن صحیح جوابات نہیں دیتا۔

7.7.2 ڈیبنگ کی تکنیکیں

ڈیبنگ الگورتھم یا فلو چارٹ میں غلطیوں کو تلاش کرنے اور ٹھیک کرنے کا عمل ہے۔ یہاں کچھ عام ڈیبنگ کی تکنیک کے بارے میں جانتے ہیں:

- مراحل کا جائزہ لیں: اپنے الگورتھم یا فلو چارٹ کے ہر مرحلے سے گزریں اور دیکھیں کہ یہ کہاں غلط ہوتا ہے۔
- کمٹنس استعمال کریں: اپنے الگورتھم میں کمٹنس یا نوٹ لکھیں تاکہ یہ وضاحت کی جاسکے کہ ہر حصے کو کیا کرنا چاہیے۔ اس سے آپ کو غلطیوں کی نشاندہی کرنے میں مدد مل سکتی ہے۔
- شرائط کی جانچ کریں: اس بات کو یقینی بنائیں کہ فیصلہ سازی کے تمام مراحل میں شرائط درست ہیں۔
- مسئلے کو آسان بنائیں: الگورتھم کو چھوٹے حصوں میں توڑ دیں اور ہر حصے کو الگ الگ ٹیسٹ کریں۔

دلچسپ معلومات:

ہمیشہ غلطی کے پیغام کو غور سے پڑھیں یہ عموماً آپ کو بات دیتے ہیں کہ مسئلہ کہاں پر ہے۔

7.7.3 LARP میں عام غلطی کے پیغامات

- یہاں کچھ عام غلطی کے پیغامات ہیں جو آپ LARP میں دیکھ سکتے ہیں اور ان کو جان سکتے ہیں:
- (i) گمشدہ مرحلہ (Missing Step): آپ شاید اپنے الگورتھم میں ایک اہم قدم شامل کرنا بھول گئے ہیں۔
 - (ii) غیر متعینہ ویری ایبلز (Undefined Variables): آپ ایک متغیر استعمال کر رہے ہیں جس کی ابھی تک وضاحت نہیں کی گئی ہے۔
 - (iii) غلط آپریشن (Invalid Operations): آپ ایک ایسا آپریشن کرنے کی کوشش کر رہے ہیں جس کی اجازت نہیں ہے، جیسے صفر سے تقسیم کرنا۔

سرگرمی

LARP میں ایک سادہ فلو چارٹ بنائیں جو تین نمبروں کی اوسط معلوم کرے۔ اپنے فلو چارٹ میں گرامر کی غلطی، رن ٹائم کی غلطی، اور منطقی غلطی متعارف کروائیں۔ پھر زیر بحث ڈیبنگ تکنیک کا استعمال کرتے ہوئے انھیں ٹھیک کرنے کی کوشش کریں۔

کیا آپ جانتے ہیں؟

”ڈیبنگ“ کی اصطلاح ایک حقیقی بگ یعنی کیڑے (Moth) سے آئی ہے جو ابتدائی کمپیوٹر میں مسائل پیدا کرتا تھا۔ جب اس کیڑے کو ہٹایا گیا تو اس عمل کو ”ڈیبنگ“ کہا گیا۔

خلاصہ

- کمپیوٹیشنل تھنکنگ ایک اہم مہارت ہے جو افراد کو ایسے طریقوں کا استعمال کرتے ہوئے پیچیدہ مسائل کو حل کرنے کے قابل بناتی ہے جو کمپیوٹر سائنس میں شامل عمل کی عکاسی کرتے ہیں۔
- ڈی کمپوزیشن ایک پیچیدہ مسئلہ کو چھوٹے، زیادہ منظم حصوں میں توڑنے کا عمل ہے۔
- پیٹرن کی شناخت میں مسائل کے درمیان اور ان کے اندر مماثلت یا نمونے تلاش کرنا شامل ہے۔
- تجرید میں پیچیدہ مسائل کو چھوٹے اور زیادہ منظم حصوں میں تقسیم کر کے آسان بنانا اور غیر ضروری مسائل کو نظر انداز کرتے ہوئے صرف ضروری تفصیلات پر توجہ مرکوز کرنا شامل ہے۔
- الگورتھم کسی مسئلہ کو حل کرنے یا کسی کام کو مکمل کرنے کے لیے ہدایات کا مرحلہ وار مجموعہ ہے۔
- مسئلہ کو سمجھنا مسئلہ کو حل کرنے میں پہلا اور سب سے اہم مرحلہ ہے، خاص طور پر کمپیوٹیشنل تھنکنگ میں۔
- کسی مسئلہ کو آسان بنانے میں اسے چھوٹے، زیادہ منظم ذیلی مسائل میں توڑنا شامل ہے۔
- بہترین حل کا انتخاب کرنے میں مختلف طریقوں کا جائزہ لینا اور سب سے زیادہ موثر کا انتخاب کرنا شامل ہے۔
- فلو چارٹ کسی عمل یا نظام میں مراحل کی بصری نمائندگی ہیں، جو تیروں سے منسلک مختلف علامتوں کا استعمال کرتے ہوئے دکھایا گیا ہے۔
- سوڈو کوڈ سادہ اور غیر رسمی زبان کا استعمال کرتے ہوئے الگورتھم کی نمائندگی کرنے کا ایک طریقہ ہے جسے سمجھنا آسان ہے۔ یہ پروگرامنگ زبانوں کی ساخت کو سادہ انگریزی زبان پڑھنے کی قابلیت کے ساتھ جوڑتا ہے، جس سے یہ الگورتھم کی منصوبہ بندی اور وضاحت کے لیے ایک مفید آلہ بن جاتا ہے۔
- ٹائم کمپلیکسٹی اس بات کی پیمائش کرنے کا ایک طریقہ ہے کہ الگورتھم کتنی تیز یا سست کارکردگی کا مظاہرہ کرتا ہے۔ یہ ہمیں بتاتا ہے کہ ان پٹ کے سائز میں اضافے کے ساتھ الگورتھم کا چلنے کا وقت کیسے تبدیل ہوتا ہے۔
- سپیس کمپلیکسٹی ان پٹ کے سائز کے سلسلے میں الگورتھم استعمال ہونے والی میموری کی مقدار کی پیمائش کرتی ہے۔ ان پٹ کے لیے ضروری میموری اور الگورتھم کے ذریعے استعمال ہونے والی کسی بھی اضافی میموری دونوں پر غور کرنا ضروری ہے۔
- ڈرائی رن میں کسی بھی غلطی کی نشاندہی کرنے کے لیے نمونے کے اعداد و شمار کے ساتھ الگورتھم کے ذریعے دستی طور پر جاننا شامل ہے۔
- سیمولیشن اس وقت ہوتا ہے جب ہم حقیقی دنیا کے عمل یا نظام کا ماڈل بنانے کے لیے کمپیوٹر پروگراموں کا استعمال کرتے ہیں۔
- LARP کا مطلب ہے لائیو الگورتھم چلانا اور مشق کرنا۔ یہ سیکھنے کا ایک دلچسپ اور انٹرایکٹو طریقہ ہے جو سیکھنے والوں کو الگورتھم کو عملی طور پر چلانے اور ان کے نتائج دیکھنے میں مدد دیتا ہے۔
- ڈیبگنگ الگورتھم یا فلو چارٹ میں غلطیوں کو تلاش کرنے اور ٹھیک کرنے کا عمل ہے۔

مشق

1۔ درست جواب کا انتخاب کریں:

(i) مندرجہ ذیل میں سے کمپیوٹیشنل تھنکنگ کی بہترین تعریف کون سی ہے؟

- (الف) صرف ریاضیاتی حساب کتاب کا استعمال کرتے ہوئے مسائل کو حل کرنے کا ایک طریقہ ہے۔
- (ب) ایک مسئلہ حل کرنے کا طریقہ جو منظم، الگورتھم اور منطقی تھنکنگ کو استعمال کرتا ہے۔
- (ج) ایک تکنیک جو خاص طور پر کمپیوٹر پروگرامنگ میں استعمال ہوتی ہے۔
- (د) ایک ایسا طریقہ جو حقیقی دنیا کی اپیلی کیشنز کو نظر انداز کرتا ہے۔

(ii) کمپیوٹیشنل تھنکنگ میں مسئلہ کی تقسیم کیوں اہم ہے؟

- (الف) یہ مسائل کو چھوٹے، زیادہ منظم حصوں میں تقسیم کر کے آسان بناتا ہے۔
- (ب) یہ مزید تفصیلات شامل کر کے مسائل کو پیچیدہ بناتا ہے۔
- (ج) یہ مسئلہ کو حل کرنے کی ضرورت کو ختم کرتا ہے
- (د) یہ صرف سادہ مسائل کے لیے مفید ہے۔

(iii) نمونے کی شناخت میں کیا شامل ہیں؟

- (الف) مسائل کے اندر مماثلت تلاش کرنا اور ان کا فائدہ اٹھانا
- (ب) تکرار کے عناصر کو نظر انداز کرنا
- (ج) مسائل کو چھوٹے چھوٹے حصوں میں تقسیم کرنا
- (د) تفصیلی الگورتھم لکھنا

(iv) کون سی اصطلاح بنیادی خیال پر توجہ مرکوز کرنے کے لیے تفصیلات کو نظر انداز کرنے کے عمل پر مشتمل ہے؟

- (الف) ڈی کمپوزیشن
- (ب) پیٹرن کی شناخت
- (ج) تجدید
- (د) الگورتھم ڈیزائن

(v) مندرجہ ذیل میں سے کون سا کمپیوٹیشنل تھنکنگ کا بنیادی اصول ہے؟

- (الف) مسئلے کی تفہیم کو نظر انداز کرنا
- (ب) مسئلے کو آسان بنانا
- (ج) حل کے ڈیزائن سے گریز کرنا
- (د) بے ترتیب حل کو نافذ کرنا

(vi) الگورتھم کیا ہوتے ہیں:

- (الف) ڈیٹا کی فہرستیں
- (ب) گرافیکل نمائندگی
- (ج) کسی مسئلے کو حل کرنے کے لیے مرحلہ وار ہدایات
- (د) بار بار دہرائے جانے والے پیٹرن

(vii) کمپیوٹیشنل تھنگ کے مطابق مسئلہ حل کرنے میں مندرجہ ذیل میں سے کون سا پہلا قدم ہے؟

- (الف) حل لکھنا
(ب) مسئلے کو سمجھنا
(ج) فلو چارٹ ڈیزائن کرنا
(د) حل کا انتخاب کرنا

(viii) فلو چارٹ کس لیے استعمال کیے جاتے ہیں:

- (الف) پروگرام کوڈ کرنے کے لیے
(ب) گرافک طور پر الگورتھم کو ظاہر کرنے کے لیے
(ج) ریاضیاتی مساوات کو حل کرنے کے لیے
(د) پیٹرنز کی ساخت کے لیے

(ix) سوڈوکوڈ کیا ہے؟

- (الف) فلو چارٹ کی ایک قسم
(ب) سادہ زبان کا استعمال کرتے ہوئے الگورتھم کی ایک اعلیٰ سطحی وضاحت
(ج) ایک پروگرامنگ زبان
(د) ایک ڈیبنگ ٹول

(x) فلو چارٹ کوڈ رائی رن کرنے میں کیا شامل ہوتا ہے؟

- (الف) پروگرامنگ زبان میں کوڈ لکھنا
(ب) نمونہ ڈیٹا کے ساتھ فلو چارٹ کی جانچ
(ج) فلو چارٹ کو سوڈوکوڈ میں تبدیل کرنا
(د) فلو چارٹ کی تفصیلات کو نظر انداز کرنا

2۔ مختصر جواب دیں۔

- (i) کمپیوٹیشنل تھنگ کی وضاحت کریں۔
- (ii) کمپیوٹیشنل تھنگ میں Decomposition کیا ہے؟
- (iii) Pattern Recognition کو ایک مثال کے ساتھ بیان کریں۔
- (iv) مسئلہ حل کرنے میں تجدید اور اس کی اہمیت کی وضاحت کریں۔
- (v) الگورتھم کیا ہے؟
- (vi) مسئلہ کی تفہیم کمپیوٹیشنل تھنگ میں کس طرح مدد کرتی ہے؟
- (vii) فلو چارٹ کیا ہیں اور وہ کس طرح استعمال ہوتے ہیں؟
- (viii) سوڈوکوڈ کے مقصد کی وضاحت کریں۔
- (ix) آپ فلو چارٹ اور سوڈوکوڈ کے درمیان فرق کیسے کرتے ہیں؟
- (x) ڈرائی رن کیا ہے اور یہ کیوں ضروری ہے؟
- (xi) LARP کیا ہے اور الگورتھم سیکھنے میں اس کی اہمیت کی وضاحت کریں۔
- (xii) دو ڈیبنگ تکنیکس لکھیں اور ان کی وضاحت کریں۔

3- تفصیلی سوالات:

(i) ایک الگورتھم لکھیں جو کسی طالب علم کے حاصل کردہ نمبروں کی بنیاد پر گریڈ تفویض کرے۔ درج ذیل گریڈنگ سسٹم کا استعمال کریں:

• 90 اور اس سے زیادہ : A+

• 80 سے 89 : A

• 70 سے 79 : B

• 60 سے 69 : C

• 60 سے نیچے : F

(ii) وضاحت کریں کہ آپ ایک پیچیدہ کمپیوٹیشنل مسئلے کو حل کرنے کے لیے الگورتھم ڈیزائن کے طریقوں، جیسے فلو چارٹ اور سوڈو کوڈ کا

استعمال کیسے کریں گے۔ ایک تفصیلی مثال کے ساتھ اپنی وضاحت کو بیان کریں۔

(iii) کمپیوٹیشنل تھنکنگ کی وضاحت کریں اور جدید مسائل کو حل کرنے میں اس کی اہمیت کو بیان کریں۔ مثالیں فراہم کریں کہ مختلف

شعبوں میں کمپیوٹیشنل تھنکنگ کو کس طرح لاگو کیا جاسکتا ہے۔

(iv) کمپیوٹیشنل تھنکنگ میں تقسیم کے تصور پر تبادلہ خیال کریں۔ یہ کیوں اہم ہے؟

(v) کمپیوٹیشنل تھنکنگ کے حوالے سے پیٹرن کی شناخت کی وضاحت کریں۔ یہ مسئلہ حل کرنے میں کس طرح مدد کرتی ہے؟

(vi) کمپیوٹیشنل تھنکنگ میں Abstraction کیا ہے؟ اس کی اہمیت پر تبادلہ خیال کریں اور اس کی مثالیں فراہم کریں کہ پیچیدہ

مسائل کو آسان بنانے کے لیے کس طرح اس کا استعمال کیا جاسکتا ہے۔

(vii) الگورتھم کیا ہے اس کی وضاحت کریں اور کمپیوٹیشنل تھنکنگ میں اس کے کردار کی وضاحت کریں۔ کسی مخصوص مسئلے کو حل کرنے کے

لیے الگورتھم کی تفصیلی مثال فراہم کریں اور متعلقہ فلو چارٹ بنائیں۔

(viii) الگورتھم ڈیزائن کے طریقوں کے طور پر فلو چارٹ اور سوڈو کوڈ کا موازنہ کریں۔ ہر طریقہ کار کے فوائد اور نقصانات پر تبادلہ خیال کریں، اور

ایسی مثالیں فراہم کریں جہاں ایک کو دوسرے پر ترجیح دی جاسکتی ہے۔

(ix) فلو چارٹ اور سوڈو کوڈ دونوں کے حوالے سے ڈرائی رن کے تصور کی وضاحت کریں۔ ڈرائی رن کرنے سے الگورتھم کے درست

ہونے کی تصدیق کرنے میں کس طرح مدد ملتی ہے؟

(x) LARP کیا ہے؟ الگورتھم سیکھنے اور مشق کرنے میں اس کی اہمیت پر تبادلہ خیال کریں۔

(xi) LARP کمپیوٹیشنل تھنکنگ کے اصولوں کی تفہیم اور اطلاق کو کس طرح بہتر بناتا ہے؟ ایسی مثال دیں کہ جہاں LARP کو

الگورتھم کو بہتر بنانے کے لیے استعمال کیا جاسکتا ہے۔